International Journal of Scientific & Engineering Research, Volume 11, Issue 9, September-2020
ISSN 2229-5518

AI-BASED MAZE SOLVER

Harikrishnan Bejishkumar

Abstract— Artificial Intelligence had been changing the way humans think for several year till now. The first step towards such an intelligent turnover took place back in 1950's. This was not that popular at the time, due to inefficiency of processing power and insufficient data on hand. But as the data and need of processing, made AI a huge developmental step in the modern world. This paper puts forward an AI system, using python, Kivy and its associated libraries, that would have a general demand in the field of science and technology. Here is an application of artificial intelligence, precisely a problem that is common in the field of robotics: smooth navigation of a robot through any area with obstacles, without human interaction. In this paper, the system represents a maze simulation environment and an agent working inside it to solve it using its brain.

Index Terms— Artificial Intelligence, Reinforcement Learning, Deep Learning, Neural Network, Maze solving, Q- learning, Deep Q-Reinforcement Learning

1 INTRODUCTION

Artificial Intelligence is growing steadily in all fields, we see around us. It has started to represent a necessity for various technological processes such as the automation of operations in various factories, certain robots that can make decisions without human help, cars that can drive alone and so on. AI machines that learn like children provide deep insight into how the mind and body acts together to get the work done. The importance of such a technology arise since pre-programmed robot cannot adapt to complicated dynamic problem that they may face in the real-world. Hence it is extremely important to introduce the "learning from experience" strategy which will be the focus of this research. This strategy is implemented using a combination of Artificial Neural Network and Deep learning.

2 RELATED WORK

A. The relevant approaches to solve the dynamic decision-making problem of the robot within a maze or any environment with obstacles had been a thoroughly discussed topics in robotics and machine learning. Many researchers have taken the duty of solving this problem in the past, For example, Norbert-Brendan, K., & Cristian Marius, T(2019)[1] ,had put forward an approach to solve a single maze with a use of different maze solving algorithms like left hand rule and dead-end filling algorithms, along with DRV8835 library. This approach failed to implement the decision-making capability of the robot hence fails when changing the nature of the environment.

B. Reinforcement learning in robotics has been a challenging subject for the past few years. The ability to equip a robot with a powerful enough tool to allow an autonomous discovery of an optimal behavior through trial-and-error interactions with its environment was the cause of many deep research projects.Tiago Ribeiro and Fernando Goncalves [3] proposed a work with two different Q- learning approaches and an extensive hyperparameter study. An approach to the autonomous mobile robot obstacle prevention problem using reinforcement learning,more precisely , Q-learning was implemented. The algorithm was developed for a simplistically simulated Bot'n Roll ONE.The simulated robot communicates with the control script with ROS. The proposed obstacle avoidance method worked with simple obstacle avoidance.The simple reinforcement learning algorithm would collapse when dealing with complex mazes.

C. Teng Zhao and Ying Wang [2] presented an autonomous navigation system based on neural networks using mobile robots. The main contribution of this work was to develop a navigation system with the ability to learn to adapt to unknown environments. A neural network based autonomous robot navigation system using mobile robots was developed and validated. The model was trained using specially designed training samples to deal with various situations seen in the real world. The trained neural network was then tested with simulations and experiments that validate its feasibility. The model was trained with certain training samples which might not be sufficient for the robot to learn to perform in the real-world environment.

In this paper, a modified learning algorithm called "Deep Q Reinforcement learning" is used to train the robot which made use of the Q values for every (state, action) pair ob-

• *Harikrishnan Bejishkumar has completed Graduate diploma program in Mechatronics at Wellington Institute of Technology,Wellington, New Zealand. E-mail : haribejishkumar95@gmail.com*

International Journal of Scientific & Engineering Research, Volume 11, Issue 9, September-2020
ISSN 2229-5518

tained from the neural network to select and perform the appropriate next action for a certain state that maximizes the reward. This also gives the robot (agent) to explore and find out different paths in the maze, like the humans find different ways to solve a problem in the real-world. The trade-off between exploration and exploitation is one of the important decisions made by the agent in the environment, this is also done during the learning period in the maze.

3 BACKGROUND STUDY

Most of the people, including engineers are not familiar with the concept of artificial intelligence.Despite its widespresd familiarity, AI is a technology that has influenced every walks of life.AI has givenbirth to a new classification of age called ‘Augmented age’,which has the power to take the place of humans in the far future.This section will walk you through AI and all the key concepts used in the research for developing the project.

3.1 Artificial Intelligence

Artificial Intelligence or AI is the science behind programming, robots to think, act and react like humans do. This helps robots to mimic human intelligence in a way that enables machines to make traits associated with human mind such as problem solving and decision making. The goal of AI includes learning, reasoning, and perception.

3.2 Machine Learning

Machine learning or ML is a subset of artificial intelligence, where machines are taught and guided through most suitable way to achieve tasks. They learn from what is taught and do the task perfectly. This is the concept that accelerated the widespread of AI in many fields. However, in ML, they need guidance if they make an inaccurate decision.

3.3 Deep Learning

Deep learning is a subset of Machine Learning,where the machines have the capability to decide whether the decisionsare accurate.It has a property that enables the machines to approximate or predict the function and learn from its experience.

3.4 Artificial Narrow Intelligence

Artificial Narrow intelligence or ANI is a part of AI where the machines are fed in, a data set as an input to train them. Their intelligence is bounded into a narrow space within the inputs provided to them. Supervised and Unsupervised learning can be considered under ANI.

3.5 Artificial General Intelligence

Artificial General Intelligence is part of AI where the machines have no clue of data, they can grab the inputs ad learn from the surroundings to achieve certain task. The algorithm can be reused in many applications with certain alterations to improve performance. Reinforcement learning can be considered as a form of AGI.

3.6 Reinforcement Learning

Reinforcement learning is a type in machine learning that could come under Artificial General Intelligence (AGI). This type of learning does not require any inputs\outputs or any intermediate actions to achieve a task. Instead, it finds a trade-off between exploration and exploitation. This make use of an idea of training an agent in an unknown environment, which has analogy to making a child learns from basic. [6].

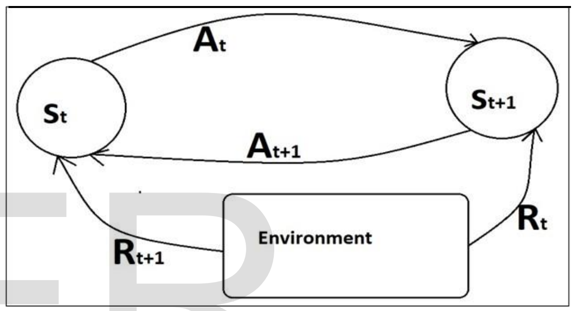


Fig. 1. Reinforcement Learning

In reinforcement learning, agent can take actions (A_t or A_{t+1}) in an environment. A reward (Rt or Rt+1) for all actions taken, whether good or bad. Reinforcement agent could perform the best actions possible by experience. It always selects a policy from the experience gained, to maximize the cumulative reward. This provides a broader reach to programming the brain of a robot for all stochastic dynamic environment. The RL agent uses the idea of Markov decision process (MDP), where it considers all the states in the environment have a Markov property.

“*A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it”* **("Markov property", 2020)** [4]

The equation representing a Markov process is called a Value function and are as given below:

$$V(s) = max_a(R(s,a) + (\gamma \times \sum_{S'}(P(s, a, s') \times max(V(s', a')))))$$

S – current state, a- current action, γ – Discount factor, s’ – next state, a’ – next action, and R- reward. [4]

3.7 Q-Learning

Q- Learning is a model free, off-policy reinforcement learning method that investigates the environment based the quality of actions take in each state. It is an off-policy learning because it even uses actions outside of the current policy to explore the environment. 'Q' stands for quality of actions. It considered stochastic immediate rewards for each state, action pair (s,a) and thereby selects the action which is more precise to accomplish a certain task. Q value of a state action pair is the sum of reward it gets for the action plus the discount factor (γ) multiplied by the sum of probabilities to go to the next state multiplied by the maximum q value for the next state. i.e.

$$Q(s,a) = R(s,a) + (\gamma \times \sum_{S'}(\mathrm{P}(s, a, s') \times max(\mathrm{V}(s', a'))))$$

s- Current state, a- Current action, γ- Discount factor, s'- Next state, a'- Next action, R- reward.

Q- Learning works with discreate actions and state spaces, because all the Q-values of (s,a) are recorded in a table which serves as function approximator for the algorithm. After learning process, the agent makes use of this table to select the perfect action at a certain state. Each time it calculates a new value, the Temporal Difference (TD loss) is calculated. Temporal difference or loss function is the difference in loss of the old q value to a target q value. This function is used to update the Q table thereby minimizing the TD loss, which is given below:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[\boxed{R_{t+1} + \gamma \max_a Q(S_{t+1}, a)} - Q(S_t, A_t) \right]$$

Where, Q (St, At) is the predicted q value or old q – value and α – Learning rate defines the rate at which the agent learns (i.e. updating weights) [5].

The action selection policy is the function that creates a balance between exploration and exploitation.This can be done using many functions like like ε-greedy, SoftMax and so on.

3.8 Deep Q- Reinforcement Learning

Deep-Q Reinforcement learning or DQRL gives an algorithm that helps the agent learn from experience. It is a model-free off-policy algorithm that allows a continuous state space and discrete action space. The difference of DQRL with Q-learning is that, it incorporates a neural network as a function approximator. The mostly used neural networks are the following:

1) Convolutional neural network (CNN): Convolutional neural network is a type of deep neural network with convolutional layers that are used for image classification, medical image analysis, Natural language processing and so on.

2) Artificial neural network (ANN): Artificial neural network is a type of neural network with fully connected linear layers that are used in deep learning algorithms. It can be used to create the algorithm efficient as this calculates the future values of output and provides different methods to learn and make it capable to do better actions from its experience. The components and process used in ANN's are:

- Activation function: This function is used to activate the neurons of hidden layers to provide the best decision possible. ReLu function and sigmoid function are the examples of activation function.

- ANN layers: ANN consists of Input, Hidden and output layers. Input layer takes in states as a batch and provides the output to hidden layer with a weighted sum and activation function. Hidden layer neurons are responsible for all the logical calculations behind neural networks that provide the output. Output layer provide the q values of actions.

- Experience reply: This is the method of storing the experience it had for learning purpose. This is done by creating a reply memory, where is stores the sample batch of experience. After storing a certain number of samples, learning starts. It randomly chooses some number of samples from it and use it to select actions for the agent. This is a reusable memory.

- SoftMax function: It is a function for action selection polices while functioning with neural networks. It is nothing but give the probability distribution of q values in the range [0,1]. The q value with the largest probability is the q value for best action and hence chosen. This is used in the project make the agent explore a bit more and hence select the best action with greater probability, rather than taking random action.

- Back propagation: It is a method by which the ANN calculates the gradient at each point in the loss function. This could help in updating weights to minimize loss, which is called stochastic gradient descent.

The Deep-Q learning algorithm using ANN is as shown below:

Whenever an agent is in a certain state there are some tuple which is known to the agent i.e. (St, At, Rt, St+1), which is current state, current actions, reward, and next state. The DQRL algorithm based artificial neural network takes states as input in a batch and calculates the output, which is the q values for the possible actions.

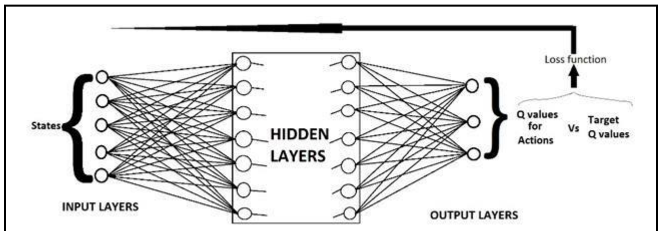


Fig. 2.DQRL Algorithm

This is done using the logical calculations and connections inside the Artificial neural network by the layers. Every time it reaches a new state, the experience is stored in the memory for a process called Experience reply. From the St+1, new target q values are calculated using the q learning algorithm equation. This is compared with the actual q values for the agent calculated by neural network. This comparison formulates a function called loss function. Then back propagation is done with the current loss function stochastically lower the gradient to bring down to the local minima of the loss function, which updates the weights to find the best action. After having certain samples in the reply memory, learning process starts and thereby slowly improve by gaining more experiences. Each time it gains experience, it takes the actions by minimizing the loss function and creating a larger cumulative reward for actions.

3.9 Kivy Simulation

Kivy is a python integrated open source simulation platform that used for rapid development of applications with user interfaces. This is multi touch application software that can be used in many Operating systems including Mac, Android, Linux, and Windows. It helps to create Apps for android using python code and is easy to learn.[7] The libraries used for KIVY are:

- Kivy.properties: That allows to store the properties of objects in the canvas. Examples are Object properties and Numeric Properties.
- Kivy. Clock: That allows to count the clock time and use functions related to it.
- Kivy. Vector: To calculate and use the direction of vector of an object at a certain point. It can also be used to store a vector in a variable.
- Kivy.uix.widget: This is used to inherit the property of the widget class.
- Kivy.config: This is to use a function that avoids multitouch.

4 PROPOSED SYSTEM

The design of the system had to go through different stages from information gathering to a finished product. The only idea which is known before going into the actual problem was 1) The task, that should be achieved, 2) The language for coding the program. The programming language which was so sure about was Python because it is a high-level language which provide simplicity in codes and many inbuilt framework libraries for AI. The python integrated framework was chosen to be Pytorch, as Pytorch was the best option as a starter, because it provides the simplicity and inbuilt libraries to code a neural network architecture and its associative functions. It had good readability, that made it possible to understand the process for a reader with ease and it also makes debugging easy. This was highly preferred for academic research and hence it could be used in all OS platforms including Windows. It also has high flexibility and more speed on working with neural networks, when compared with other platforms.

For the AI code which was developed required huge amount of research about lot of technical terms from scratch.

A. AI Code

The design of the AI code was developed such that it follows the DQRL algorithm precisely. It acts as the brain of the system. The neural network architecture used consists of
1 input layer, 3 hidden layers and 1 output layer. Input layers consists of 5 input neurons in which first 3 inputs are from the sensors of the agent and 2 are the orientations of the agent in the environment. The output layer consists of 3 output neurons, which are 3 possible actions for the agent (left, right and forward). The AI code runs each time, it takes an action and goes into a new state. While obtaining the results, the code was made some alterations to meet the expectations about the performance of the robot. The architecture of the neural network was changed to improve the decision-making capability of the robot and to accurately predict the best action. The temperature parameter was also changed to make sure the robot is more accurate about certain actions. This made the robot exploit the information it had in new paths. All these solutions where tried out during the learning process and thereby had to select the suitable one which produced more results within the training it was given.

B Simulation Environment

The idea simulation environment was emerged later after the AI code was made. This was written using python with the aid of imported kivy libraries. The platform runs using the kivy.app library. The environment was designed in such a way that it gives an option to draw a maze using Input management in kivy. Initially, values of all the pixels in the app platform is initialized to zero using numpy arrays from numpy library. The maze can be drawn using the cursor as kivy creates a touch simulation app. The properties of the object such as velocity, angle, sensors, signals from the sensors and so on is obtained using the kivy.properties library. When start is pressed, there are series of functions that is happening behind the scene. They are:

- The origin and initial vector of the car is fixed.
- The goal is fixed, which is the nearest vertex/corner to the last point in the maze.
- The walls of the maze are detected and is given the value of 1 for each pixel.
- The agent starts to move.

The goal is recorded only once during one exploration unless changed the orientation of the maze. Each actions of the agent are selected by the DQRL algorithm, which changes the state. The state is determined by the properties of the agent which is a batch of 5 variables that comprises of 3 signals from the sensors and two orientations of the agent. It is by using these values and other properties like velocity, angle and so on, the properties of the agent are updated for the next action. The updating takes place with a given clock interval of 1/60, which is using the kivy.clock library. The simulation environment also provides the rewarding system for the agent. This rewarding system is made after a lot of trial and error repetitions to find the optimal rewards for different situations that could give the best results. Each time it takes an action every property is updated, and reward is obtained, which then fed into the brain. A button for saving and loading the current brain is provided to save the path file and re- use it in another maze. The clear button clears all the drawings in the canvas. These are provided using kivy.uix.button library. Other property that is made to use from kivy is the kivy-language. It is made to create canvas objects and its position, rotation and to reflect the updates in the canvas. This could also be written using python code, but the code looks simpler and arranged by using kivy-language. This runs in parallel to the simulation code, as this is fed back to the classes in the simulation code that inherit from widget class library in kivy.

5 RESULTS & DISCUSSION

The result was obtained as a plot of x-coordinate of the agent's position and loss function at x and y coordinates. The graph was plotted using the values stored in the .csv file. The plot is also shown with the values of Q for all 3 possible actions, in a tensor.

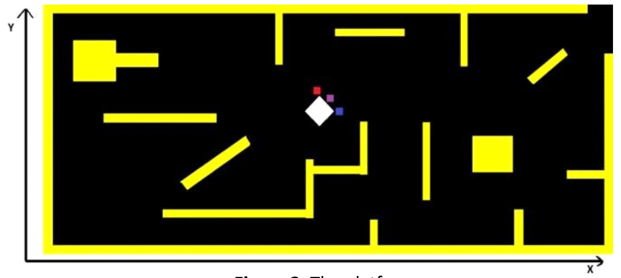
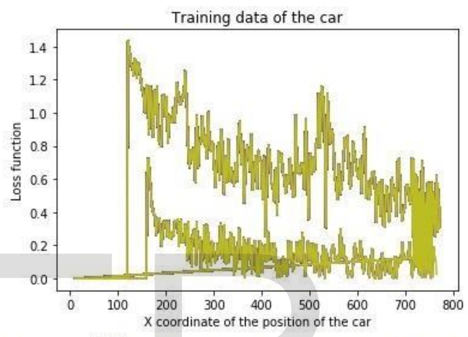


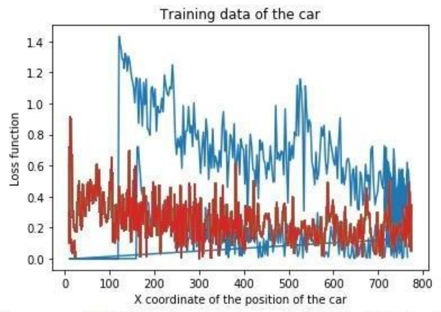
Figure 3: The platform

Figure 3 shows the platform or the maze the robot was trained for. The results in the traines maze showed better perfomence and the brain loaded with this experience can be tested in the other mazes as well.



```
tensor([[47847.0430, 17359.2441, 38727.9531]],
grad_fn=<AddBackward0>)
```

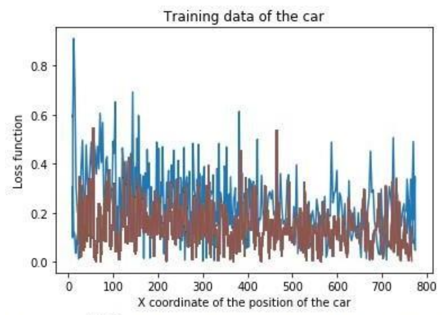
Fig. 4.Training 1



```
tensor([[250360.9062, 146017.6094, 174997.4375]],
grad fn=<AddBackward0>)
```

Fig.5. Training 2

International Journal of Scientific & Engineering Research, Volume 11, Issue 9, September-2020
ISSN 2229-5518



```
tensor([[537319.3750, 320988.7500, 367144.7188]],
grad_fn=<AddBackward0>)
```

Fig. 6. Training 3

The Figure 4, Figure 5 and Figure 6 shows the last 3 consecutive trainings of the robot in the above maze (Figure 3). The blue curve in the plot shows the plot for the previous epoch. This enables the comparison of the consecutive performances.

The initial training showed in Figure 7 has a huge loss function.

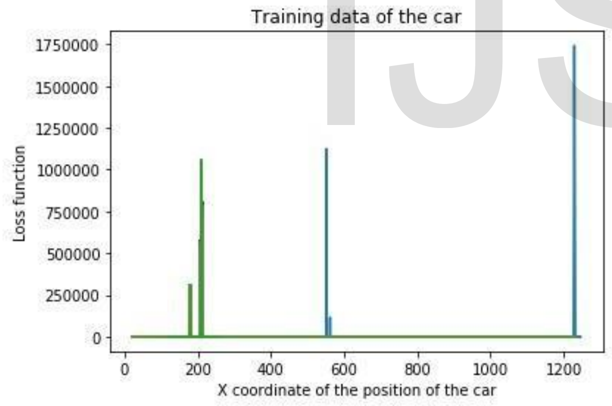


Fig. 7. Huge loss function for initial training

The discussions based on the observations from the results are given below:

The agent could explore and learn from a given maze (Figure 3). The agent creates a policy within the information obtained from the maze. The initial trainings were time consuming as it tried to explore different options to move. The graph was plotted for different properties like velocity, orientation, position of the agent, but the graph for loss function showed the actual performance of the agent. The data about the exploration, i.e. the loss function and x- coordinate of the agent for each training is saved on .csv files. The agent's performance was good enough after several trainings, as it was sure about the actions to take in most of states. The greater number of trainings given to the agent made it accomplish the tasks more efficiently. The graph for the initial trainings had larger values of loss function (Figure7) because the agent had not had any experience about the paths sure about the actions to take. As it gained experience, the values for loss function considerably decreased to a lower value and some of the last plots showed lower values for the peaks. The training was ceased when it obtained a lower value for the loss function. Figure 4, Figure 5 and Figure 6 has proved that the policy that the agent selected, solved the maze within the experience it gained from the trainings, tried to minimize the loss function there by increasing the total cumulative reward for those 3 actions possible (move left left, move right, move forward).

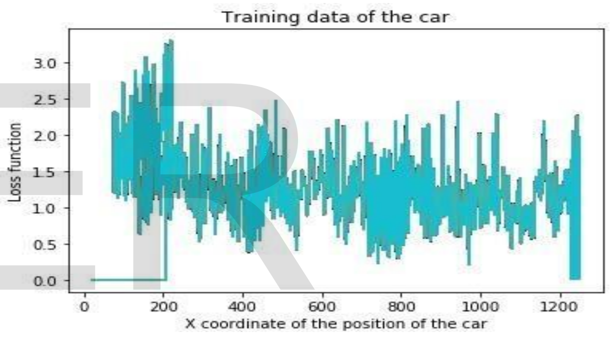


Fig. 8.Trained with different maze using loaded brain

The results obtained for the trained maze with a different starting point (Figure 9) had almost the same range of values for the loss function as it had for the original maze trainings. This was because, agent was sure about the states and actions that could possibly be done in the maze, even if the starting point were altered. The option for saving and loading the brain made it to be reused in different mazes. But the accuracy was not good for initial trainings, however, the accuracy increased within a smaller number of trainings.

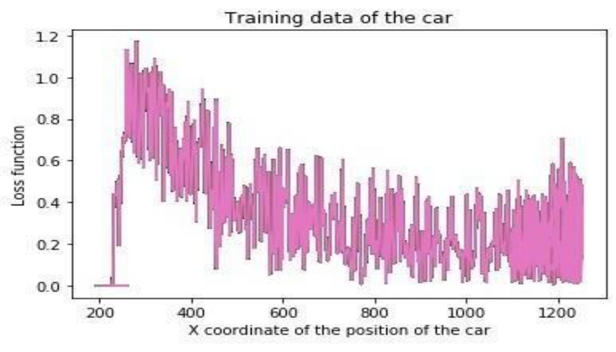


Fig. 9.Trained Maze with a different starting position

When comparing the results for the initial trainings in a different maze with loaded brain (Figure 8) with that of the previously trained maze, shows that different maze had much better accuracy, as it used loaded brain. But the loaded brain produced much better results in the trained maze with a different starting point (Figure 9), when compared it with the results of loaded brain in a different maze (Figure 8).

6 CONCLUSION

The models that uses Q-learning or neural network provides a good result within the training samples they had trained for. In this paper, use of Deep-Q reinforcement learning that used a combination of Q-learning and neural network, to find an optimal solution with the experience it gets within the training. This technique could even outperform most of conventional maze solving techniques with a better architecture or use of better functions in the learning process, that could be considered in the future enhancement of the work. This technique could also be used in different fields from home-serving humanoid robots to self-driving cars.

The development of hardware model can be considered among one of the future improvements. The performance can also be improved if the agent were given a greater number of trainings and hence could gain much more experience.

7 REFERENCES

[1] Norbert-Brendan, K., & Cristian Marius, T. (2019). *Autonomous Line Maze Solver Using Artificial Intelligence. 2019 15th International Conference on Engineering of Modern Electric Systems (EMES).* doi:10.1109/emes.2019.8795101

[2] T. Zhao and Y. Wang, A neural network based autonomous navigation system using mobile robots, 2012 12th International Conference on Control Automation Robotics & Vision (ICARCV), Guangzhou, 2012, pp. 1101-1106, doi:10.1109/ICARCV.2012.6485311.

[3] T. Ribeiro, F. Gonçalves, I. Garcia, G. Lopes and A. F. Ribeiro, *Q-Learning for Autonomous Mobile Robot Obstacle Avoidance*, *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Porto, Portugal, 2019, pp. 1-7, doi: 10.1109/ICARSC.2019.8733621.

[4] Markov property. (2020). Retrieved 25 June 2020, from https://en.wikipedia.org/wiki/Markov_property

[5] Python, A. (2020). Deep Q-Learning | An Introduction To Deep Reinforcement Learning. Retrieved 25 June 2020, from https://www.analyticsvidhya.com/blog/2019/04/introduction-deep- q- learning-python/ M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[6] Reinforcement learning. (2020). Retrieved 23 June 2020, from https://en.wikipedia.org/wiki/Reinforcement_learning

[7] Kivy: Cross-platform Python Framework for NUI. (2020). Retrieved 23 June 2020, from https://kivy.org/#home