CS6611 - Creative and Innovative Project

Project Guide : Dr.S.Muthurajkumar ***Team number: 09***

Path Navigation using Deep Q Learning

Team Members

Rahul B (2019503545)

Sanjay kumar L S (2019503042)

Gokkul E(2019503517)

Introduction :

There is an increased need for automation in this ever advancing tech world.Automation reduces time, money, labour, while also reducing manual errors, giving you more time to concentrate on other work. Manual tasks are hectic and boring, sometimes they are dangerous considering the work. Navigation plays a vital role in many industries, some of them are vehicles, drones, transportation, and there are certain cases where navigation might be dangerous like deep forest, underground, underwater, firefighting environments. Visually challenged people have a difficult time navigating unfamiliar places. So there is a necessity for an automatic navigation system to overcome these situations.Hence in our project we are going to implement automatic path navigation simulation using deep Q-learning .

Objective :

- To use kivy software and deep Q learning for path navigation of an object.
- To make the object automatically navigate from source to designated destination by avoiding all the obstacles using 3 sensors ,one to right, one to the left and in front

Literature Survey :

S.No	***Title of the paper and Published year***	***Proposed Work and Results***	***Limitations***
1	**Multi-Robot Path Planning Method Using Reinforcement Learning** **Hyansu Bae, Gidong Kim, Jonguk Kim, Dianwei Qian and Sukgyu Lee.**	They dealt with information and strategy around reinforcement learning for multi-robot navigation algorithms where each robot can be considered as a dynamic obstacle or cooperative robot depending on the situation. That is, each robot in the system can perform independent actions and simultaneously collaborate with each other depending on the given mission. After the selected action, the relationship with the target is evaluated, and rewards or penalty is given to each	The environment where the generated path is simple or without obstacles, an unnecessary movement occurs.and it did not take into account the dynamics of robots and obstacles.

		robot to learn.	
2	**Robot Training and Navigation through the Deep Q-Learning Algorithm** **Madson Rodrigues Lemos; Anne Vitoria Rodrigues de Souza; Renato Souza de Lira; Carlos Alberto Oliveira de Freitas;**	They aimed to present the results of an assessment of adherence to the Deep Q-learning algorithm, applied to a vehicular navigation robot. The robot's job was to transport parts through an environment, for this purpose, a decision system was built based on the Deep Q-learning algorithm, with the aid of an artificial neural network that received data from the sensors as input and allowed autonomous navigation in an environment. For the experiments, the mobile robot-maintained communication via the network with other robotic components present in the environment through the MQTT protocol.	The research was limited to the use of educational robots. The algorithm does not perform more complex tests with dynamic environments.

3	**Towards Real-Time Path Planning through Deep Reinforcement Learning for a UAV in Dynamic Environments** **Chao Yan, Xiaojia Xiang & Chang Wang**	They have proposed a Deep Reinforcement Learning (DRL) approach for UAV path planning based on the global situation information. They have chosen the STAGE Scenario software to provide the simulation environment where a situation assessment model is developed with consideration of the UAV survival probability under enemy radar detection and missile attack.	Although their research is highly efficient in simulation , it is hard to develop this in real life environment and it is not feasible
4	**Path planning of mobile robot in unknown dynamic continuous environment using reward-modified deep Q-network** **Runnan Huang Chengxuan Qin Jian Ling Li Xuejing Lan**	Their research aimed at the path planning of mobile robots in UDE, a continuous dynamic simulation environment is built in this article. Based on DQN, a reward function with reward weight is designed, and the influence of reward weight has been analysed experimentally. Moreover, the abnormal rewards	Their work focused on the performance of DQN on the policy of the robot's moving direction, hence the velocity and the acceleration of the robot are not considered Moreover, this article does not consider the specific dimension.

		caused by the relative motion between obstacles and robot have been analysed and solved by adding a reward modifier to DQN. The comparative experiment among RMDQN, RMDDQN, dueling RMDQN, and dueling RMDDQN was done, and turns out that the result of RMDDQN is the best.	
5	**Autonomous Navigation for Omnidirectional Robot Based on Deep Reinforcement Learning** **Van Nguyen Thi Thanh , Tien Ngo Manh, Cuong Nguyen Manh, Dung Pham Tien, Manh Tran Van , Duyen Ha Thi Kim and Duy Nguyen Duc**	They aimed to illustrate how the Omni robot performs navigation using model-free deep Q learning to navigate in unpredicted environments. It will also explain how to obtain the policy when such a model is unknown in advance by using a virtual environment to conduct in simulation.	Their system attempted to find the best route by moving around or near the obstacle several times which is not practical in real life scenarios.

Architecture Diagram :

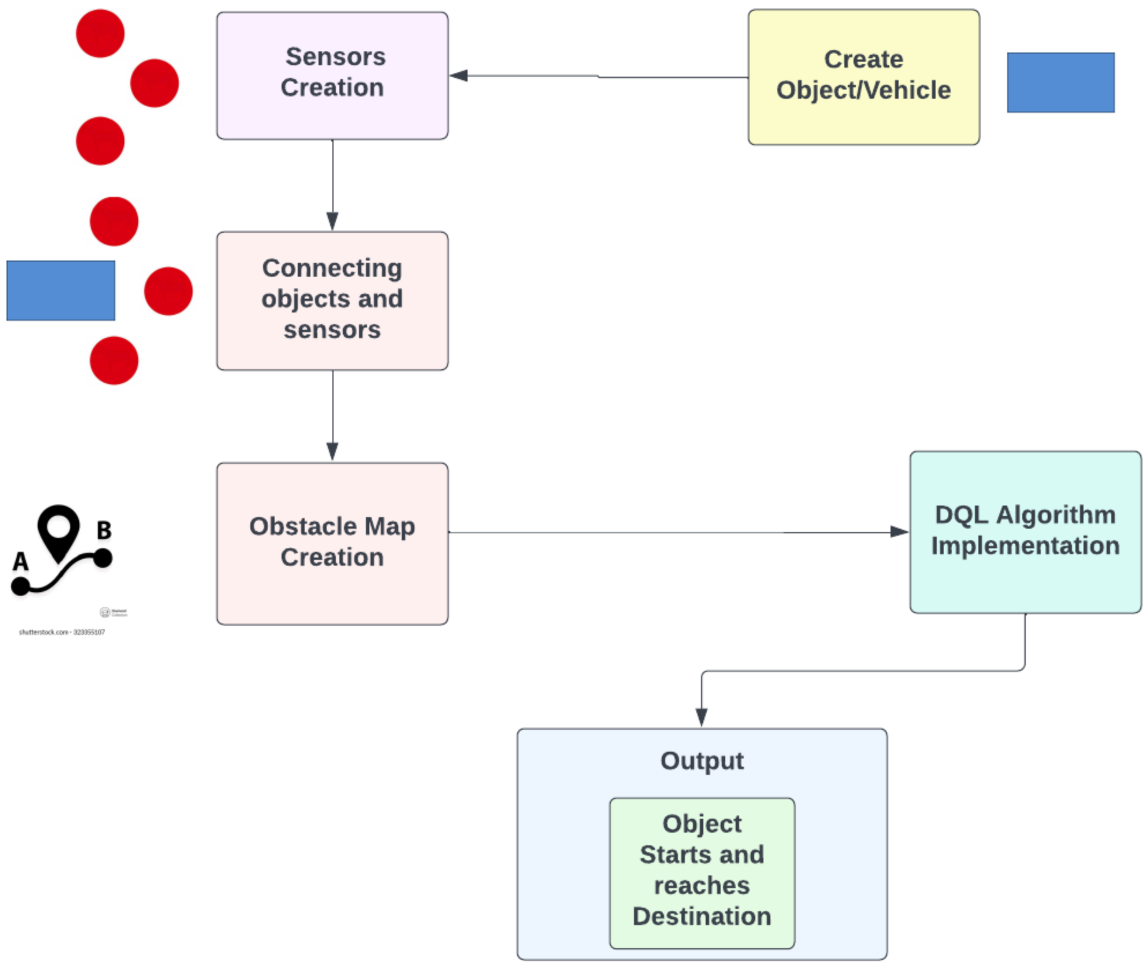

Figure 1 - Architecture Diagram

The project's Architecture diagram in figure 1 has 6 modules , Create Object/Vehicle - This module creates the object required, that is vehicle using Kivy. Sensors Creation - This module creates the 3 sensors required using Kivy. Connecting objects and sensors - Connecting the 4 objects into a single entity.Obstacle Map Creation - In this module, we create the environment in which the object runs. DQL Algorithm Implementation - This module deals with the DQL Algorithm which is used by the object to learn avoiding obstacles. The last module that is the output module Object starts and reaches destination - This module is used to build and run the project.

Proposed work :

The goal of this simulation is to build an environment with a complex path, an object with 3 sensors which uses Deep Q-Learning to train the object and to assign a destination for the object to reach.

Anaconda is what we'll use to install PyTorch and Kivy. It is a free and open source distribution of Python which offers an easy way to install packages.

We will build this 2D map inside a Kivy webapp. Kivy is a free and open source Python framework with a user interface inside which you can build your games or apps. It will be the container of the whole environment.

PyTorch is the AI framework used to build our Deep Q-Network. PyTorch is great to work with and powerful. It has dynamic graphs which allow fast computations of the gradient of complex functions, needed to train the model.

This can be divided into 3 integral parts to simplify the process ,
1.To create the object and sensors.
2.To build the map and to assign inputs and outputs.
3.Implementing Deep Q-Learning to train the object.

1. To create the object and sensors.

We use Kivy WebApp to create 4 Kivy objects, a rectangle shape representing the object and three sensors to detect any obstacle and to navigate to the destination.

2. To build the map and to assign inputs and outputs.

We build an environment containing a map and assign input states, output actions.

we train our object to go from the upper left corner of the map, to the bottom right corner, through any path we create between these two spots.

3. Implementing Deep Q-Learning to train the object.

Using Deep Q-Learning we build and train our object to navigate its way avoiding any obstacles to its destination.

Algorithm :

1.Importing the libraries and the Kivy packages.

2. Initialising variables to keep the last point in memory when we draw the sand on the map , the total number of points in the last drawing , the length of the last drawing.

3. Create the brain of our AI, the list of actions and the reward variable :

I. 4 inputs, 3 actions, gamma = 0.9.
II. action = 0 => no rotation, action = 1 => rotate 20 degrees, action = 2 => rotate -20 degrees.
III. The reward received after reaching a new state.

4. Initialising the map :

I. Sand is an array that has as many cells as our graphic interface has pixels. Each cell has a one if there is sand, 0 otherwise.
II. Building x-coordinate and y-coordinate of the goal.
III. Initialising the sand array with only zeros.
IV. The goal to reach is at the upper left of the map. (the x-coordinate and y-coordinate)
V. Initialising the last distance from the object to the goal.

5.Creating the object class :

I. Initialising the angle of the object.
II. Initialising the last rotation of the object.
III. Initialising the x-coordinate and y-coordinate of the velocity vector and the velocity vector.
IV. Initialising the x-coordinate and y-coordinate of all 3 sensors and their respective sensor vector.

6. Updating the position of the object according to its last position and velocity :

I. Getting the rotation of the object.
II. Updating the angle and the position of sensors 1,2 and 3.
III. Updating the signal received by sensors 1,2 and 3. (density of sand around sensor 1,2,3)
IV. If any sensor is out of the map (the object is facing one edge of the map) that sensor detects full sand.
V. Update sensors 1,2 and 3.

7.Creating the game class :

I. Getting the object and the sensors 1,2 and 3 from our kivy file.
II. Starting the object when we launch the application.
III. The object will start at the centre of the map.
IV. The object will start to go horizontally to the right with a speed of 6.

8. Update function that updates everything that needs to be updated at each discrete time when reaching a new state (getting new signals from the sensors) :

I. Specifying the global variables.
II. Store width and height of the map (horizontal edge and vertical edge).
III. Storing the difference of x-coordinates and of y-coordinates between the goal and the object.
IV. Initialising the direction of the object with respect to the goal (if the object is heading perfectly towards the goal, then orientation = 0)
V. Initialising our input state vector, composed of the orientation plus the three signals received by the three sensors.
VI. Updating the weights of the neural network in our ai and playing a new action
VII. Converting the action played (0, 1 or 2) into the rotation angle (0°, 20° or -20°)

VIII. Moving the object according to this last rotation angle
IX. Getting the new distance between the object and the goal right after the object moved
X. Updating the positions of the 3 sensors 1,2 and 3 right after the object moved.

9. Assigning reward system :

I. If the object is on the sand, it is slowed down (speed = 1) and reward = -1.
II. Otherwise and it gets a bad reward of -0.2.
III. However if it is getting closer to the goal it still gets a slightly positive reward of 0.1.
IV. If the object is in any edge of the frame (top,right,bottom,left), it comes back 10 pixels away from the edge and it gets a bad reward of -1.
V. When the object reaches its goal, the goal becomes the bottom right corner of the map and vice versa (updating of the x and y coordinate of the goal).
VI. Updating the last distance from the object to the goal.

10. Painting for graphic interface :

I. Putting some sand when we do a left click.
II. Put some sand when we move the mouse while pressing left.

11. API and switches interface :

I. Building the app.
II. Creating the clear, save and load buttons.
III. Running the app.

Implementation :

Tools Used :

- Kivy
- PyTorch Framework

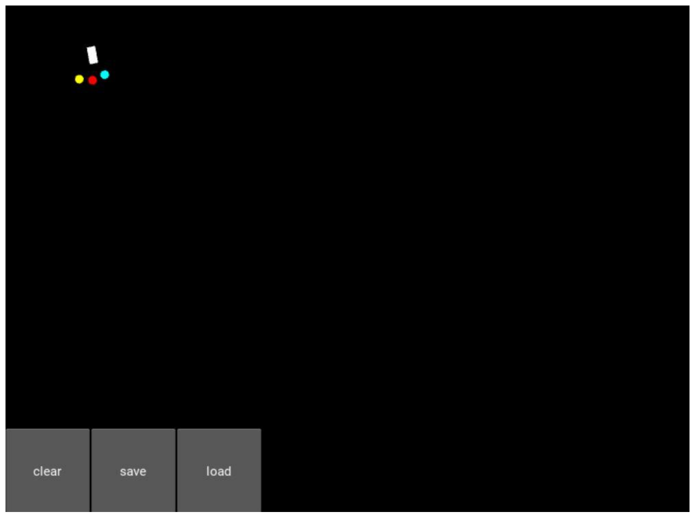

Figure 2 - Random Movement 1

Figure 3 - Random Movement 2

References :

1. **Huang, R, Qin, C, Li, JL, Lan, X. Path planning of mobile robot in unknown dynamic continuous environment using reward-modified deep Q-network. Optim Control Appl Meth. , pp. 1– 18, 2021, https://doi.org/10.1002/oca.2781.**
2. **Bae, Hyansu, Gidong Kim, Jonguk Kim, Dianwei Qian, and Sukgyu Lee. "Multi-Robot Path Planning Method Using Reinforcement Learning" Applied Sciences 9, pp no. 15: 3057 ,2019, https://doi.org/10.3390/app9153057.**
3. **M. R. Lemos, A. V. R. de Souza, R. S. de Lira, C. A. O. de Freitas, V. J. da Silva and V. F. de Lucena, "Robot Training and Navigation through the Deep Q-Learning Algorithm," 2021 IEEE International Conference on Consumer Electronics (ICCE), pp. 1-6, 2021, doi: 10.1109/ICCE50685.2021.9427675.**
4. **Yan, C., Xiang, X. & Wang, C. Towards Real-Time Path Planning through Deep Reinforcement Learning for a UAV in Dynamic Environments. J Intell Robot Syst 98, pp. 297–309 , 2020, https://doi.org/10.1007/s10846-019-01073-3.**
5. **W. Zaher, A. W. Youssef, L. A. Shihata, E. Azab and M. Mashaly, "Omnidirectional-Wheel Conveyor Path Planning and Sorting Using Reinforcement Learning Algorithms," in IEEE Access, vol. 10, pp. 27945-27959, 2022, doi: 10.1109/ACCESS.2022.3156924.**