**AUTONOMOUS NAVIGATION OF SELF LEARNING ROBOTS AUGMENTED WITH DEEP REINFORCEMENT LEARNING *AGILEFRIEND***

***Abstract***

The study aims at finding a solution for mobile robots to navigate in an environment which is spatially constrained. Reinforcement learning algorithms are seen as a powerful tool in autonomous navigation scenarios since the complexity of the algorithm is automatically learned by trial and error with the help of simple reward functions. This paper proposes a methodology to define simple reward functions and come up automatically with a complex and successful autonomous navigation policy. Reactions are learned based on the maximization of the cumulative reward obtained during the learning process. This paper explains the proposed methodology and discusses the setup of experiments and the results for the validation of the methodology in scenarios with static obstacles. Results of study are verified on CoppeliaSim simulation software.

1. **Introduction**

The study is intended for presenting an adaptive system for mobile robot . The adaptive system is based on self-learning in continually changing environment. Navigation of mobile robot with obstacle avoidance is a well researched area owing to its comprehensive applications [1] The adaptive system for mobile robots is an area of study which is under focus for last many decades. [2], [3], [4]Describes that adaptive systems in the realm of robotics are systems which can adjust their hardware and software to perform wide range of tasks while adapting to varying environments. Anuradha Annsawamy in her lectures states that the goal of adaptive control is real-time control of uncertain dynamic systems through adaptation and learning [5] [6] Experts [7] are of the opinion that an adaptive control is a type of feedback control system that can adjust its parameters and behavior according to the changes in the system or the environment. For example, if a robot needs to navigate in a terrain with different slopes and friction, the adaptive controller can modify its speed and torque based on the sensor data and the desired path. The advantage of adaptive control is that it can cope with uncertainties and disturbances, and improve the performance and robustness of the system. However, adaptive control is more complex and computationally demanding than other types of feedback control systems.Autonomous navigation requires a robot to be able to precept the surrounding environments through processing or fusing the data, collected from sensors, and highly performed robot perception enables a robot to make a right decision and thus to have a right response to any anomaly situation in its surrounding environment. Autonomous path planning with obstacles avoidance in dynamic environments is a crucial issue in the navigation of a robot. [8] Machine learning plays a vital role in enabling robots to fulfill tasks in accordance with users needs. The reinforcement learning technique is a hybrid model that encompasses both supervised and unsupervised learning techniques [9]. It is now widely utilized in various applications, including health, logistics, industrial, military, and other areas, to improve human life. [10] Deep reinforcement learning (DRL) is the combination of reinforcement learning (RL) and deep learning. It has been able to solve a wide range of complex decision-making tasks that were previously out of reach for a machine, and famously contributed to the success of AlphaGo. Furthermore, it opens up numerous new applications in domains such as healthcare, robotics, smart grids and finance. Correct selection of an algorithm for study is of paramount. One important aspect to consider when choosing an algorithm is the state- and action space of the environment. The state- and action space can be either discrete or continuous [11]. For example, if the action space is discrete then the number of actions is countable while if it is continuous the number of actions is, in theory, infinite. An algorithm with a discrete action space can only be applied to a continuous environment if the action space of the environment is discretized. [12]

For the purpose mobile robot navigation and avoiding obstacles and moving towards goal this stuy applies Artificial Potential Field (APF) is a nature inspired technique. The basic idea of APF is to fill the robot environment with the artificial potential field in which obstacles are repelled using repulsive force and robot is attracted toward the goal using attractive force. The potential field depends on two forces, that is, attractive and repulsive force. The goal produces the attractive force towards robot and obstacles produce repulsive force, which is inversely proportional to the distance from robot to the obstacles and directing toward obstacles. In APF field, robot travels from high potential to low potential. [13]After going through different algorithms we have selected deep deterministic policy gradient (DDPG). it is a model free, off policy actor-critic algorithm. it uses deep function approximators that can learn policies in high-dimensional, continuous action spaces.

This study implements artificial potential field (APF)In the artificial potential field, the robot motion is controlled by the attractive force and the repulsive force, i.e., the attractive force is generated by the distance and direction to the target point, whereas the repulsive force is generated by the distance and direction to obstacles. [14]The farthest reach (i.e., the tip) of a manipulator or an entire mobile robot can be considered to move in a field of forces, creating a *potential field* (PF). The desired positions to be reached are treated as attractor poles while obstacles are treated as repulsive surfaces (repulsive poles) for the robot. Potential fields, also known as *vector fields* in mathematics, reflect a snapshot of the current orientation for a cluster of moving objects at a given moment. An electric potential field, this concept is related to the potential energy of a positive test charge at various locations within an electric field. PFs cause the robot to appear to “slide around” obstacles. PF is defined as the sum of the attracting and repulsing forces in the field. Attracting forces get smaller as the robot approaches its goal. Repelling forces get larger as the robot approaches obstacles. PF is often constructed with a fixed radius of effective forces. For a detailed, mathematical, and algorithmic coverage of potential field methods, the reader may consult Latombe’s *Robot Motion Planning* [15]A potential field is any physical field that obeys Laplace’s equation. Some common examples of potential fields include electrical, magnetic, and gravitational fields. A potential field algorithm uses the artificial potential field to regulate a robot around in a certain space. For our ease, we consider a space to be divided into a grid of cells with obstacles and a goal node. Two kinds of artificial potential fields are generated within the system: Attractive field and Repulsive fields.

[16] describes as in APF robot is “attracted” to the goal and “repelled” from the obstacles • The robot is assumed as a positively charged particle moving towards negatively charged goal i.e. attractive force . Obstacles have same charge as robot – repelling force States far away from goal have large potential energy, goal state has zero potential energy. The Path of robot is from state of high energy to low (zero) energy at the goal . The planning space is considered as an elevated surface, and the robot is a marble rolling “downhill” towards the goal.

[17]describes that the idea of a potential field is taken from nature. For instance a charged particle navigating a magnetic field, or a small ball rolling in a hill. The idea is that depending on the strength of the field, or the slope of the hill, the particle, or the ball can arrive to the source of the field, the magnet, or the valley in this example.In robotics, we can simulate the same effect, by creating an artificial potential field that will attract the robot to the goal. By designing adequate potential field, we can make the robot exhibit simple behaviors.For instance, lets assume that there is no obstacle in the environment, and that the robot should seek this goal. To do that in conventional planning, one should calculate the relative position of the robot to the goal, and then apply the suitable forces that will drive the robot to the goal.In potential field  approach, we simple create an attractive filed going inside the goal. The potential field is defined across the entire free space, and in each time step, we calculate the potential filed at the robot position, and then calculate the induced force by this field. The robot then should move according to this force.

The APF algorithm was introduced in APF in 1985. [18] This algorithm considers the robot as a point in potential fields and then combines stretching toward the target and repulsion of obstacles. The final path of the output is the intended path. This algorithm is useful given that the trajectory is obtained by quantitative calculations. [19] Describes as the potential field method is widely used for autonomous mobile robot path planning due to its elegant mathematical analysis and simplicity

The classical Artificial Potential Field (APF) method consists of assigning an attractive artificial potential field to the destination point that attracts the robot, and a repelling artificial potential field to the obstacles that repel the robot. Being under the influence of these two combined potentials, the robot moves to its destination while avoiding the obstacles on its way. The way this algorithm works is that although the potentials are only artificial, they can generate an artificial force field which, in turn, combined with the robot’s state and artificial dynamics can produce a virtual velocity and acceleration that are used as an instantaneous reference to control the robot’s pose. [20]

The methodology presented in this work proposes an adaptive system for a robot driving in a scenario with spatial restrictions (limited drivable space), interacting with static and dynamic obstacles, and fulfilling at all times safety, legal, and comfort requirements. The Reinforcement learning technique is applied to guide a mobile robot from a desired starting position to a desired target position in an environment with static and dynamic obstacles is examined with a robot simulator called CoppeliaSim. The remaining part of this study is organized as follows:

In Methods section the Deep Deterministic Gradient Policy method is discussed in detail. Simulation methodology of DDGP is also highlighted. In results and section results are discussed and their implications are analysed. These results are achieved \by applying methods in previous section. In discussion section, DDGP method is discussed in detail. Its application in instant study is highlighted. Based on previous section conclusions are inferred in conclusion section.

1. **Methods**

The methodology proposed is based on the deep deterministic policy gradient (DDPG) algorithm. The DDPG is a model-free algorithm. That is, it does not use any embedded model of the robot inside the agent logic. Instead, it learns the optimal policy directly from the environment. The DDPG method is intended for environments with continuous states and actions. This is possible by approximating the actor model (which computes the action *at*) with deep neural network *µ* of weights *θ* as a function of state *s* at time *t* (1).

*at* = *µ*(*st|θ*) = *µθ* (*st*) (1)

*Qϕ*(*st* , *at*) = *Qϕ*(*st* , *µθ* (*st*)) (2)

Actor–critic methods combine the benefits of policy-based and value-based methods. They benefit from the good convergence properties of policy-based methods and they also benefit from the sample efficiency of value-based methods, which tend to find an optimal solution faster. The deterministic policy gradient theorem updates the weight *θ* of the actor model *µθ* in the direction of the maximum value of the critic model *Qϕ*:

*∇θ J*(*µθ* ) = E*s∼D*h *∇aQϕ*(*s*, *a*) *a*=*µθ* (*s*)*∇θµθ* (*s*) i (3)

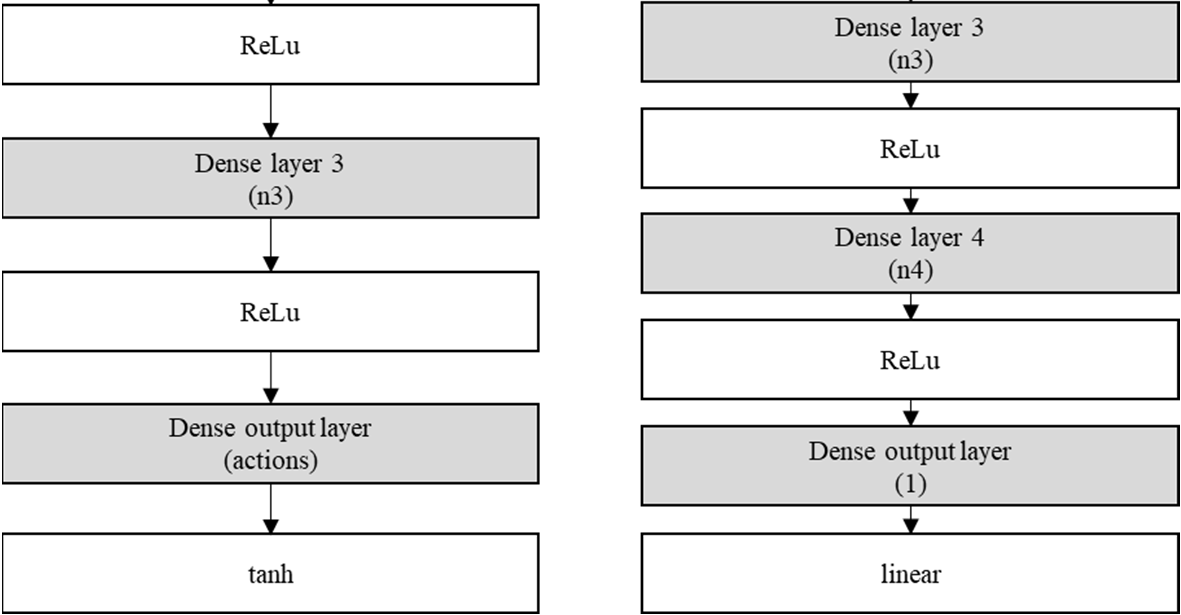
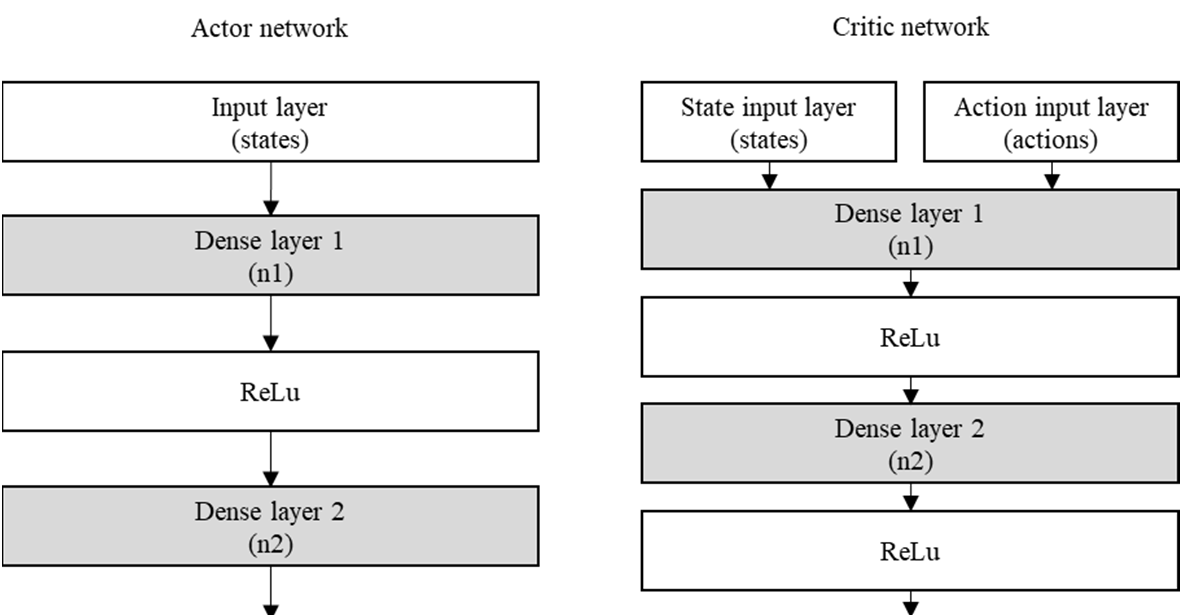
On the other hand, the Bellman equation is used to iteratively update the weight *ϕ* of the critic network *Qϕ* (4) and eventually learn the optimal critic and actor networks

*Qϕ*(*st* , *at*) = E*s∼D*h *r*(*st* , *at*) + *γQϕ* (*st*+1, *µθ* (*st*+1))I (4)

Additionally, this actor–critic method uses the replay buffer and target networks to bring more stability to the learning process. Random observations are selected from the replay buffer to avoid training the neural networks with consecutive and correlated observations. The weights *ϕ ′* and *θ ′* of the target networks *µ ′ θ ′*(*s*) and *Q′ ϕ′*(*s*, *a*) are updated after the weights of the actor and critic networks with a soft factor *τ* (5). This lag brings stability to the learning.

*θ ′* = *τθ* + (1 *− τ*)*θ ′*

*ϕ ′* = *τϕ* + (1 *− τ*)*ϕ ′ (5)*



**Figure 1. (**Actor and critic deep neural network architecture.)

.

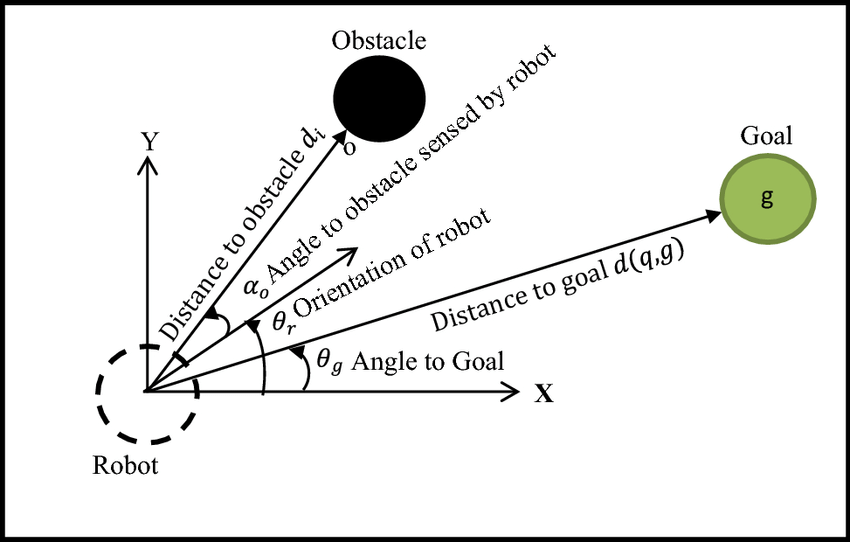


Figure-2 Distance of robot from obstacle and the intended goal at a particular time)

The APF method uses two types of forces, Attract force and repulsive force

f\_attr\_x = k\_attr \* (goal\_position[0] - x)

f\_attr\_y = k\_attr \* (goal\_position[1] - y)

f\_rep\_x, f\_rep\_y = 0.0, 0.0

the orientation of robot as well as obstacle avoidance is calculated based on these forces.

if future\_distance\_to\_obstacle < distance\_to\_obstacle:

                    repulsion\_magnitude = k\_rep \* (1.0 / future\_distance\_to\_obstacle - 1.0) / (future\_distance\_to\_obstacle \*\* 2)

                    f\_rep\_x += repulsion\_magnitude \* (future\_position[0] - obstacle\_position[0])

                    f\_rep\_y += repulsion\_magnitude \* (future\_position[1] - obstacle\_position[1])

                else:

                    repulsion\_magnitude = k\_rep \* (1.0 / distance\_to\_obstacle - 1.0) / (distance\_to\_obstacle \*\* 2)

                    f\_rep\_x += repulsion\_magnitude \* (robot\_position[0] - obstacle\_position[0])

                    f\_rep\_y += repulsion\_magnitude \* (robot\_position[1] - obstacle\_position[1])

        # Combine attractive and repulsive forces

        f\_total\_x = f\_attr\_x + f\_rep\_x

        f\_total\_y = f\_attr\_y + f\_rep\_y

**3. Results**

All the experiments were carried out in the simulator CoppeliaSim (Figure 2). Python scripts were used to connect and simulate with the simulation. The Application Programming Interface (API) *ZeroMQ*” is used for this purpose. For model development VS Code IDE is used. Two python files are created to simulated mobile robot navigation in continuous environment. The env.py is used to set Oppeliasim environment such as objects handling, reward function, start and stop of simulation. DDPG.py contains main logic for simulating DDPG algorithm. It includes ‘actor’ ‘critic’, ‘network’, ‘replaybuffer’ etc . The core DDPG training agent is also encapsulated in this file.



Figure 3 (CoppeilaSim Simulator)

The main libraries used to build the reinforcement learning algorithm were ‘numpy’, ‘torch’, ‘opencv’ and ‘TensorFlow’ . The Tensorflow and torch libraries provide the necessary functions to build actor and critic neural network models whereas opencv library is used to process sensor data. The simulation scene consists of PioneerP3DX Robot, a Vision sensor attached with the robot , 3 Obstacles Cuboid[0],Cuboid[1],Cuboid[2]. For target a Cone object is placed. This scene is depicted in in Figure-3.



Figure-4 (Mobile Robot Navigation Scene)

The env.py contains simulation control logic. It is responsible to start simulation by connection VS Code to API. It handles different simulation objects such as PionerP3DX Robot, right and left motors of robots, vision sensor, cuboids and cone. Important robot actions such as collision avoidance, rewards are also written in this python file.

The ddpg.py file contains important logics for DDPG agent. It contains Actor class, Critic class, Replaybuffer class, DDPG class. The train\_DDPG functions is responsible to execute training cycle of DDPG agent.

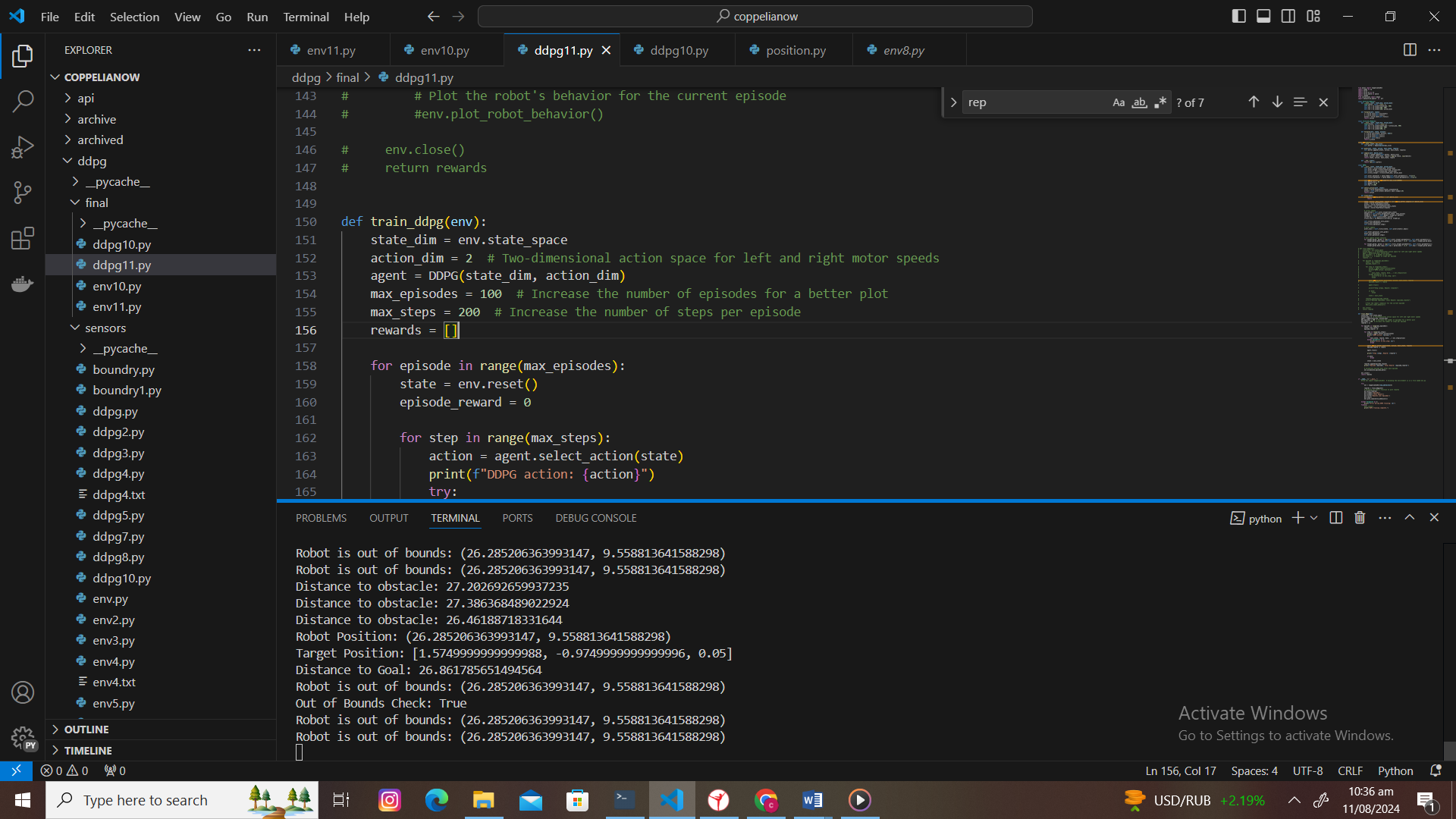


Figure-4(code snippet depicting number of episodes)

Here in this code snippets, it is visible that episodes are defined as 100 whereas each episode is to be iterated 50 times. This is sufficient enough to train the model and deduct unbiased inferences. With Each episode simulation scene is reset.

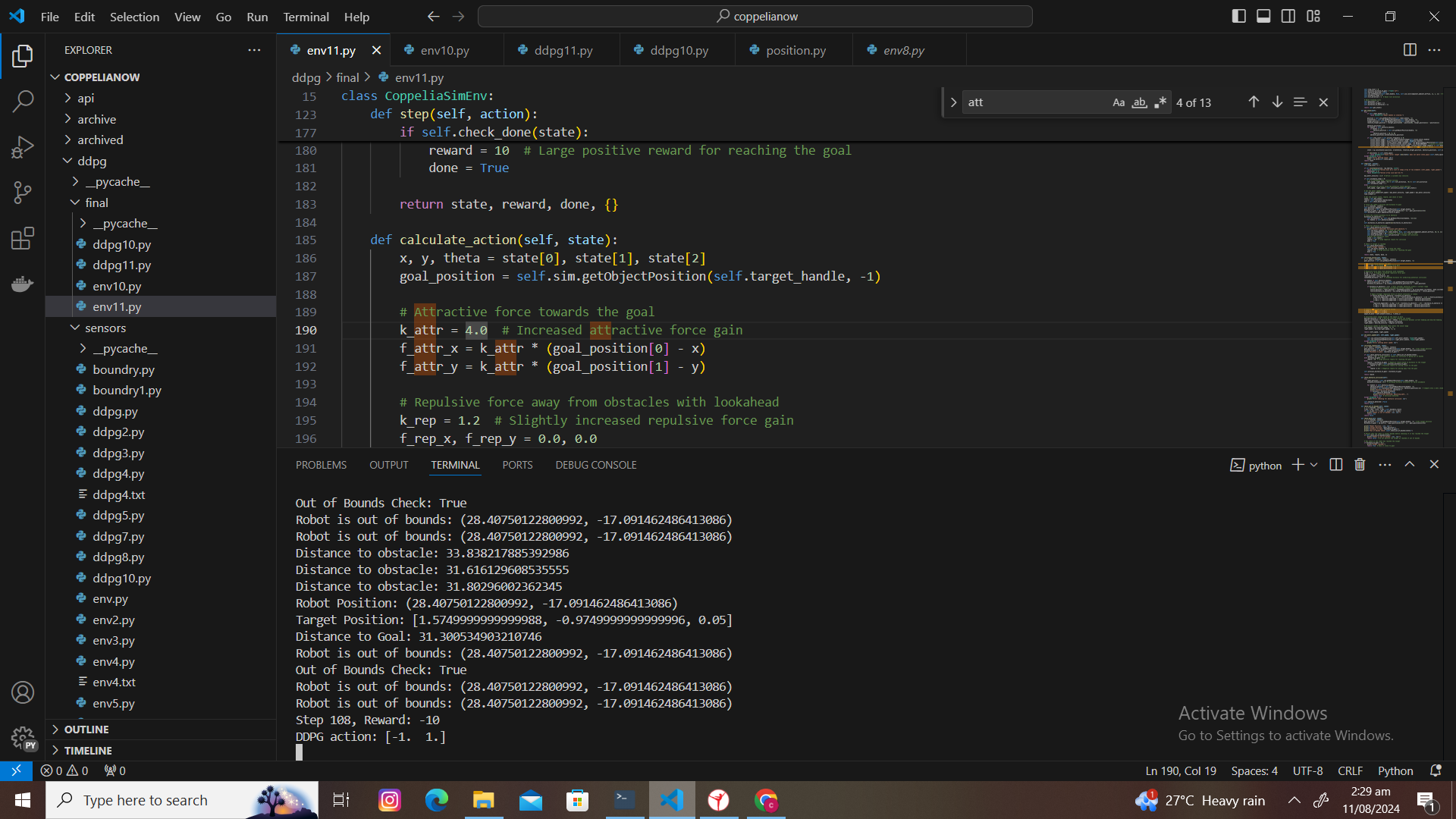


Figure-5(output of each episode with steps and rewards)

It is visible that actions are discrete which is essential to be an effective solution for dynamic environment. Moreover, rewards are also visible.. Negative rewards indicate a bad outcome/ policy, whereas positive rewards show satisfactory policy. At the start of each episode, the environment is reset to get the initial state. For each step in the episode, the agent selects an action based on the current state. The action is taken in the environment, resulting in a new state, a reward, and a done flag indicating if the episode is finished. The experience (state, action, next state, reward) is stored in the replay buffer. For training, The agent samples a batch of experiences from the replay buffer. The critic network is updated by minimizing the loss between the predicted Q-values and the target Q-values. The actor network is updated by maximizing the Q-value estimated by the critic for the actions it predicts. The target networks are softly updated to slowly track the learned networks. Obstacle avoidance is implemented. The robot's sensors provide data on obstacles, which are included in the state representation. The reward function penalizes collisions and encourages the robot to move towards a goal, incorporating obstacle avoidance implicitly. Over time, the agent learns a policy that balances exploration and exploitation, avoiding obstacles while navigating towards the goal.

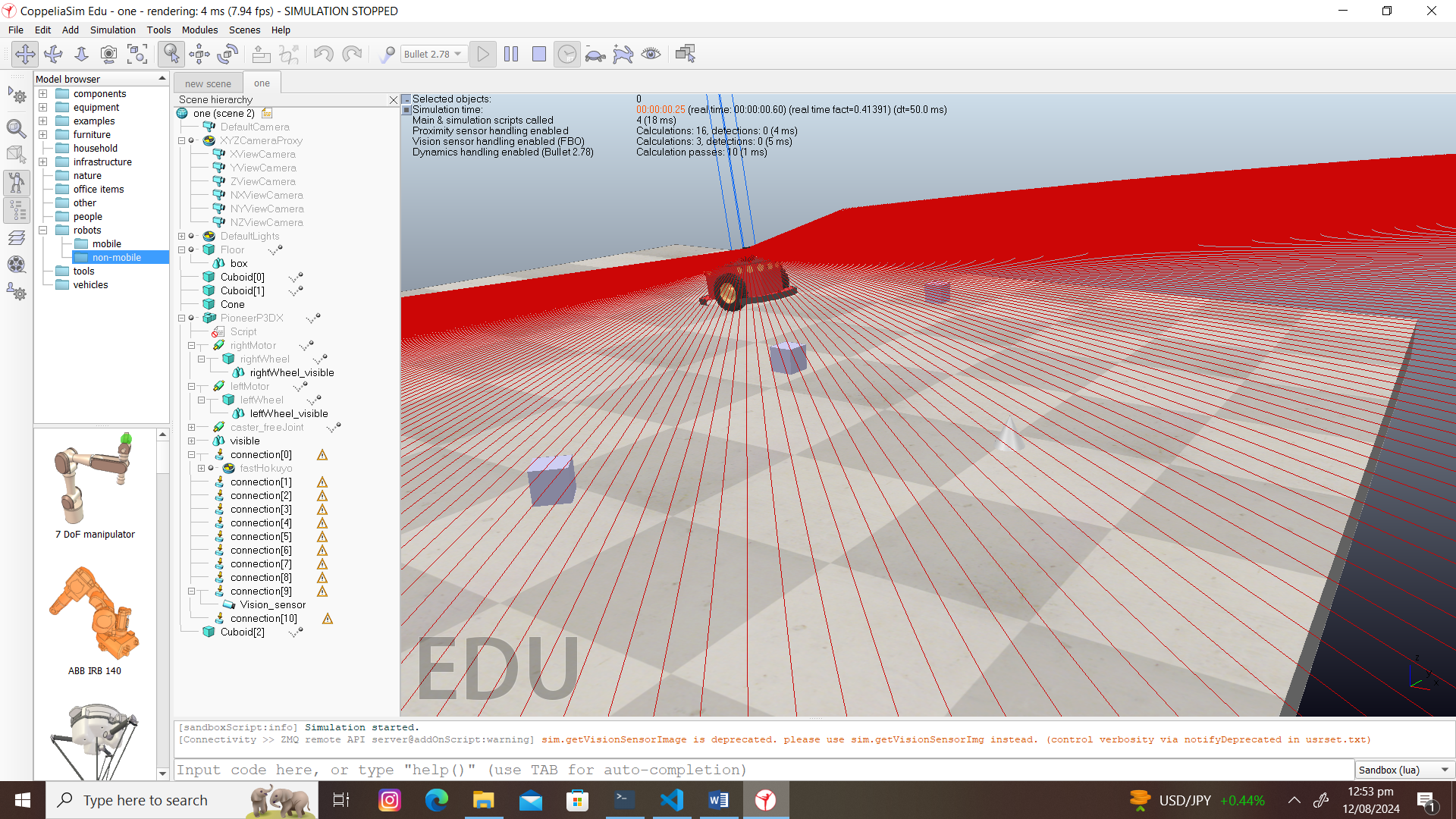


Figure-6 (The mobile robot is navigating while an obstacle is placed in route)

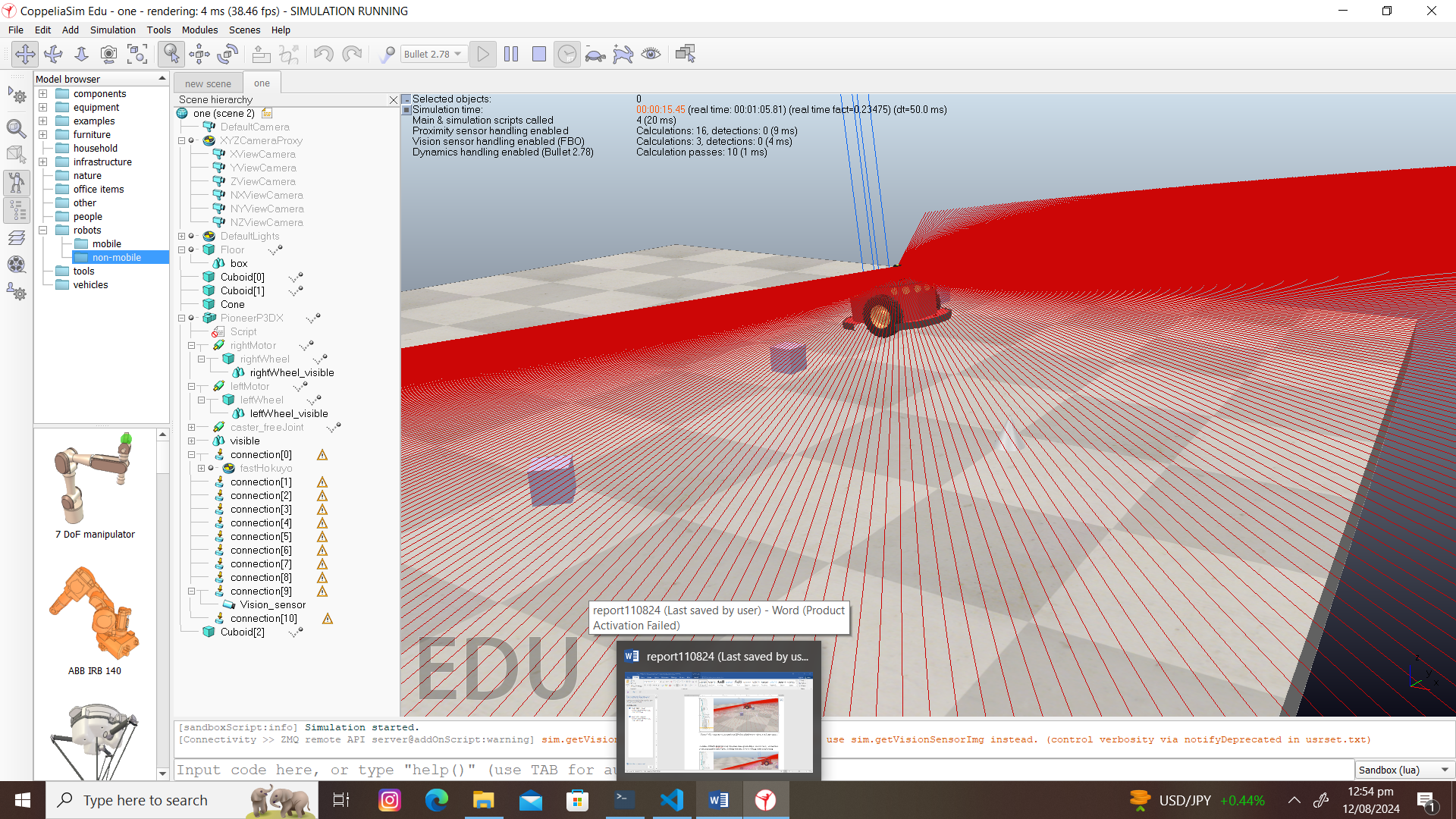


Figure-7 (The vision sensor and addition of LIDAR is helpful to detect obstacle and alter course)

Addition of LIDAR (HokuYo) along with vision sensor greatly help to detect obstacles. It is however pertinent to mention that robot was colliding with these obstacles during early part of training.



Figure-9 (the robot is reaching towards its target)

The study uses Artificial Potential Field to avoid obstacle and move towards goal

 def calculate\_action(self, state):

        x, y, theta = state[0], state[1], state[2]

        goal\_position = self.sim.getObjectPosition(self.target\_handle, -1)

        # Attractive force towards the goal

        k\_attr = 4.0  # Increased attractive force gain

        f\_attr\_x = k\_attr \* (goal\_position[0] - x)

        f\_attr\_y = k\_attr \* (goal\_position[1] - y)

        # Repulsive force away from obstacles with lookahead

        k\_rep = 0.8  # Slightly increased repulsive force gain

        f\_rep\_x, f\_rep\_y = 0.0, 0.0

        robot\_position = np.array([x, y])

        lookahead\_distance = 0.5  # Lo

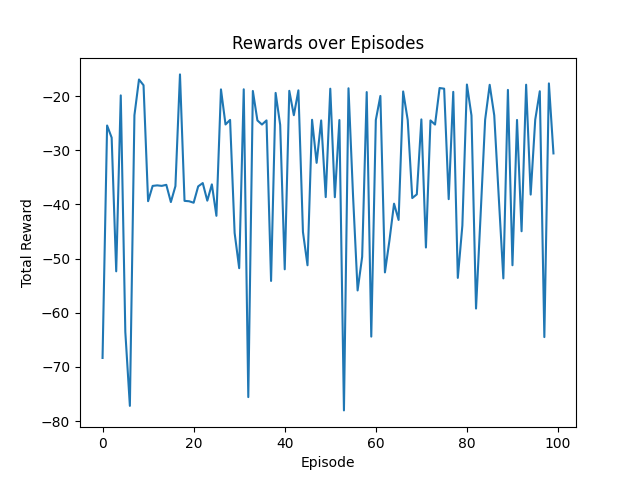


Figure-9(graphical representation training progress)

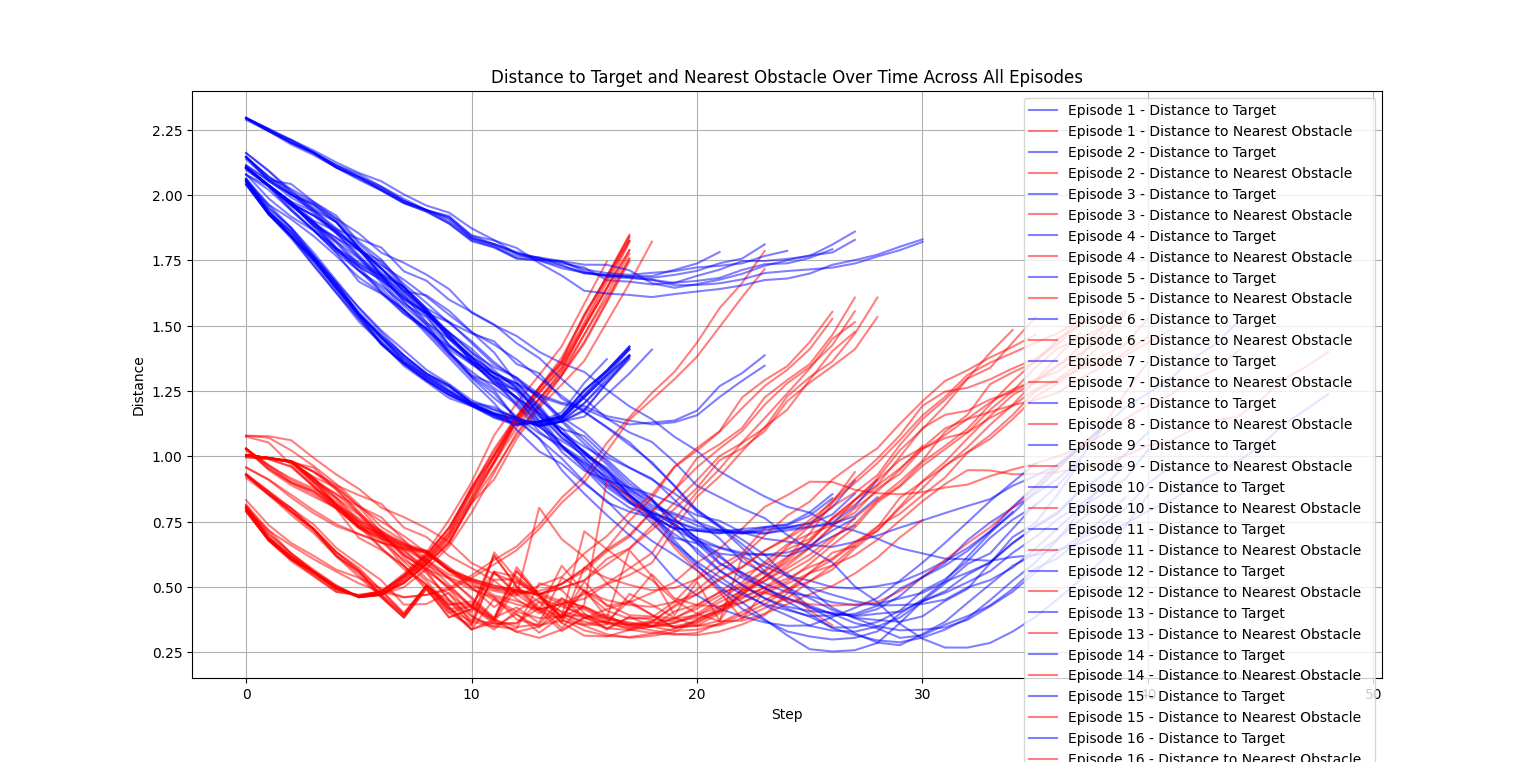


Figure -10

1. **Discussion**

Based on results of experiment carried out using coppeliaSim simulator it is found that Critic Network: Evaluates the quality of actions taken by the robot in various states, thus, helping to guide the learning process. The training process: Involves sampling experiences, computing target Q-values, and minimizing the loss between predicted and target Q-values to train the Critic.: The Actor is updated to maximize the Q-values predicted by the Critic, ensuring the robot learns to take actions that lead to higher cumulative rewards. The addition of vision sensor and LIDAR greatly helped to detect obstacles and targets. Obstacle avoidance logic was used by the DDPG agent gradually with some initial collisions with obstacles. This study uses potential field where movement of robot is controlled by two force ie attractive force and repulsive force.

The study opens new vistas for research on adaptive systems. The implementation of reinforcement method with neural network augments performance of agent to adjust in changing environment which is foremost requirement to be an adaptive system. The study , However it can be improved by incorporating following :

**Fixed Exploration Noise**: The exploration noise is set to a fixed standard deviation (exploration\_noise=0.1). A more sophisticated approach would involve using adaptive noise strategies, such as Ornstein-Uhlenbeck process or parameter noise, which can improve exploration in continuous action spaces.

**Fixed Hyperparameters**: The network architectures, learning rates, and other hyperparameters are fixed. Hyperparameter tuning or using automated methods like Bayesian optimization can lead to better performance.

**Single-threaded Training**: The training loop runs in a single thread. For large-scale problems, parallelizing environment interactions and experience sampling can significantly speed up training.

**GPU Utilization**: The current code does not explicitly utilize GPU acceleration. Leveraging GPUs for neural network training can greatly enhance performance, especially for large networks and large batches.Complex program like this requires extensive learning on large data sets. Although 200 episodes with each 100 steps are simulated but increasing this data will yield more concrete results.

# References

|  |  |
| --- | --- |
| [1] | Y.-J. W. H. H. Meng-Yuan Chen, "A Novel Navigation System for an Autonomous," *arXiv:2006.0496.* |
| [2] | A. M. A. a. A. L. Fradkov, "A Historical Perspective of Adaptive Control". |
| [3] | article2, Adaptive Control, Algorithms, Analysis and Applications. |
| [4] | W. U. Christopher I. Byrnes, Systems & Control: Foundations & Applications, Washington: Washington University. |
| [5] | A. Annaswamy, "http://aaclab.mit.edu/material/lect/2\_153\_Lecture\_01.pdf," [Online]. |
| [6] | A. Annaswamy, "[http://aaclab.mit.edu/material/lect/2\_153\_Lecture\_02.pdf," [Online]. |
| [7] | "Feedback Control Systems in Robotics: Types and Examples (linkedin.com)," [Online]. |
| [8] | "A Novel Navigation System for an Autonomous". |
| [9] | O. A. Taner YILMAZ, "Deep deterministic policy gradient reinforcement learning for collision-free navigation of," *doi: 10.5505/fujece.2023.85047.* |
| [10] | T. P. Lillicrap∗, "CONTINUOUS CONTROL WITH DEEP REINFORCEMENT," Google Deepmind. [Online]. |
| [11] | A. S. Kanervisto, "Action Space Shaping in Deep Reinforcement," *arXiv:2004.00980v2.* |
| [12] | "A COMPARISON BETWEEN DEEP Q-LEARNING AND DEEP DETERMINISTIC POLICY GRADIENT FOR AN AUTONOMOUS DRONE IN A SIMULATED ENVIRONMENT," *¨alardalen University.* |
| [13] | W. Rahiman, "A comprehensive study for robot navigationtechniques," *Cognet Enginnering,* vol. https://doi.org/10.1080/23311916.2019.1632046. |
| [14] | "Smart Obstacle Avoidance Using a Danger Index for a Dynamic Environment". |
| [15] | https://doi.org/10.1007/978-3-031-01563-2\_7. |
| [16] | "Principles of Robot motion". |
| [17] | M. University, "LOcal path planning using virtual potential field," [Online]. Available: https://www.cs.mcgill.ca/~hsafad/robotics/. |
| [18] | R. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots.". |
| [19] | T. P. Lillicrap, "CONTINUOUS CONTROL WITH DEEP REINFORCEMENT". |
| [20] | "doi: 10.5505/fujece.2023.85047". |
| [21] | "DOI: 10.1561/2200000071". |
|  |  |
|  |  |
|  |  |

The End