Hard Exploration Robotic Task in an Uneven Terrain via Reinforcement Learning

Eduardo Gutierrez
West Virginia University
eg0013@mix.wvu.edu

***Abstract*—Reinforcement learning in real-world environments is still a challenge since the insufficient rewards of the environment makes the process of learning inefficient. The sparse rewards doesn't give the agent enough feedback and it´s forced to use hard exploration methods. In this work, the performance of the Go-explore and the Deep Q Network (DQN) algorithms are analyzed in the task of training a mobile robot end-to-end to solve a navigation task relying solely on camera input and without the access to location information. The results showed that the Go-explore algorithm was not compatible with this environment and the robot couldn't solve the task using it. Moreover, the DQN algorithm showed that input obtained by the depth camera was better than the Red-Green-Blue camera (RGB) and that a larger range of epsilon would solve the problem more efficiently. The work showed that after 7000 episodes, the robot was able to reach an stable reward value of 0.75 and that it reached a maximum reward value of 0.76 at 11000 episodes. It also showed that as the number of episodes increase, the number of steps per episode decrease.**

***Index Terms*—real-world environment, sparse rewards, hard exploration, navigation, Go-explore algorithm, DQN algorithm.**

I. Introduction

Reinforcement learning is the area of machine learning that analyzes how an agent takes different actions in an environment to maximize the cumulative reward [14]. Applying reinforcement learning algorithms to real-world environments has always been a challenge because of the sparse/deceptive rewards this kind of environments have. The problem with the sparse reward is that the agent doesn't receive enough feedback from the environment [14]. The problem with deceptive reward is that the feedback received from the environment is wrong or false (locally) and it doesn't let the agent to get to the optimal policy [14]. In other words, real-world environments have a lack of rewards and they are sometimes confusing. This is the reason why it is really difficult to find methods that efficiently explore and exploit this type of environments. This idea of exploration consists of exploring the environment to find high reward behaviors [15]. The idea of exploitation consists of exploiting the knowledge to maximize the cumulative reward [15].

This problem of how to explore/exploit a real-world environment has not been solved yet but there are different methodologies and algorithms that try to train agents in these circumstances. For example, one of the most popular methods it´s called reward shaping [1]. This method consists of populating the amount of rewards in the environment, creating a net of small rewards that would guide the agent in the process of learning [1,2]. We can stand out another popular method called curiosity-driven exploration. In this method, the agent receives bonus rewards that stimulates its curiosity [3]. In other words, the agent receives an intrinsic reward signal to explore its environment [3]. This would help him to find the sparse reward quicker and learn the optimal policy more efficiently. There is also the possibility of combining both: a small-net shaped extrinsic reward and a curiosity-based intrinsic reward [4]. A different method would be the count-based exploration. In this method, the agent is penalized every time he visits a state already known, state visitation counts [5,6]. This would force the agent to explore the environment before exploiting it to minimize the penalty and maximize the cumulative reward [5,6]. There are more methods of exploring the environment like the random network distillation or the maximizing empowerment, but they are not as popular as the ones already mentioned. The random network distillation is based on a fixed random target network and a prediction network, where the intrinsic reward would be the loss of the prediction network [7]. The maximizing empowerment is based on the long-term goal of the agent, maximizing its control on the environment [8]. Another method that has shown promising results is the Go-explore algorithm. This algorithm firs remembers previously visited states, then returns to a promising state without exploration) and finally explores from it [9].Lastly, the DQN method is really useful in this type of tasks. This algorithm first stores samples with the current policy and then randomly performs an action [13]. With this experience is able to update the Q network and repeat the process [13].

The picture below presents a environment composed by two mazes. The green part represents the intrinsic reward available. The purple part represents the area explored by the algorithm. The white part represents the part already explored where there is not more intrinsic reward. The algorithm used in this case would start exploring the left side of the maze. It would not complete the intrinsic reward collection on that side but it would start exploring the right side. After completely exploring the right side, it would need to move to the left side to completely explore that part and solve the task. Due to the lack of explicit memory, the algorithm is not able to come back to the left side and that part of the maze remains unexplored and therefore, the task is not solved. We can observe in this example how the randomness of the actions of an algorithm to try to explore new alternatives can be counterproductive if

its not appropriate for it.

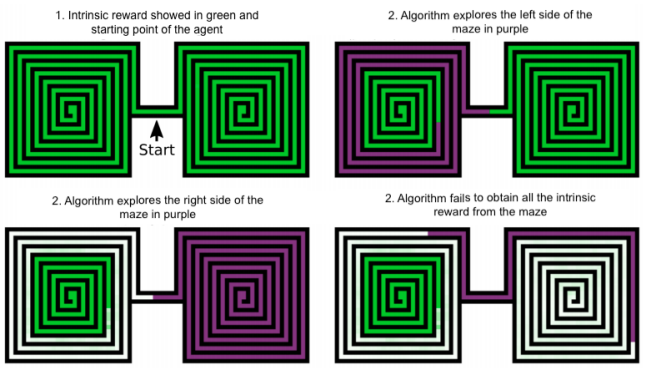


Fig. 1: Algorithm failing to obtain all the intrinsic reward possible[9]

In this paper, the performance of the Go-explore algorithm will be tested and compared wit the performance of the DQN algorithm. Both algorithms will be analyzed to show which one would solve more efficiently the navigation task in the real-world environment.

II. Problem Definition

A. Navigation Task

The task for this work is to train a wheeled robot end-to-end to solve a navigation task relying solely on camera input and without the access to location information[12]. The uneven terrain will have a monolith centered on it. The maximum reward will be obtained by the robot if it approaches the monolith within a radius of 40.0 cm [10]. The environment is reset whenever the robot completes the task or whenever it approaches the boundary of the environment [10]. After each reset, the robot is moved to a random position with a random orientation to complete the task again [10].
The learning algorithm implemented in the robot will help him to obtain the maximum reward and get to the monolith after every reset with the minimum amount of actions. This is a hard-explore problem because of the lack of feedback that the robot will receive from the environment. The agent will only know if he is doing good or bad if he either gets close to the centered monolith or goes out of the boundaries of the environment.

B. Real-World Environment

The virtual environment that is going to be used to test and train the agent with our algorithm is called Off World Gym. Off World Gym is a collection of real-world environments specifically designed to test reinforcement learning algorithms for robots. They have two environments fully developed and two other environments coming. The ”OffWorldMonolithDiscrete” is defined by a wheeled robot on an uneven terrain with four discrete actions: left, right, forward and back [10]. The ”OffWorldMonolithContinous” is defined by a wheeled robot on an uneven terrain with two continuous actions: angular velocity and linear velocity [10]. The other two environments that are sill in progress are the ”OffWorldMonolithObstacleDiscrete” and the ”OffWorldMonolithObstacleContinous” [10]. The difference between these two and the previous ones is the presence of different obstacles in the environment.The scrips where these environments are defined can be found at their GitHub repository [11].
In this work, the algorithm will be implemented in the ”OffWorldMonolithDiscrete” environment. This environment simulates a real-world environment. It is a $3 \times 4 \times 2$ meters uneven terrain designed to visually emulate the lunar surface with a monolith in the center of it[10]. The wheeled robot that will complete the navigation task has four discrete actions: left, right, forward and back. A comparison between the real-world environment (left) and the ”OffWorldMonolithDiscrete” simulated environment (right) can be graphically observed in the figure below.

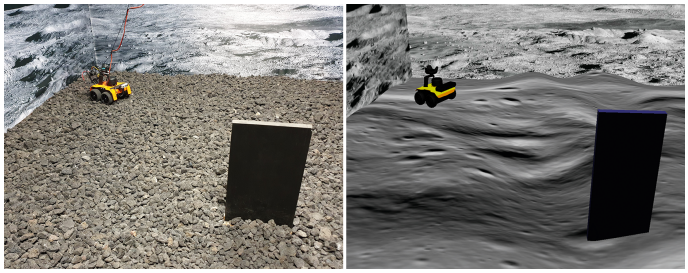


Fig. 2: ”OffWorldMonolithDiscrete” Environment [10]

Another thing to mention is the sensors that will be available for the robot to use in the ”OffWorldMonolithDiscrete” environment. The robot will only have the input from the camera to use [10]. For this robot we have two different camera input: detph camera and RGB camera. A graphic representation and comparison of the different images collected by the the robot can be observed in the figure below. From left to right it can be observed: real word view, RGB images and depth image.

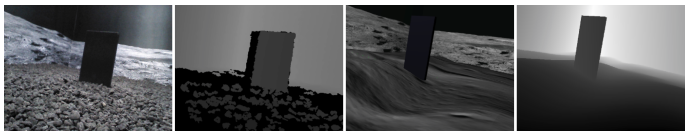


Fig. 3: Images from the robot´s camera [10]

III. Methodology

A. Go-explore algorithm

The Go-explore methodology consists on two phases. The first phase is called ”Explore until solved” [9]. Here the agent will take several actions to explore completely the environment. First, it will select a state from the archive[9]. After that, it will go to that state and explore from that state[9]. Finally, it will update the archive[9]. This phase will be repeated by the agent until every single state has been explored and the environment is completely solved. After this, the second phase called ”Robustify” will start[9]. This phase

is not always needed but consist on running imitation learning on the best trajectory. The diagram shown below explains graphically the methodology of the Go-explore algorithm. The base code for the Go-explore algorithm can be found at the GitHub repository [16]. The only thing that needs to be done here is implement this algorithm to our environment and show and analyze the results obtained.

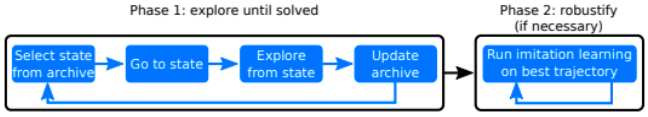


Fig. 4: Methodology of the Go-explore algorithm [9]

B. DQN algorithm

The DQN algorithm is similar to a Q-Learning algorithm but with a different agent's brain. For DQN, the brain would be a neural network instead of a Q-Table. The first thing that composes the architecture of the DQN algorithm is the input. This input is the raw images abstracted from the environment. After collected this input it will be received by the neural network. This neural network is composed by a number of different hidden layers that will process the input [13]. After this hidden layers, the data will go to the output layer, the last part of the system. This output layer maps directly to the different actions that the agent could take, giving us the Q-Values [13]. These Q-Values will tell the agent which actions it should take next [13]. The diagram shown below explains graphically the architecture of the DQN algorithm.

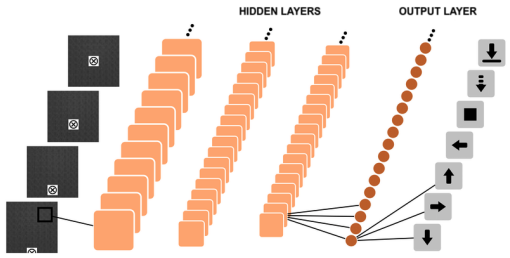


Fig. 5: Architecture of the DQN algorithm [13]

Another thing that needs to be mentioned is that the raw images obtained by the environment can come from the depth camera or the RGB camera. Both will be analyze to observe the performance of the algorithm with different inputs. Moreover, the value of epsilon can be modified to obtain different performances and see which value would solve the task more efficiently.

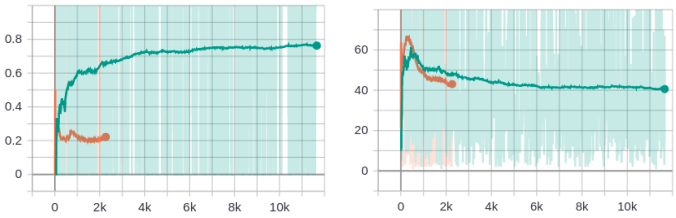
IV. Experiments

A. Go-explore algorithm

Different versions of the go-explore algorithm were tried to be implemented in the Off World Gym environment. It was already shown in related works that the Go-exlore algorithm was able not only to perform hard explorations missions but to outperform DQN and other type of algorithms in other types of environments[16, 17, 18]. Nevertheless, neither the original version [16], Rude's version [17] or Adeikalam's version [18] were compatible with the Off World Gym environment. This work showed that the Go-explore algorithm and the Off World Gym environment are not compatible and that this algorithm cannot work in this environment. No useful analyzable results were obtained from this algorithm. Therefore, we can conclude that this experiment was a failure in terms of obtaining analyzable and useful results.

B. DQN algorithm

The DQN algorithm successfully trained the agent in the Off World environment. The code used is explained and posted on the GitHub repository [19]. This code follows the methodology already explained in the previous section. The algorithm was tested with the two different input images available from the robot. The result of this experiment is shown in the graphs below.



(a) Reward vs Episode (b) Steps vs Episode

Fig. 6: Depth Images vs RGB Images

In both graphs, the green line represents the data obtained with the depth images input and the orange line represents the data obtained with the RGB images input. In the graph on the left side, the x-axis represents the amount of episodes and the y-axis the reward obtained. It can be clearly observed that the agent reaches a higher reward value with the input of depth images. In the graph on the right side, the x-axis represents the amount of episodes and the y-axis the number of steps per episode. It can be clearly observed that as the amount of episodes increase, the amount of steps per episode decrease. This conclusion means that the agent needs less episodes to complete the task and therefore is more efficient. The table below summarizes the key values obtained from this experiment.

TABLE I: Data Summary Depth - Images and RGB Images

-	Depth Images	RGB Images
Stable Reward	0.75	0.20
Number of Episodes	7000	1500
Max Reward	0.76	0.22
Number of Episodes	11000	2000
Stable Steps	41	43
Number of Episodes	7000	2200
Min Steps	40	43
Number of Episodes	11000	2000

The stable reward value is the reward value were the graph starts getting stable and almost horizontal. The number of episodes below refers to the number of episodes needed to reach this point. The max reward value refers to the maximum reward value obtained by the algorithm during the whole simulation and the number of episodes below refers to the amount of episodes needed to obtain this value. The stable steps value is the number of steps were the graph starts getting stable and almost horizontal. The number of episodes below refers to the number of episodes needed to reach this point. The min steps value refers to the minimum number of steps obtained by the algorithm during the whole simulation and the number of episodes below refers to the amount of episodes needed to obtain this value.
Another parameter that was modified and analyzed was the epsilon value. The influence of this parameter on the learning process is shown in the diagram below.

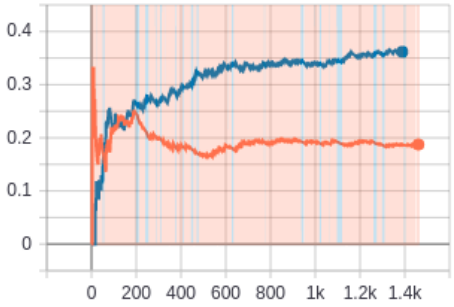


Fig. 7: Reward vs Episode (Epsilon)

In the graph above, the blue line represents the data obtained with the epsilon range from 0.6 to 0.8 and the orange line represents the epsilon range from 0.1 to 0.3. In the graph, the x-axis represents the amount of episodes and the y-axis the reward obtained. Even though both graphs give a reward value smaller than the original epsilon range from 0.1 to 0.8. It can be observe that the higher range gives better results than the lower range.
It was also possible to obtain the different images that the algorithm was processing. In the figure below, it can be observed the processed depth images after 11000 episodes (left) and the processed RGB images after 2200 episodes (right).

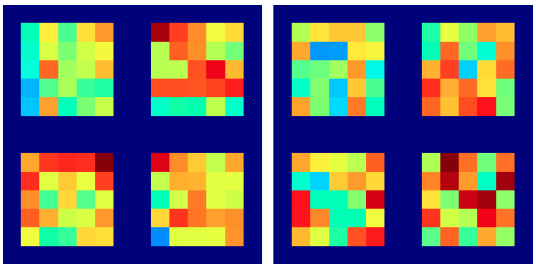


Fig. 8: Processed Images by the algorithm

V. Conclusions

To sum up, the task for this work was to train a wheeled robot end-to-end to solve a navigation task relying solely on camera input and without the access to location information. The real-world environment where the algorithm trained the robot was the ”OffWorldMonolithDiscrete” Environment from Off World Gym. This environment consisted on a uneven squared terrain with a monolith centered on it. The goal of the robot was to get as close as possible to the monolith. Due to the sparse reward of the environment, it made this task a hard exploration mission.
The performance of two different algorithms were originally planned to be analyzed in this real-world environment. On one hand, the Go-explore algorithm was not compatible with the environment and the robot was not able to learn to perform its mission using the algorithm. On the other hand, the DQN algorithm successfully trained the robot to perform the mission. Since only the DQN algorithm seem to be compatible with the Off World Gym, there is no comparison between different algorithms performances. Nevertheless, the parameters and inputs of the DQN algorithm were changed to obtain the most efficient performances and obtain better results. It can be observed from the results that the depth images were a better input since they clearly improved the learning of the robot. Moreover, a larger range of epsilon would make the algorithm more efficient. Lastly, we can conclude that the DQN algorithm reaches a stable reward value of 0.75 after 7000 episodes and that it reaches a maximum reward value of 0.76 after 11000 episodes. It also showed that as the number of episodes increase, the number of steps per episode decrease. This conclusion means that the agent needs less episodes to complete the task and therefore is more efficient. A way to improve these results would be creating a different algorithm similar to the Go-explore algorithm. This algorithm would be compatible with the Off World Gym environment and would outperform other algorithms performances on it, as it was already demonstrated in other environments.

Acknowledgment

This work was supported by Dr. Ali Baheri, professor of the course Reinforcement Learning and Control at West Virginia University.

References

[1] S.M. Devlin and D. Kudenko: Dynamic Potential-Based Reward Shaping (Jun 2012), http://eprints.whiterose.ac.uk/75121/.
[2] O. Marom and B. Rosman: Belief reward shaping in reinforcement learning. In:Thirty-Second AAAI Conference on Artificial Intelligence (2018).
[3] D. Pathak, P. Agrawal, A.A. Efros and T. Darrell: Curiosity-driven exploration by self-supervised prediction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (2017).
[4] A. Jeewa, A. Pillay and E. Jembere: Directed curiosity-driven exploration in hard exploration, sparse reward environments (2017), http://ceur-ws.org/Vol-2540/FAIR2019paper42.pdf.
[5] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton and R. Munos: Unifying count-based exploration and intrinsic motivation. In Advances in neural information processing systems (2016).

[6] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck and P. Abbeel: Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning (2017), http://papers.nips.cc/paper/6868-exploration-a-study-of-count-based-exploration-for-deep-reinforcement-learning.pdf.

[7] Y. Burda, H. Edwards, A. Storkey and O. Klimov: Exploration by random network distillation. arXiv preprint arXiv:1810.12894 (2018).

[8] K. Gregor, D.J. Rezende and D. Wierstra: Variational intrinsic control. arXiv preprint arXiv:1611.07507 (2016).

[9] A. Ecoffet, J. Huizinga, J. Lehman, K.O. Stanley and J. Clune: Go-explore: a new approach for hard-exploration problems. arXiv preprint arXiv:1901.10995 (2019).

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

[11] A. Kumar, T. Buckley, J.B. Lanier, Q. Wang, A. Kavelaars, I. Kuzovkin: OffWorld Gym GitHub Repository (2020), https://github.com/offworld-projects/offworld-gym.

[12] A. Baheri: Final Project Manual for MAE 693 (2021).

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmillerhttps: Playing Atari with Deep Reinforcement Learning (2013), arxiv.org/pdf/1312.5602v1.pdf.

[14] K. Daaboul:Reinforcement Learning: Dealing with Sparse Reward Environments (2020), https://medium.com/@m.k.daaboul/dealing-with-sparse-reward-environments-38c0489c844d.

[15] S. Sinha: The Exploration–Exploitation Dilemma:A Review in the Context of Managing Growth of New Ventures (2015), https://journals.sagepub.com/doi/pdf/10.1177/0256090915599709.

[16] A. Ecoffet: Uber research, Go-explore (2020), https://github.com/uber-research/go-explore.

[17] R. Rudes: Minimal Go-explore GitHub Repository (2021), https://github.com/Ryan-Rudes/minimal-goexplore.

[18] A. Adeikalam: Go-explore GitHub Repository (2020), https://github.com/Adeikalam/Go-Explore.

[19] E. Gutierrez: RLC2021 Final Project GitHub Repository (2021), https://github.com/eguti98/RLC2021.