ROS Compatible Local Planner and Controller Based on Reinforcement Learning

Muharrem Küçükyılmaz and Erkan Uslu(✉)

Computer Engineering, Yıldız Technical University, İstanbul, Turkey
muharrem.kucukyilmaz@std.yildiz.edu.tr, euslu@yildiz.edu.tr

Abstract. The study's main objective is to develop a ROS compatible local planner and controller for autonomous mobile robots based on reinforcement learning. Reinforcement learning based local planner and controller differs from classical linear or nonlinear deterministic control approaches using flexibility on newly encountered conditions and model free learning process. Two different reinforcement learning approaches are utilized in the study, namely Q-Learning and DQN, which are then compared with deterministic local planners such as TEB and DWA. Q-Learning agent is trained by positive reward on reaching goal point and negative reward on colliding obstacles or reaching the outer limits of the restricted movable area. The Q-Learning approach can reach an acceptable behaviour at around 70000 episodes, where the long training times are related to large state space that Q-Learning cannot handle well. The second employed DQN method can handle this large state space more easily, as an acceptable behaviour is reached around 7000 episodes, enabling the model to include the global path as a secondary measure for reward. Both models assume the map is fully or partially known and both models are supplied with a global plan that does not aware of the obstacle ahead. Both methods are expected to learn the required speed controls to be able to reach the goal point as soon as possible, avoiding the obstacles. Promising results from the study reflect the possibility of a more generic local planner that can consume in-between waypoints on the global path, even in dynamic environments, based on reinforcement learning.

Keywords: q-learning · deep reinforcement learning · DQN (deep q-learning) · ROS (robot operating system) · navigation · local planner

1 Introduction

In recent years, Autonomous Mobile Robots (AMR) have started to be preferred to improve operational efficiency, increase speed, provide precision, and increase safety in many intralogistics operations, like manufacturing, warehousing, and hospitals [1]. AMRs are a type of robot that can understand its environment and act independently. In order for AMR to understand the environment and act independently, it must know where it is and how to navigate. This topic has attracted the attention of many researchers and various methods have been proposed. One of the most widely accepted and used methods today is to first map the environment and use this map for localization and navigation.

Z. Şen et al. (Eds.): IMSS 2023, LNME, pp. 402–414, 2024.
https://doi.org/10.1007/978-981-99-6062-0_37

Localization and mapping play a key role in autonomous mobile robots. Simultaneous localization and mapping (SLAM) is a process by which a mobile robot can build a map of an unknown environment using sensors which perceive the environment and at the same time use this map to deduce its location [2]. For this task, sensors such as cameras, range finders using sonar, laser, and GPS are widely used. Localization can be defined as the information of where the robot is on the map. Localization information can be obtained from the wheel odometer. But the error in the wheel odometer is incremental over time [3]. Therefore, robot position can be found with triangulation formula by using beacon [4] or reflector [3]. In addition, it is possible to improve the position with object detection using a depth camera [5] or by matching the measurements taken from the real environment with the map using lidar sensors. Navigation can be defined as the process of generating the speed commands necessary for a robot to reach a destination successfully. The navigation methods for mobile robots can roughly be divided into two categories: global planner and local planner. Global planner methods such as A*, Dijkstra and Rapidly exploring Random Tree (RRT) aim to find a path consisting of free space between start and goal. The second submodule in navigation is a local planner. Local planner methods such as Artificial Potential Field, Dynamic Window Approach (DWA), Time Elastic Band (TEB) and Model Predictive Control (MPC) aim to create the necessary velocity commands to follow the collision free trajectory that respects the kinematic and dynamic motion constraints.

In the last decade, Reinforcement Learning (RL) methods have been used for navigation [6, 7] RL is an area of machine learning concerned with how agents take actions in an environment by trial and error to maximize cumulative rewards [8]. Many methods such as Value Function, Monte Carlo Methods, Temporal Difference Methods, and Model-based algorithms have been proposed to maximize the cumulative rewards for agents. Q-learning learning proceeds similarly to method of temporal differences (TD) is a form of model-free reinforcement learning [9]. However, since these models are insufficient in the complexity of the real world, more results are obtained with deep learning-based reinforcement learning methods such as Deep Q-Network (DQN) [1, 10]. In this study, a new local planner method called dqn_local_planner is proposed using the DQN algorithm. For DQN algorithm, a fully connected deep-network is used to generate velocity data for autonomous mobile robots using laser field scanner data, robot position and global path after a series of preprocessing. ROS framework, gazebo simulation and gym environment are used for model training and testing. To compare the proposed model with teb_local_planner and dwa_local_planner, static and dynamic environments are created in the gazebo. The paper is structured as follows: in Sect. 2, algorithms used for proposed method and comparison are mentioned, the explanation of proposed method is mentioned in Sect. 3, the experimental environment set up to test the model is mentioned in Sect. 4, the results obtained and the evaluation of the results are mentioned in Sect. 5 and a general evaluation is made in Sect. 6.

2 Methodology

2.1 Time Elastic Band

Time Elastic Band (TEB) is a motion planning method for local navigation. The classic 'elastic band' is described as a sequence of n intermediate robot poses. TEB is augmented by the time that the robot requires to transit from one configuration to the next configuration in sequence. Its purpose is to produce the most ideal velocity command for mobile robots by optimizing both the configuration and the time interval with weighted multi-objective optimization in real-time considering the kino-dynamic constraints such as velocity and acceleration limits. The objective function is defined as in Eq. (1), in which γ denotes weights and f_k denotes the constraints and objectives with respect to trajectory such as obstacle and fastest path.

$$f(B) = \sum\nolimits_k \gamma_k f_k(B) \tag{1}$$

The f_k are generalized in TEB as in Eq. (2), in which $x_\mathfrak{r}$ denotes the bound, S denotes the scaling and n denotes the polynomial order and ϵ denotes the small translation of the approximation.

$$e_\mathfrak{r}(x, x_\mathfrak{r}, \epsilon, S, n) \cong \begin{cases} \left(\frac{x-(x_\mathfrak{r}-\epsilon)}{S}\right)^n & if\ x > x_\mathfrak{r} - \epsilon \\ 0 & otherwise \end{cases} \tag{2}$$

2.2 Dynamic Window Approach

Dynamic Window Approach (DWA) aims to find the ideal velocity (v, w) search space containing the obstacles-safe areas under the dynamic constraints of the robot and find the maximum velocity in this space using objective function. To get the velocity search space, circular trajectories (curvatures) consisting of pairs (v, w) of translational and rotational velocities are determined. Then, this space is constrained by the admissible velocity at which it can stop without reaching the closed obstacle on the corresponding curvature. Finally, this space is constrained by the dynamic window which restricts the admissible velocities to those that can be reached within a brief time interval given the limited accelerations of the robot. The objective function as shown in Eq. (3) is maximized.

$$\mathrm{G(v, w)} = \sigma(\alpha * \mathrm{heading(v, w)} + \beta * \mathrm{dist(v, w)} + \gamma * \mathrm{vel(v, w)}) \tag{3}$$

The target heading, $heading(v, w)$, measures the alignment of the robot with the target direction. It is maximal if the robot moves directly towards the target. The clearance, $dist(v, w)$, is the distance to the closest obstacle that intersects with curvature. $vel(v, w)$ Function represents the forward velocity of robot.

2.3 Deep Q-Network

DQN is one of the methods of learning optimum behavior by interacting with environment by taking action, observing the environment, and rewarding. In general, observation at time t does not summarize the entire process, so previous observations and actions also should be included in the process. In DQN method, sequences of actions and observations, $s_t = x_1, a_1, x_2, : : : a_{t-1}, x_t$ are included as inputs. The agent's goal is to choose actions that will maximize future rewards as shown in Eq. (4), in which T is the time-step and γ is the discounted factor.

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t \tag{4}$$

The optimal action-value function $\mathcal{Q}^*(s, a)$ allows us to determine the maximum reward we get when we take action a while in state s. The optimal action-value function can be explained as Eq. (5) with the Bellman equation.

$$\mathcal{Q}^*(s, a) = \left[r + \gamma \underset{s'}{max} \mathcal{Q}^*\left(s', a'\right) \mid s, a \right] \tag{5}$$

This can be obtained by iterative methods such as the value iteration algorithms, but it is impractical. It is more common to estimate the action-value function $\mathcal{Q}(s, a; \theta) \approx \mathcal{Q}^*(s, a)$ usually using linear function approximators and sometimes nonlinear approximators instead, such as neural network. Neural network function approximator with weights θ is used as Q-network. However, nonlinear approximators may cause reinforcement learning to be unstable or even to diverge. As a solution to this, replay memory is used that updates iteratively the target values periodically towards action-values $\mathcal{Q}$. Deep Q-learning with experience replay algorithm can be given as follows:

Deep Q-learning with experience replay

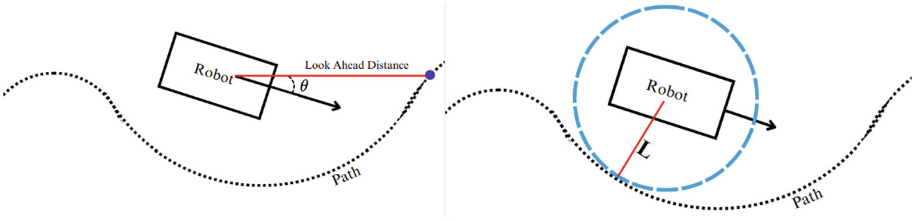
1. Initialize replay memory D to capacity N
2. Initialize action-value function Q with random weights θ
3. Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
4. For episode = 1, M do
   a. Initialize sequence $s_1 = \{x_1\}$ and pre-processed sequence $\phi_1 = \phi(s_1)$
   b. For $t = 1, T$ do
      (1) With probability ε select a random action a_t
      (2) Otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
      (3) Execute action a_t in emulator and observe reward r_t and image x_{t+1}
      (4) Set $s_{t+1} = s_t, a_t x_{t+1}$ and pre-process $\phi_{t+1} = \phi(s_{t+1})$
      (5) Store transition $(\phi_j, a_j, r, \phi_{j+1})$ in D
      (6) Sample random minibatch of transitions $(\phi_j, a_j, r, \phi_{j+1})$ from D
      (7) Set $y_j = \begin{cases} r_j, & if\ episode\ term.\ at\ step\ j+1 \\ r_j + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & otherwise \end{cases}$
      (8) Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
      (9) Every C steps reset $\hat{Q} = Q$
   c. End for
5. End for

3 Proposed Method

This study proposes a new local planner for Mobile Robots using the DQN method. The state space consists of 14 values. The continuous state space that consists of sensor measurements and robot-path measures is sampled at a certain resolution to obtain a discrete state space. One of the state space parameters is the angle difference as shown in Fig. 1a. The vector passing from the intermediate point as far as the lookahead distance of global path and robot's position is obtained. The angle difference (θ) is the difference between this vector and robot angle. The resolution of the angle difference is 0.35 rad. The second is the distance (L) of the robot to its closest point on the global path as shown in Fig. 1b the resolution of the distance is 0.2 m.

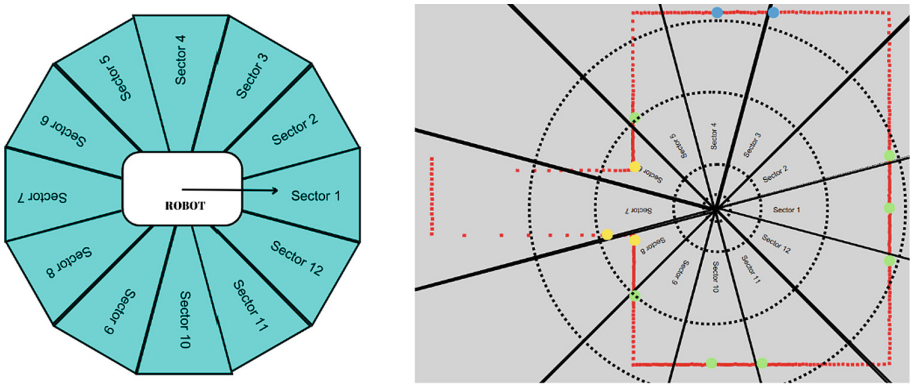
The third is the sensor data obtained from the preprocessed laser scanner. The 360-degree sensor data is divided into 12 sectors as shown in Fig. 2a. The smallest distance data in each sector is a state in the state space as shown in Fig. 2b. The resolution of this distance is determined according to Table 1. Thus, the model is able to review its decisions more precisely as it gets closer to the obstacle.

3 actions have been determined for our model. Go forward at 1 m/s linear velocity, slowly turn right at −0.5 rad/s angular velocity and 0.3 m/s linear velocity, slowly turn



(a) (b)

Fig. 1. Two of the state space parameters a) angle difference between lookahead distance and robot heading, b) the distance (L) of the robot to its closest point on the global path.



(a) (b)

Fig. 2. a) sector tiling around the robot, b) smallest distances of laser data on each sector

Table 1. The resolution table by distance

Min Distance (m)	Max Distance (m)	Resolution (m)
0.0	0.2	0.05
0.2	1.0	0.1
1.0	5.0	0.5
5.0	15.0	1.0
15.0	max range	5.0

left at 0.5 rad/s angular velocity and 0.3 m/s linear velocity. Our agent is rewarded with an inverse ratio to the closest distance to the global path, an inverse ratio to the angular difference and a reward if the robot reaches the goal. However, a negative reward is given if it is 3m away from the global path, if it got too close to the obstacle or if the angle difference was more than 2.44 rad. Expect for the first two rewards, the episode ends in other positive/negative reward cases. There are 3 fully connected layers with a rectified

linear activation function in our network used for DQN agent. These layers consist of 64, 128, 32 and 3 output layers neurons as shown in Fig. 3.

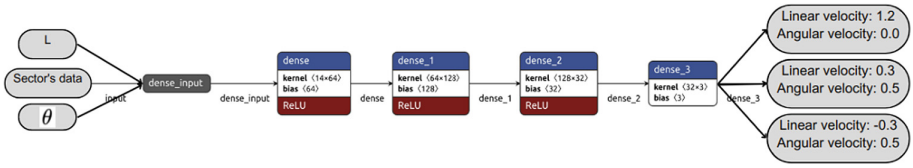


Fig. 3. Network architecture of DQN agent

4 Experimental Setup

The experiment is carried out in a simulation environment. Gazebo, an open-source robotic simulator, is chosen to simulate the environment. Turtlebot3 is chosen for mobile robot. The Turtlebot3 robot is both physically available and has a model in the gazebo environment. It is also compatible with the Robot Operating System (ROS). ROS, which we also use as our experimental setup, is a widely used open-source software for mobile robots. It allows complex algorithms developed for topics that form the basis of robotics such as mapping, localization, navigation used in robots to communicate with each other. In our study, we used 'gmapping' for mapping, 'amcl' for localization and 'move_base' for navigation. We used 'gym', the standard API for reinforcement learning, to observe the environment and determine the next action. We used the 'keras' library for model training. DQN was used as the reinforcement learning model.

In gazebo environment, a (1, 3, 1) size box is placed in the center of the empty space as an obstacle. In each episode, the robot starts from the point $(-3.9, 0)$ and is expected to navigate to the goal at $(3, 0)$. In this way, DQN model is trained for about 5000 episodes. At the end of 5000 episodes, it is observed that the robot mostly navigates to the goal point. As the second stage, the location of the obstacle and goal point is changed in each episode. The obstacle is randomly generated between $(-0.5, 0.5)$ on the x-axis and $(-2, 2)$ on the y-axis. Path given to system as a global path is the straight line between the start point and the target point with 0.2 m waypoints. In this way, the model is trained for about 5000 more episodes. The trained model is integrated into ROS as local planner with the name dqn_local_planner. Scenarios in Table 2 are created to test our model and to compare it with teb_local_planner and dwa_local_planner.

To compare the local planners, the number of times the goal is successfully achieved, the total execution time and the sum of absolute areas between the planned path and the executed path metrics are used. Experiments are repeated 5 times. Reaching a 0.2 m radius circle with the goal in the center is classified as an achieved goal. Final heading angle is not considered.

Table 2. Scenarios

Scenario 1 (Fig. 4a): • Place the robot at point (−3.9, 0.0) • Give a point a. (2.0,0.0) as goal b. (4.0, 0.0) as goal c. (11.0, 0.0) as goal • Trigger global path planning using 'navfn' Place a (1.0, 3.0, 1.0) sized box at (0.0, 0.0) as a fixed obstacle	Scenario 2 (Fig. 4b): • Place the robot at point (−3.9, 0.0) • Give point (4.0, 0.0) as goal • Trigger global path planning using 'navfn' Place a (1.0, 3.0, 1.0) sized box at random $y \in [-2.0, 2.0]$ and fixed x = 0.0 and let the obstacle move repeatedly between −2.0 and 2.0 in y-axis with a speed of 0.2 m/s
Scenario 3 (Fig. 4c): • Place the robot to point (−3.9, 0.0) • Give point (10.0, 0.0) as goal • Trigger global path planning using 'navfn' Place a (1.0, 3.0, 1.0) sized box at random $y \in [-2.0, 8.0]$ and fixed y = 0.0 and let the obstacle first move away from the robot and then repeatedly move between −2.0 and 8.0 in x-axis with a speed of 0.1 m/s	Scenario 4 (Fig. 4d): • Place the robot at point (−3.9, 0.0) • Give point (10, 0.0) as goal • Trigger global path planning using 'navfn' Place a (1.0, 3.0, 1.0) sized box at random $x \in [-2.0, 8.0]$ and fixed y = 0.0 and let the obstacle first move towards the robot and then repeatedly move between 8.0 and 0.0 in x-axis with a speed of 0.1 m/s

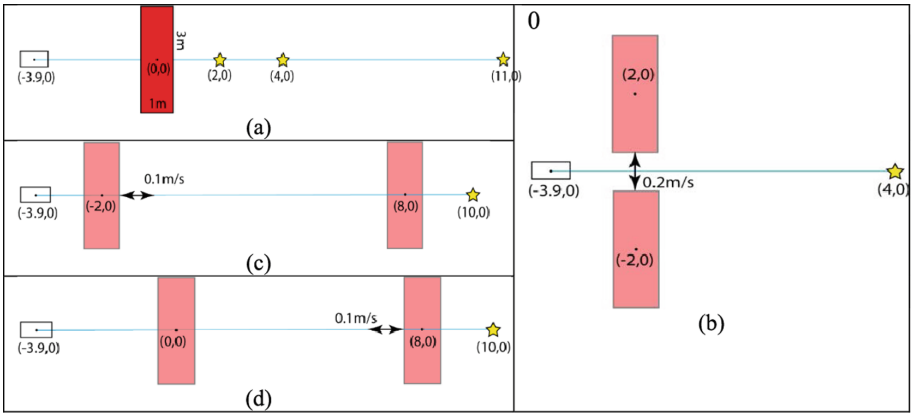


Fig. 4. 2D representation of a) scenario 1, b) scenario 2, c) scenario 3, d) scenario 4

5 Results and Discussion

The graph in Fig. 5 depicts the reward during the training of our model. The model is trained in 10050 episodes. At around episode 4000, our model has learned to go to a fixed goal in a static environment. Then the obstacles and the goal point are randomly generated in a large area and the model is trained for about 2000 more episodes. The drop on reward observed around episode 5000 is due to change in policy, but the robot quickly adapts to the unfamiliar environment. During the training, it is also observed that there was usually no obstacle between the robot and its goal. So, this area is narrowed.

Therefore, the model training decreased again after around episode 6000, but the robot quickly adapts again. It is observed that the model success does not increase after around episode10050, so the model training is terminated here.

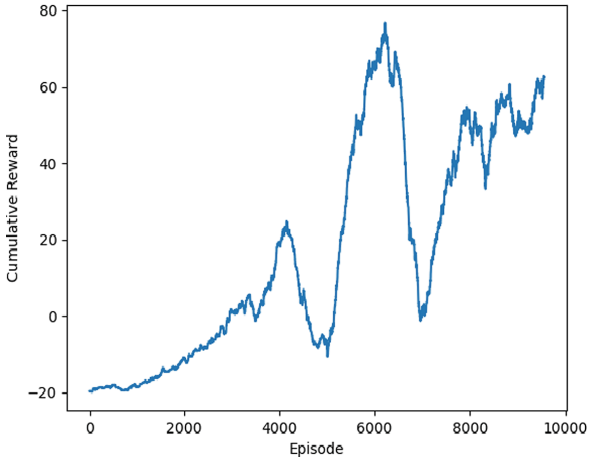


Fig. 5. Cumulative reward graph of proposed model

In the figures below, the blue line represents the global path, green arrows represent the robot's movement and red dots represent the laser sensor readings from the obstacle. Within the scope of scenario 1, the robot is given (4, 0) as goal point. The scenario is repeated 5 times. As shown in Fig. 6a and Fig. 6b, teb_local_planner and dwa_local_planner respectively, fails to pass the obstacle in any of the trials.teb_local_palnner and dwa_local_planner can trigger the move_base to recalculate the path in the case of suddenly encountering an obstacle, however DQN does not have this feature. Normally, there is a parameter in the ROS navigation stack for the global planner to detect dynamic obstacles as well. But here we have turned off that parameter to observe the performance of the pure local planner. As shown in Fig. 6c, the dqn_local_planner reaches the goal in an average of 33.59 ∓ 3.468 s. The robot has moved in an average of 12.032 ∓ 1.121 m away from its path during its entire movement. After passing the obstacle in each attempt, it does not immediately enter the path and goes directly to the goal point.

However, as shown in Fig. 7a, when the goal point is given as (2, 0), the robot could not reach the goal in all trials. If the projection of the robot's position to the path at any time becomes the goal point, it cannot reach the goal. When the goal point is given as (11, 0) as shown in Fig. 7b, the dqn_local_planner reaches the goal in an average of 57.395 ∓ 1.492 s. The robot has moved an average of 14.18 ∓ 1.876 m away from its path during its entire movement. The robot enters the path after passing the obstacle and continues to follow the path. In one of the experiments, the robot passed the goal without reaching the goal, so the experiment was terminated.

In scenario 2, the target is given when the obstacle is in various positions. The local path differs because the obstacle was a moving object as shown in Fig. 8a and Fig. 8b.

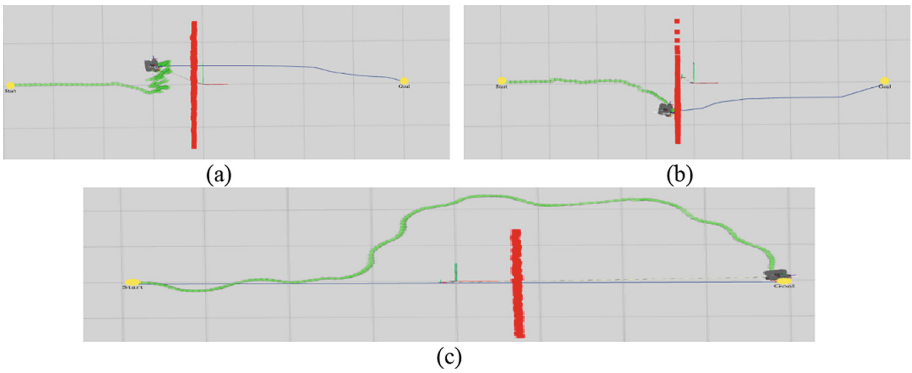


Fig. 6. Scenario 1b results for a) teb_local_planner, b) dwa_local_planner, c) dqn_local_planner

Fig. 7. dqn_local_planner behaviours for a) scenario 1a, b) scenario 1c

The dqn_loca_planner reached the target with an average of 38.88 ∓ 11.040 s. The robot has moved an average of 13.07 ∓ 16.86 m away from its path during its entire movement. It encounters an obstacle once and the alternative path cannot be reproduced. After the obstacle moves away from the robot, the robot continues its path as shown in Fig. 8c.

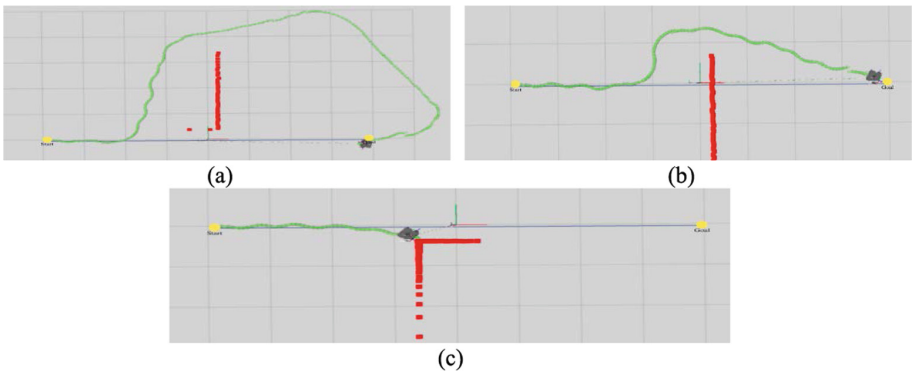


Fig. 8. dqn_local_planner behaviours from different runs for scenario 2 when obstacle is encountered at, a) $y > 0$, b) $y \cong 0$, c) $y < 0$

As shown in Fig. 9a, the teb_local_planner reaches the goal in an average of 25.9 ∓ 1.1 s. The robot has moved an average of 1.71 ∓ 0.244 m away from its path during

its entire movement. When an obstacle is in front of the Robot, teb_local_planner waits until it gets out of the way. After the obstacle is removed, it continues its movement. As shown in Fig. 9b, the dwa_local_planner reaches the goal in an average of 25.13∓4.995 s. The robot has moved an average of 1.41 ∓ 1.374 m away from its path during its entire movement. DWA waits when an obstacle is in front of it and continues after it is removed. However, while the obstacle is moving towards the robot from the side of the robot, it cannot react adequately, and the obstacle hits the robot.

Fig. 9. Scenario 2 behaviours for, a) teb_local_planner, b) dwa_local_planner

In scenario 3, the teb_local_planner and dwa_local_planner could not pass the obstacle as in the first scenario. As shown in Fig. 10, the dqn_local_planner reached the goal in an average of 49.23 ∓ 0.959 s. The robot has moved an average of 22.02 ∓ 5.17 m away from its path during its entire movement. Dqn_local_planner can easily pass slower objects in the same direction.

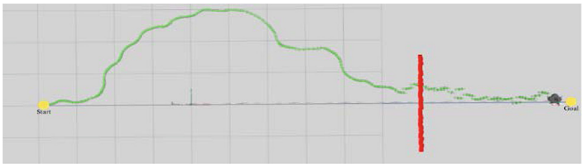


Fig. 10. Scenario 3 behaviour of dqn_local_planner.

Scenario 4 gave equivalent results to scenario 3. Thanks to the robot dqn_local_planner, it easily passed an obstacle coming towards it with a speed of 0.1 m/s.

To summarize the outputs of all scenarios, although teb_local_planner and dwa_local_planner give much better results when there are no obstacles or small obstacles on the path, dqn_local_planner gives better results if there are large obstacles on the robot's path. However, in the current model, if the robot accidentally passes the goal point while following the path, it is exceedingly difficult to return and reach the goal point again. It is usually stuck at the local minimum point as shown in Fig. 7a. The robot may also not take the shortest path while avoiding obstacles. If the robot is too close to the obstacle, it waits until the obstacle disappears and continues its movement. If the obstacle moves to the right in the y-axis forever and the robot encounters the obstacle, if it turns to pass to the right of the obstacle in the first step, they will go together forever in this way. The robot prefers not to go left at any given moment. Table 3 summarizes the result of DQN based local planner, when five tests are carried out for each scenario,

by means of successful navigations, path execution times and error measure about the area between planned and executed paths.

Table 3. Comparison of all scenarios

Metrics	Scn 1			Scn 2	Scn 3	Scn 4
	a	b	c			
Global plan length (m)	5.9	7.9	14.9	7.9	13.9	13.9
The number of times the goal is successfully achieved	0	4	5	4	5	5
Total execution time (s)	–	33.59 ± 3.47	57.39 ± 1.49	38.88 ± 11.04	49.23 ± 0.96	49.34 ± 0.93
Sum of areas between planned and executed paths (m)	–	12.03 ± 1.12	14.18 ± 1.88	13.07 ± 16.86	22.02 ± 5.17	20.58 ± 3.59

6 Conclusion

In conclusion, a new method developed with DQN, which is a reinforcement learning method, is proposed for the local planner. In this method, the environment is first taught to the robot by using sensor data, robot location and global path. For testing the model, fixed, and moving (toward the robot, moving away from the robot, moving vertically) objects that are not in the environment map are added and it is observed whether the robot could reach the goal by avoiding obstacles. The same experiments are also performed with TEB and DWA. To test the performance of the pure local planner, the obstacles that are subsequently introduced into the environment are not added to the global path. While TEB and DWA planners are not successful in avoiding obstacles, our model is able to pass the obstacle easily in all scenarios. If the model does not learn the entire space, local minimum points may occur. In addition, if the robot and the obstacle start parallel movement in the same direction and at a similar speed, they will move together forever. However, our current model is trained in a specific environment and so works in a small world. More complex environments can be selected for training in later models. The action space of our model is also exceedingly small. It includes forward, turn right, and turn left actions. A subspace of each action can be created with a certain resolution.

Moreover, only a fully connected network is used in our model. While teaching more complex spaces, training can be continued with networks such as Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) network. As another future work it is planned also to include the dynamic objects' direction and speed values in the state space, in order to enable the robot to learn not to move in a crossing path with the dynamic object and let the robot move away from the obstacle in an efficient way.

References

1. Fragapane, G., De Koster, R., Sgarbossa, F., Strandhagen, J.O.: Planning and control of autonomous mobile robots for intralogistics: literature review and research agenda. Eur. J. Oper. Res. **294**, 405–426 (2021)
2. Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: part I. IEEE Robot. Autom. Mag. **13**, 99–110 (2006)
3. Betke, M., Gurvits, L.: Mobile robot localization using landmarks. IEEE Trans. Robot. Autom. **13**, 251–263 (1997)
4. Leonard, J.J., Durrant-Whyte, H.F.: Mobile robot localization by tracking geometric beacons. IEEE Trans. Robot. Autom. **7**, 376–382 (1991)
5. Biswas, J., Veloso, M.: Depth camera based indoor mobile robot localization and navigation. In: 2012 IEEE International Conference on Robotics and Automation (2012)
6. Ruan, X., Ren, D., Zhu, X., Huang, J.: Mobile robot navigation based on deep reinforcement learning. In: 2019 Chinese Control and Decision Conference (CCDC) (2019)
7. Heimann, D., Hohenfeld, H., Wiebe, F., Kirchner, F., Quantum deep reinforcement learning for robot navigation tasks, arXiv preprint arXiv:2202.12180 (2022)
8. François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J., et al.: An introduction to deep reinforcement learning. Found. Trends® Mach. Learn. **11**, 219–354 (2018)
9. Watkins, C.J.C.H., Dayan, P.: Q-learning. Mach. Learn. **8**, 279–292 (1992)
10. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)