Gazi Üniversitesi
Fen Bilimleri Dergisi
PART C: TASARIM VE TEKNOLOJİ

Gazi University
Journal of Science
PART C: DESIGN AND TECHNOLOGY

GU J Sci, Part C, 12(3): 620-627 (2024)

Evaluation of the Deep Q-Learning Models for Mobile Robot Path Planning Problem

Mehmet GÖK[1*]

[1]*Kahramanmaraş İstiklal University, Faculty of Arhitecture, Engineering, and Design, Department of Computer Engineering, Kahramanmaraş, Turkey*

Article Info

Research article
Received: 20/03/2024
Revision: 05/07/2024
Accepted: 16/08/2024

Keywords

Deep Q-Learning
Mobile Robots
Model Inference
Path Planning

Makale Bilgisi

Araştırma makalesi
Başvuru: 20/03/2024
Düzeltme: 05/07/2024
Kabul: 16/08/2024

Anahtar Kelimeler

Derin Q-Öğrenme
Mobil Robotlar
Model Çıkarımı
Yol Planlama

Graphical/Tabular Abstract (Grafik Özet)

This study proposes an approach to compare the performances of the models pre-trained for mobile robot path planning in terms of path length, path curvature, and journey time. / Bu çalışma, mobil robot yol planlaması için önceden eğitilmiş modellerin yol uzunluğu, yol eğriliği ve yolculuk süresi açısından performansını karşılaştırmak için bir yaklaşım önermektedir.

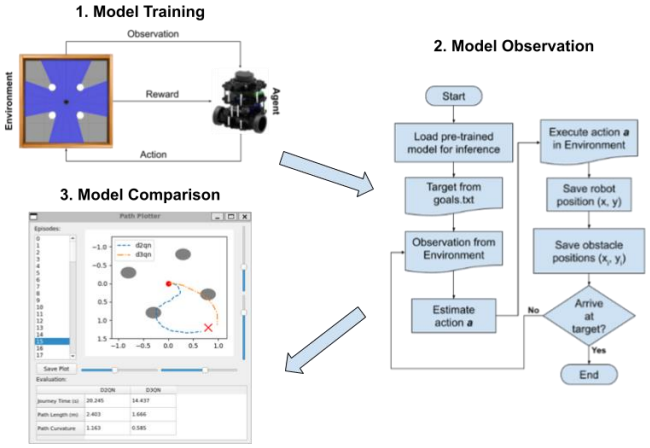


***Figure A**: The flow of the proposed model evaluation approach /**Şekil A:**. Önerilen model değerlendirme yaklaşımının akışı*

Highlights (Önemli noktalar)

- *A comparison approach is devised to evaluate the performances of the Deep Q-learning models trained for mobile robot path path planning tasks. / Mobil robot yol planlama görevleri için eğitilen Derin Q-Öğrenme algoritmalarının performans değerlendirmelerinin yapılabilmesi için bir karşılaştırma yaklamaşımı tasarlanmıştır.*
- *The proposed approach is implemented by means of Robot Operating System (ROS), Python programming language, and PyTorch libraries. / Önerilen yaklaşım Robot İşletim Sistemi, Python programlama Dili ve PyTorch kütüphaneleri ile icra edilmiştir.*
- *The attained results from model inferences are tracked via the proposed scheme. / Model çıkarımlarından elde edilen sonuçlar önerilen akış ile izlenmiştir (kaydedilmiştir).*
- *Evaluation and path planning results are displayed by using an implemented Graphical User Interface (GUI). / Değerlendirme ve yol planlama sonuçları icra edilen Grafiksel kullanıcı arayücü ile görüntülenmiştir.*

***Aim (Amaç):** The aim of this article is to design a comparison approach to evaluate the performances of the pre-trained Deep Q-Learning models for dynamic path planning. / Bu makalede dinamik yol planlama için eğitilen Derin Q-Öğrenme modellerinin değerlendirilebilmesi için bir karşılaştırma yaklaşımının tasarlanması amaçlanmıştır.*

***Originality (Özgünlük):** This paper focuses on the inference performances of the models rather than training performance. / Bu makale modellerin eğitim performanslarından ziyade modellerin çıkarım performanslarına odaklanmıştır.*

***Results (Bulgular):** With the proposed approach, the success of the models in path planning tasks can be observed effectively. / Önerilen yaklaşım ile modellerin yol planlama görevlerindeki başarısı etkin bir şekilde gözlemlenebilmektedir.*

***Conclusion (Sonuç):** The obtained results show that the proposed approach can be used to compare models trained for dynamic path planning tasks with the implemented interface. / Elde edilen sonuçlar, önerilen yaklaşım, icra edilen arayüz ile dinamik yol planlama görevleri için eğitilen modellerin karşılaştırılması amacı kullanılabileceğini göstermektedir.*

*Corresponding author, e-mail: mehmet.gok@istiklal.edu.tr
DOI: 10.29109/gujsc.1455778

GU J Sci, Part C, 12(3): 620-627 (2024)

Gazi Üniversitesi
Fen Bilimleri Dergisi
PART C: TASARIM VE TEKNOLOJİ

Gazi University
Journal of Science
PART C: DESIGN AND TECHNOLOGY

http://dergipark.gov.tr/gujsc

Evaluation of the Deep Q-Learning Models for Mobile Robot Path Planning Problem

Mehmet GÖK[1*]

[1]*Kahramanmaraş İstiklal University, Faculty of Arhitecture, Engineering, and Design, Department of Computer Engineering, Kahramanmaraş, Turkey*

Article Info

Research article
Received: 20/03/2024
Revision: 05/07/2024
Accepted: 16/08/2024

Keywords

Deep Q-Learning
Mobile Robots
Model Inference
Path Planning

Abstract

Search algorithms such as A* or Dijkstra are generally used to solve the path planning problem for mobile robots. However, these approaches require a map and their performance decreases in dynamic environments. These drawbacks have led researchers to work on dynamic path planning algorithms. Deep reinforcement learning methods have been extensively studied for this purpose and their use is expanding day by day. However, these studies mostly focus on training performance of the models, but not on inference. In this study, we propose an approach to compare the performance of the models in terms of path length, path curvature and journey time. We implemented the approach by using Python programming language two steps: inference and evaluation. Inference step gathers information of path planning performance; evaluation step computes the metrics regarding the information. Our approach can be tailored to many studies to examine the performances of trained models.

Mobil Robot Yol Planlama Problemi için Derin Q-Öğrenme Modellerinin Değerlendirilmesi

Makale Bilgisi

Araştırma makalesi
Başvuru: 20/03/2024
Düzeltme: 05/07/2024
Kabul: 16/08/2024

Anahtar Kelimeler

Derin Q-Öğrenme
Mobil Robotlar
Model Çıkarımı
Yol Planlama

Öz

Mobil robotlar için yol planlama problemini çözmek için genellikle A* veya Dijkstra gibi arama algoritmaları kullanılır. Ancak bu yaklaşımların bir harita gereksinimi bulunmakla birlikte dinamik ortamlarda performansları düşer. Bu dezavantajlar araştırmacıları dinamik yol planlama algoritmaları üzerinde çalışmaya yöneltmiştir. Derin takviyeli öğrenme yöntemleri bu amaçla kapsamlı bir şekilde incelenmiş ve bu yöntemlerin kullanımı her geçen gün artmaktadır. Ancak bu çalışmalar çoğunlukla modellerin eğitim performansına odaklanmakta olup çıkarıma dayalı bir performans değerlendirmesi yapılmamaktadır. Bu çalışmada, modellerin performansını yol uzunluğu, yol eğriliği ve yolculuk süresi açısından karşılaştırmak için bir yaklaşım önerilmektedir. Önerilen yaklaşım Python programlama dili kullanılarak çıkarım ve değerlendirme olarak iki adımda gerçeklenmiştir. Çıkarım adımı yol planlama performansı hakkında bilgi toplamakta olup; değerlendirme adımı ise bu bilgilerle ilgili metrikleri hesaplamaktadır. Önerilen yaklaşım, eğitilen modellerin performanslarını incelemek için birçok çalışmaya uyarlanabilir.

1. INTRODUCTION (GİRİŞ)

Path planning in robotics research can be defined as a task for calculating the best path from initial location to the goal location for a mobile robot. Path planning generates sub-locations comprising an obstacle-free trajectory to the target. For this aim, algorithms such as A* and Dijkstra are adopted for the environments have a priori map [1, 2]. However, changes in the environment and dynamic obstacles require re-routing for a pre-planned path in real-world missions. Dynamic path planning methods are applied in such cases and studies are being assessed continuously to obtain a decent dynamic path planning method. Reinforcement learning and Deep Reinforcement Learning methods have become popular in the field of dynamic path planning recently [3-5].

*Corresponding author, e-mail: mehmet.gok@istiklal.edu.tr
DOI: 10.29109/gujsc.1455778

Among the deep reinforcement learning algorithms, algorithms such as Deep Q-Learning and Double Deep Q-Learning are used in studies using discrete action, while algorithms such as Deep Deterministic Policy Gradients are used in studies where continuous action setting is preferred [6, 7]. In this regard, it can be said that there is a trade-off between performance and complexity. According to our experiences, while Deep Q-Learning algorithms are relatively simple, algorithms based on policy gradients are somewhat more complex. Although there are many studies on this subject, the absence of an accessible comparison and testing environment in general makes it difficult to make comparisons between the performances of those approaches. For instance, [4] presents an approach Q-learning and topological maps employing an authentic evaluation setting based on Box2D library. Similarly, [8] employs a 2D simulation tool based on Pyglet package to obtain a better training time. Even individual simulation solutions provide low resource usage, performance, and time gain, reimplementation of these approaches may be cumbersome.

In this manuscript, a study on the design and evaluation of test software that can be used to analyze the performance of Deep Q-learning models is presented. For this aim, we first trained two local path planning models with Double Deep Q-Network (D2QN) and Dueling Double Q-Network (D3QN), then we deployed these models in the proposed test scenario for the inference. The test software drives the mobile robot to 99 randomly generated points and uses the measures of path length traveled, path curvature, number of collisions and travel time to decide which model is better. In addition, the route followed by the robot and the working environment are visualized with the designed graphical user interface (GUI) of the software. Although the proposed scheme has been tested with D2QN and D3QN models, it can also be opted for other deep reinforcement learning models.

The generally accessible and well-documented Turtlebot3 mobile robot and Gazebo simulation software is adopted for the experiments. To facilitate better reproducibility, the experimental environment presented in the Turtlebot3 Machine Learning documentation was chosen as the reference [9]. The reference study employs robot operating system (ROS) infrastructure to establish an appropriate communication between simulation and machine learning software. ROS is a meta-operating system that combines communication and utilities for robotics research. Thanks to the tools it provides and its flexible structure, it became a major robotics research environment [10].

2. REINFORCEMENT LEARNING APPROACH FOR MOBILE ROBOT PATH PLANNING (MOBİL ROBOT YOL PLANLAMA İÇİN TAKVİYELİ ÖĞRENME YAKLAŞIMI)

The Reinforcement learning problems are basically considered as a Markov Decision Process (MDP), modeled as a tuple *(S, A, T, R, γ)*, where *S* and *A* implies the environment's state space and action space, respectively. The term agent in Reinforcement Learning refers to the learner that executes learning by interacting with the environment through trial and error. The environment expresses a world formation in which the agent implements learning tasks [3, 11, 12]. At each time step *t*, the agent takes an action $a_t \in$ A and changes its state from $s_t \in S$ to the new state s_{t+1}. $T(a_t, s_t, s_{t+1}) = P(s_{t+1}|s, a)$ denotes the transition probability for the agent moving from state s_t to the new state s_{t+1} after taking action at. The agent receives a scalar reward r_t from the environment and develops an action selection policy $\pi: S \rightarrow A$. Reward function is defines as $R : S \rightarrow A$ and the agent aims to obtain a maximized sum of discounted future rewards beginning from t_0 under the policy π as given in Eq. 1:

$$R_t = \sum_{t=t_0}^{T} \gamma^{t-t_0} r_t \quad (1)$$

where *T* denotes the terminal time step and $\gamma \in [0,1]$ is the discounting factor. The discounting factor penalizes the value of future rewards due to the uncertainty. Each state can be associated with a value function $V(s_0)$ estimating the expected amount of future rewards can be received beginning from s_0 following the policy π. Value function can be formulated by using Eq. 2:

$$V_\pi(s) = \mathbb{E}[R_t|s_t = s; \pi] \quad (2)$$

The action value represents the gain of an action a_t at state s_t, where the gain is defined as the *expected future reward* under policy π as given in Eq. 3:

$$Q^\pi(s, a) = \mathbb{E}[R_t|s_t = s, a_t = a; \pi] \quad (3)$$

Eq. 3 can also be obtained by adding a condition for action selection to the value function equation Eq. 2. A policy is expressed optimal if it achieves the best expected return and is defined as in Eq. 4:

$$Q^*(s,a) = maxQ^{\pi}(s,a) \quad (4)$$

Q-learning refers to the iterative estimation of the optimal state-action value function based on the Bellman optimality as defined in Eq. 5:

$$Q^*(s_t,a_t) = \mathbb{E}[r_t + maxQ^*(s_{t+1},a)|s_t,a_t] \quad (5)$$

When the agent has the full view of the environment, it can directly calculate Q-values by utilizing Eq. 4 for each state-action pair (Figure 1). However, real world problems mostly represent too many states, which are infeasible and complex to compute one by one. In such cases, convergence approach is utilized instead of direct calculation of the Q-values by employing neural networks [10]. The term Q-Network is a specialized version of a neural network that is opted to converge to the Q-values (Figure 2). The action-state value function is rewritten as $Q(s, a, \theta_t)$ where θ denotes the weights of the Q-Network.

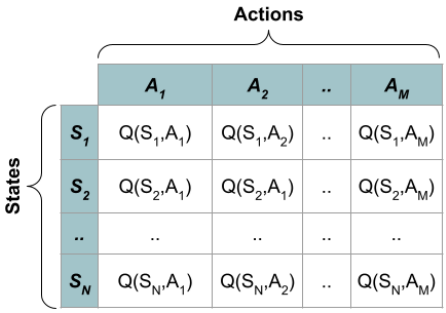


Figure 1. Q-Table filled up by using Bellman Equation for M actions and N states (M aksiyon ve N durum için Q-Tablosunun Bellman Eşitliği ile doldurulması)

Those weights are updated by utilizing past experiences of the agent due to the lack of prior information. In a reinforcement learning environment, moving from an old state to a new state is called a transition. Every transition with a returning reward from the environment is called an experiment. An experiment can be written as a tuple $<s_t, a_t, s_{t+1}, r>$.

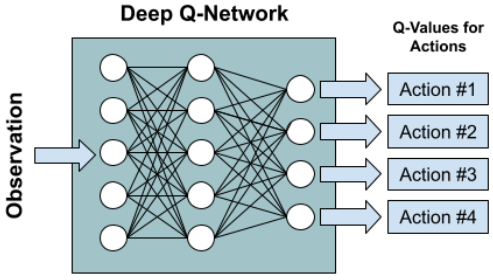


Figure 2. Deep Q-Network and approximated Q-Values (Derin Q-Ağı ve yaklaşılan Q-Değerleri)

Update procedure of the Q-Network can be formulated as in Eq. 6:

$$\theta_{t+1} = \theta_t + \alpha\left(y_y^Q(s_t,a_t;\theta_t)\right)\nabla_{\theta_t}Q(s_t,a_t;\theta_t) \quad (6)$$

Where α is the learning rate and y_t^Q is the target value function. Target value function can be computed using Eq. 7:

$$y_t^Q = R_{t+1} + \gamma \max_a Q(s_{t+1},a;\theta_t) \quad (7a)$$

$$= R_{t+1} + \gamma Q(s_{t+1}, \operatorname{argmax}_a Q(s_{t+1},a;\theta_t);\theta_t) \quad (7b)$$

The studies in the literature adopted Double Deep Q-Network approach where a second neural network whose weights θ^- are updated lately to tackle the instability problems encountered in Deep Q-Network utilization. Eq. 7 can be written as in Eq. 8 to indicate new situation:

$$y_t^{DQN} = R_{t+1} + \gamma \max_a Q(s_{t+1},a;\theta_t) \quad (8a)$$

$$= R_{t+1} + \gamma Q(s_{t+1}, \operatorname{argmax}_a Q(s_{t+1},a;\theta_t);\theta^-) \quad (8b)$$

Dueling Double Q-Network approach is the further optimized version of DQN which taking value function into account to obtain advantage function stated in Eq. 9:

$$Q_\pi(s,a) = V_\pi(s) + \left[A_\pi(s,a) - \frac{1}{|A|}\sum_{a'} A_\pi(s,a)\right] \quad (10)$$

where the computation is realized within the neural network stream by sharing network parameters [13, 14]. The network scheme regarding the Eq. 10 is given Figure 3.

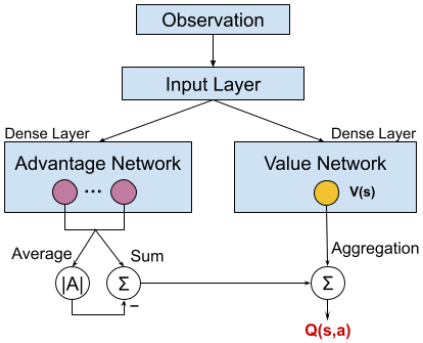


Figure 3. Dueling Q-Network Architecture (Düello Q-Ağı Mimarisi)

When dynamic path planning is considered in the DRL framework, the mobile robot statement replaces the term agent. An example environment with four static obstacles, walls, and robot model is given in Fig. 4 demonstrating a typical application for a mobile robot simulation. For a typical Deep Q-Learning scenario employing a mobile robot for

path planning, a mobile robot explores its surrounding environment by crashing into walls or obstacles. Each formation also containing the robot's position in the environment denotes a state. The mobile robot tries to reach its goal by taking pre-determined actions such as turning or going straight. These actions are chosen based on the robot's observations according to its state in the environment, and each action moves the environment to a new state. Moving from an old state to a new state is called a transition. Every transition with a returning reward from the environment is called an experiment. Reaching the goal or colliding with an obstacle also refers to an experiment during the learning progress [8, 15].

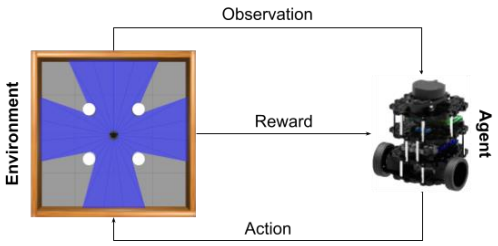


Figure 4. Typical Application of Agent-Environment Interaction Process (Ajan-Ortam etkileşiminin tipi uygulaması)

The observations and rewards are the training data for the Q-value evaluation according to Eq. 8 given in previous section. To facilitate a stable learning process, an experience replay buffer is utilized during neural network training. Experiences of the robot are saved into a table; then randomly selected mini-batches of experiences are used for training (Figure 5). In this way, past experiences are exploited to ensure stable learning progress [16].

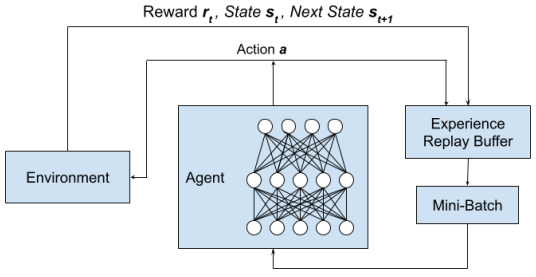


Figure 5. Experience Replay Buffer utilization (Deneyim tekrar kullanım tamponunun kullanılışı)

The simple flow given in Fig. 5 can be summarized as follows:

- The agent selects an action a from action set A regarding its observation and executes it in the environment.
- Action a, current state s_t, next state s_{t+1} after the execution of the action and reward rt are stored into ERB

The DQN model is updated with randomly sampled experiments from ERB. D2QN and D3QN methods, as value-based algorithms, were employed for dynamic path planning by numerous papers such as [4, 14, 19]. These models are preferred due to extensive usage and presenting a straightforward approach to Q-Learning. Hereby, the reference implementation employs D2QN method, and D3QN implementation is easily obtained alongside little changes in the model definition. Integration of policy-based algorithms like Actor-Critic (A2C), and Deep Deterministic Policy Gradients (DDGP) may provide better path planning performance as reported in [10, 16], however, the source code of the reference needs to be altered much for training. We favor the D2QN and D3QN approaches by adhering to the reference code base to present better reproducibility.

3. MATERIALS AND METHODS (MATERYAL VE METOD)

Robot simulations are frequently used in the training of deep reinforcement learning models for mobile robots due to the difficulties encountered in real environments. For instance, a mobile robot can run out of battery during time consuming training process or frequent collisions at the beginning of training can damage the robot. Gazebo and ROS integration is a widely preferred combination of robotics research. Gazebo provides efficient robot modeling and environment simulation capabilities within physics and 3D engines. In addition to this, researchers can interface robots via adequate messages, topics and services defined in ROS. The environment expresses a world formation in which the robot implements learning tasks. For instance, a mobile robot explores its surrounding environment by crashing into walls or obstacles [10, 16, 17].

3.1. Sate and Observation (Durum ve Gözlem)

State can be considered in three measures; distance from the goal point (D_g), mobile robot heading angle from the goal (θ), and 26 laser distance data acquired from Light Detection and Ranging (LiDAR) Unit mounted on the top of the Turtlebot3. Measured D_g and θ are depicted in Figure 6 and LIDAR traces can be seen in Fig. 4. The distance data represents the distances from the walls and

obstacles occupying space in the environment. All the distance measurements clipped into the interval of [0, 3.5]. Observation provides a partial measurement of the environment with the mobile robot and with the training phase [9, 18]. The observation combines these three measures in a vector formation as given in Eq. 11:

$$Observation =< I_1, I_2, .. I_{26}, D_g, \theta > \quad (11)$$

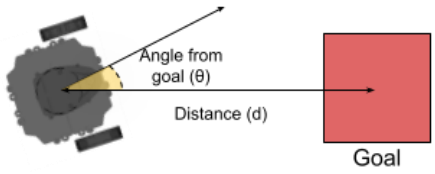
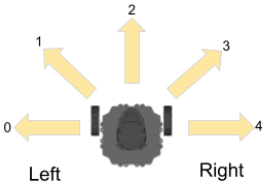


Figure 6. State of the robot with respect to the Goal (Robotun hedefe göre durumu)

3.2. Action Set and Action Selection (Eylem Kümesi ve eylem Seçimi)

In Deep Q-Learning approach, basically, the actions getting the robot close to the goal are rewarded while the actions getting the robot away from the goals are penalized. Actions are discretized into a finite set to ease the implementation of training and inference phases. In this study, Turtlebot3 has a constant linear velocity of 0.15 m/s and angular velocity is determined through action selection policy. The five discrete actions given in Figure 7. Linear velocity is related to the distance and angular velocity is related to the heading angle, respectively.



Action	Angular Velocity (rad/s)
0	-1.5
1	-0.75
2	0
3	0.75
4	1.5

Figure 7. Action set and related angular speeds (Eylem kümesi ve ilgili açısal hızlar)

A mobile robot takes random actions to reach its goal at the beginning of the training. The DQN model is updated according to the taken action and its effect on the state of the robot. During the training progress, action selection relies on the trained model more and random action selection is less preferred. Random action selection expresses the exploration allowing an agent to improve its current knowledge about each action. Taking actions with respect to the model means exploitation that makes the agent greedy about action selection. Even if exploitation seems to be more advantageous from the perspective of obtaining better rewards, it may lead to sub-optimal behavior. Thus, a simple method called *epsilon-greedy* is utilized to balance exploration and exploitation to cope with the dilemma. In this method, a uniform random value $p \in [0,1)$ is determined and compared to the hyperparameter epsilon ε. If p is smaller than ε, a random action is determined, otherwise, current-best action is estimated through the model. At the end of each episode, ε is decreased by multiplying an epsilon decay parameter and, thus the mobile robot exploits its knowledge more [14, 19].

3.3. Experimental Environment (Deney Ortamı)

The experimental environment is the same as given in the Turtlebot3 Machine Learning reference [9] and includes three scenarios with the same dimensions (2mx2m) and different configurations. In all three scenarios, the area where the robot roams are surrounded by a wall. In the first scenario, the working environment of the robot is empty, and the robot is driven to randomly generated targets. In the second scenario, there are four cylindrical fixed obstacles in the mission area. In the third, the situation in which these obstacles move in a circular manner is examined. The point to be noted here is that in Gazebo, these obstacles are grouped as a single object with a circular velocity component (*w=0.5 rad/sn*). As a result, the four obstacles are rotated around the robot and create a dynamic situation for the robot. The simulation environment with static obstacles is depicted in Figure 8. Thanks to Gazebo and ROS integration, various tests can be done easily in the experimental environment.

Reference implementation was conducted on ROS Melodic on Ubuntu 18.04, which both are not supported currently. Thus, in this study, the experiments are carried out with ROS Noetic on Ubuntu 20.04. Furthermore, since ROS Noetic supports Python3, Python 2.7 scripts were rewritten in Python 3.8. We also preferred PyTorch 2.0 library instead of Tensorflow 2.1 due to performance and incompatibility issues we encountered during the study. Tensorflow 1.8, which is utilized in reference, is not available with Python 3.x versions. Remaining package structure is utilized as is, thus, current work can be reproduced and validated for future assessments. We trained D2QN and D3QN models within the aforementioned scenarios with the modified versions Python scripts of the reference

implementation. Finally, we save the trained models for the inference and evaluation steps.

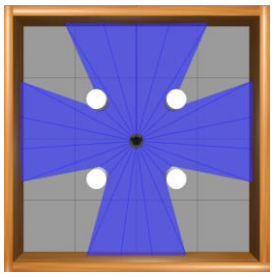


Figure 8. Simulation environment (Simülasyon Ortamı)

Inference step (script) utilizes saved model to collect information to assess the driving performance. The script simply subscribes to the *odom* topic of Turtlebot3 node and saves position and time information during robot's journey to the goal; thus, inference script can be adopted to evaluate other reinforcement learning models utilized in ROS environments. In the evaluation step, saved information is interpreted by evaluation script to make a comparison between trained models.

3.4. Model Inference (Model Çıkarımı)

The experimental software consists of two Python script files. The inference script, *tb3_drl.py*, reads the previously trained model and the coordinates from the target coordinate file *goals.txt*. There are 99 randomly generated goal points in the *goals.txt*. It takes observations by spawning each target in the Gazebo environment and starts to predict the actions that will drive Turtlebot3 to the target. The program flow is illustrated in Figure 9. Here, each task to go to a destination can be called an episode. Each step of the episode is called a step. An episode is completed each time the robot reaches the target without collision.

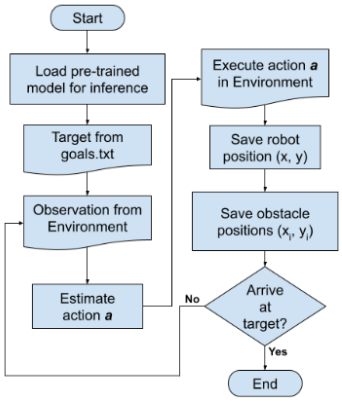


Figure 9. Inference Flow (Model Çıkarımı)

The script file records information of each episode in a text file called *episodes.txt*. Each of row of the text file contains the fields given in Fig. 10. The field *travel time* gives the elapsed time to reach the goal; *Succeeded* field states whether the mobile robot reached the target successfully.

Episode Number (*int*)	X value for Goal Coordinate (*float*)	Y value for Goal Coordinate (*float*)	Travel Time (*float*)	Succeeded? (*Boolean*)

Figure 10. Information row for each episode (Herbir episode için alınan bilgi satırı)

In addition to this, the information for each step is recorded in discrete text files named with episode numbers (*0.txt, 1.txt, .., 99.txt*). Fig. 11. shows the positions of the robot and the obstacles for each step in a row. In this way, evaluation information is generated for each goal.

Robot Position *(x, y)*	Obstacle #1 Position (x_1, y_1)	Obstacle #2 Position (x_2, y_2)	Obstacle #3 Position (x_3, y_3)	Obstacle #4 Position (x_4, y_4)

Figure 11. Positions row for each step (Herbir adım için alınan pozisyon satırı)

Thus, by recording the position of both the robot and the obstacles at each step, the trajectory followed by the robot can be drawn, the path length and path curvature can be easily calculated. The higher values of the road curvature metric indicate that the robot performs the turning maneuver more. This means that the robot consumes more energy on the way to the target.

Path length and path curvature demonstrate the performance of the model under test. In Figure 12, a sample experimental model is given, as well as the screenshot of the evaluation application where the calculations are made. As seen here, the metrics related to the evaluation of the model are shown. The two models used here can be easily compared with each other via the evaluation tool.

3.5. Model Evaluation (Model Değerlendirme)

Evaluation script displays a GUI (Graphical User Interface) to reveal the performance of each model under consideration clearly. It reads previously saved information from *episodes.txt* and from each file involving position data in connection with episodes (*0.txt, 1.txt, .., 99.txt*). The metrics *path length* and *path curvature* are computed by placing trajectory points in Eq. 12 and Eq. 13., respectively.

$$Path\ Length = \sum_{i=0}^{N-1} \sqrt{(x_{i+1} - x_i) + (y_{i+1} - y_i)} \qquad (12)$$

$$Path\ Curvature = \frac{1}{N} \sum_{i=0}^{N-1} \left| arctan2\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) \right| \qquad (13)$$

In Eq. 12 and Eq. 13, N denotes the number of states the mobile robot must pass through to reach the goal; (x_i, y_i) and (x_{i+1}, y_{i+1}) denotes the coordinates of the mobile robot between consecutive states. In Eq. 13, the average of the directional changes is calculated using the built-in function *arctan2* with the Python3 math library [15].

To showcase the advantages of each model, pre-determined metrics are shown in the GUI (Figure 12). There's a sample journey belonging to Stage 3 with moving obstacles. Thus, trajectories in the plot area seems to coincide with obstacles. The trajectory generated by D3QN model is better in terms of journey time, path length and path curvature. Our mobile robot adopting D2QN model, follows a policy which tends to avoid the obstacles causing an inefficient path generation. On the contrary, our robot adopting D3QN model, tracks a more straightforward path to the goal.

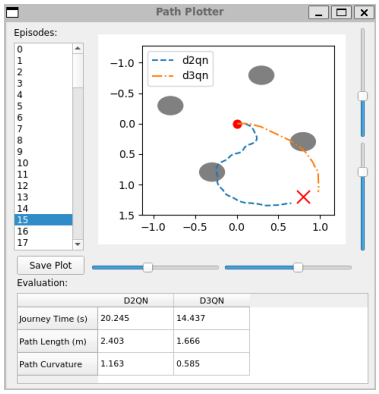


Figure 12. User interface of evaluation script
(Değerlenrime betiğinin Kullanıcı Arayüzü)

Track visualization in the GUI provide insight how the employed model performs path planning. Orange colored path exhibits the effect of the value branch in the D3QN structure to the path planning performance enhancing the way of reacting to the dynamic obstacles. The paths ending in a distant location from target are regarded as collision. The collision case is evident in journey time values relatively low. Since, the shape of the paths does not provide detailed information about turning maneuvers during the journey, we utilize path curvature to obtain a measure of changes in angular speed. The models tend to keep the angular speed close to zero as possible can be evaluated as better among utilized models due to low energy consumption. Eventually, generated paths infer the integration of immediate reactions of the model on the path to the goal location.

The scale of the plot area can be adjusted using horizontal and vertical sliders in the GUI. However, for different obstacle configurations, evaluation script should be modified accordingly. For this purpose, configuration can be parsed through launch files of ROS nodes. Although our approach present insights about model deployment for the scenarios considered is limited to Turtlebot3 Machine Learning repo, it can be adopted to new settings providing accurate localization information at certain time intervals. Since maples approaches lack of adaptive localization compensations, we assume that we receive correct positions through *odom* topic of the robot simulations.

4. CONCLUSIONS (SONUÇLAR)

Although there are numerous studies conducted on dynamic path planning with DRL methods, the performance evaluation is mainly focused on training performance considering total cumulative rewards. However, model performance can be further assessed by use of inference methods and appropriate metrics. This present study proposes a method for evaluating the performance of DRL models within the ROS framework and the Gazebo environment. An application is carried out to enable researchers to better assess the outcomes of the DRL models to be deployed within dynamic path planning. This application consists of two steps: first, the information related to the inference test is recorded, then the recorded data is evaluated to decide which model performs better. Planned paths are visualized and associated metrics are calculated to show the performance of each model. In the end, the D3QN method was found to outperform the D2QN model in inference tests. The proposed testing scheme could be used for any other DRL model with simple modifications.

DECLARATION OF ETHICAL STANDARDS
(ETİK STANDARTLARIN BEYANI)

The author of this article declares that the materials and methods they use in their work do not require ethical committee approval and/or legal-specific permission.

Bu makalenin yazarı çalışmalarında kullandıkları materyal ve yöntemlerin etik kurul izni ve/veya yasal-özel bir izin gerektirmediğini beyan ederler.

AUTHORS' CONTRIBUTIONS (YAZARLARIN KATKILARI)

***Mehmet GÖK*:** He conducted the experiments, analyzed the results and performed the writing process.

Deneyleri yapmış, sonuçlarını analiz etmiş ve maklenin yazım işlemini gerçekleştirmiştir.

CONFLICT OF INTEREST (ÇIKAR ÇATIŞMASI)

There is no conflict of interest in this study.

Bu çalışmada herhangi bir çıkar çatışması yoktur.

REFERENCES (KAYNAKLAR)

[1] H. Aydemir, M. Tekerek, and M. Gök, "Complete coverage planning with clustering method for autonomous mobile robots", Concurr. Comput. Pract. Exp., 2023, doi:10.1002/cpe.7830

[2] M. Gök, Ö. Ş. Akçam, and, M. Tekerek, "Performance Analysis of Search Algorithms for Path Planning", Kahramanmaraş Sütçü İmam University Journal of Engineering Sciences, 26 (2), 379-394., doi:10.17780/ksujes.1171461

[3] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning", in 4th International Conference on Learning Representations, 2016, pp. 1-14.

[4] Y. Kato, K. Kamiyama, and K. Morioka, "Autonomous robot navigation system with learning based on deep Q-network and topological maps", in 2017 IEEE/SICE International Symposium on System Integration, 2018, pp. 1040-1046.

[5] A. I. Karoly, P. Galambos, J. Kuti, and I. J. Rudas, "Deep Learning in Robotics: Survey on Model Structures and Training Strategies", IEEE Trans. on Systems, Man, and Cybernetics: Systems, vol. 51, no. 1, pp. 266–279, 2021.

[6] H. Van Hasselt, "Double Q-learning", in 24th Annual Conference on Neural Information Processing Systems, 2010, pp. 1–9.

[7] A. Kamalova, S. G. Lee, and S. H. Kwon, "Occupancy Reward-Driven Exploration with Deep Reinforcement Learning for Mobile Robot System", Applied Sciences (Switzerland), vol. 12, no. 18, 2022.

[8] J. Gao, W. Ye, J. Guo, and Z. Li, "Deep reinforcement learning for indoor mobile robot path planning", Sensors, vol. 20, no. 19, 2020, pp. 1–15.

[9] Turtlebot3 ROBOTIS e-Manual. https://emanual.robotis.com/docs/en/platform/turtlebot3/machine_learning/ (accessed Sept. 15, 2023).

[10] J. Tsai, C. C. Chang, Y. C. Ou, B. H. Sieh, and Y. M. Ooi, "Autonomous Driving Control Based on the Perception of a Lidar Sensor and Odometer", Applied Sciences (Switzerland), vol. 12, no. 15, 2022.

[11] T. Ribeiro, F. Gonçalves, I. Garcia, G. Lopes, and A. F. Ribeiro, "Q-Learning for Autonomous Mobile Robot Obstacle Avoidance", in 19th IEEE International Conference on Autonomous Robot Systems and Competitions, 2019.

[12] M. C. Bingöl, (2021). Investigation of the Standard Deviation of Ornstein - Uhlenbeck Noise in the DDPG Algorithm. Gazi University Journal of Science Part C: Design and Technology, 9(2), 200-210. https://doi.org/10.29109/gujsc.872646

[13] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Frcitas, "Dueling Network Architectures for Deep Reinforcement Learning", in 33rd International Conference on Machine Learning, vol. 4, no. 9, 2016, pp. 2939–2947.

[14] R. Van Hoa, L. K. Lai, and L. T. Hoan, "Mobile Robot Navigation Using Deep Reinforcement Learning in Unknown Environments", International Journal of Electrical and Electronics Engineering (SSRG-IJEEE), vol. 7, no. 8, 2020, pp. 15–20.

[15] U. Orozco-Rosas, K. Picos, J. J. Pantrigo, A. S. Montemayor, and A. Cuesta-Infante, 'Mobile Robot Path Planning Using a QAPF Learning Algorithm for Known and Unknown Environments', IEEE Access, vol. 10, no. August, 2022, pp. 84648–84663.

[16] M. Wu, Y. Gao, A. Jung, Q. Zhang, and S. Du, "The actor-dueling-critic method for reinforcement learning", Sensors, vol. 19, no. 7, 2019, pp. 1–20.

[17] H. Aydemir, M. Tekerek, and M. Gök, "Examining of the effect of geometric objects on slam performance using ROS and Gazebo", El-Cezeri Journal of Science and Engineering, vol. 8, no. 3, 2021, pp. 1441–1454.

[18] M. Luong and C. Pham, "Incremental Learning for Autonomous Navigation of Mobile Robots based on Deep Reinforcement Learning", Journal of Intelligent & Robotic Systems, vol. 101, no. 1, 2021, pp. 1–11.

[19] M. F. R. Lee and S. H. Yusuf, "Mobile Robot Navigation Using Deep Reinforcement Learning", Processes, vol. 10, no. 12, 2022.