Deep Reinforcement Learning for Autonomous RC Car Navigation in Dynamic Environments

Ali Bolat
TED University
Email: ali.bolat@tedu.edu.tr

***Abstract*—This research investigates the application of deep reinforcement learning (DRL) in enabling autonomous RC car navigation within dynamic environments, specifically focusing on scenarios without predefined paths. Utilizing MetaDrive as the primary simulation platform, this study designs, tests, and benchmarks DRL algorithms, including deep Q-Network (DQN), double deep Q-Network (DDQN), and Proximal Policy Optimization (PPO). The evaluation emphasizes comparative performance analysis in terms of learning efficiency, adaptability to dynamic conditions, and overall navigation success. The key components of this work include reward shaping to optimize training processes. The findings highlight the relative strengths and weaknesses of each algorithm in navigating complex and dynamic scenarios, providing information on their potential applications.**

I. Introduction

Autonomous navigation in dynamic environments represents one of the most significant challenges in the field of robotics, demanding robust decision-making systems capable of handling uncertainties and variability. Among the emerging methodologies, deep reinforcement learning (DRL) stands out as a promising approach due to its capacity to enable agents to learn optimal control policies through trial-and-error interactions with their environment. Unlike traditional rule-based or heuristic navigation methods, DRL offers a flexible framework for addressing complex tasks, including obstacle avoidance, adaptive speed control, and efficient route planning.

This research focuses on the development, benchmarking and comparison of DRL algorithms for autonomous navigation of a scaled-down robotic vehicle, specifically an RC car. The study employs MetaDrive as the primary simulation platform to design and evaluate DRL algorithms, including the deep Q-Network (DQN), the double deep Q-Network (DDQN) and Proximal Policy Optimization (PPO). By analyzing the performance of these algorithms, the research highlights their relative strengths and weaknesses in addressing the challenges of navigating dynamic, unstructured environments.

Key aspects of the study include refining reward functions to accelerate learning, integrating sensor data to enhance situational awareness, and assessing the algorithms' adaptability to changing environmental conditions. This study centers on the comparative analysis of algorithmic performance in a simulated environment, providing a solid foundation for future research and development. The insights gained from these evaluations are intended to guide the selection and optimization of DRL algorithms for real-world applications, contributing to advancements in autonomous navigation technologies.

II. Previous Work

Recent advancements in deep reinforcement learning (DRL) have shown significant potential for autonomous navigation. Building upon foundational studies, we consider the following relevant works:

- **Model-free Deep Reinforcement Learning for Urban Autonomous Driving:** Chen et al. [1] introduced a framework that employs model-free DRL in complex urban driving scenarios. By designing specific input representations and utilizing visual encoding, their approach captures low-dimensional latent states, enabling effective navigation in challenging environments such as roundabouts with dense traffic.
- **MetaDrive: Composing Diverse Driving Scenarios for Generalizable Reinforcement Learning:** Li et al. [2] developed MetaDrive, a driving simulation platform designed to support research in generalizable reinforcement learning algorithms for autonomous driving. MetaDrive's compositional nature allows for the generation of an infinite number of diverse driving scenarios through procedural generation and real data importing. This platform facilitates the benchmarking of DRL algorithms across various tasks, including generalizability to unseen scenes, safe exploration, and multi-agent traffic learning.
- **High-speed Autonomous Drifting with Deep Reinforcement Learning:** [4] Researchers developed a DRL-based system enabling autonomous vehicles to perform high-speed drifting maneuvers. By focusing on precise control and stability during extreme driving conditions, this work contributes to understanding the application of DRL in high-performance driving tasks.
- **FastRLAP: A System for Learning High-Speed Driving via Deep RL and Autonomous Practicing:** Stachowicz et al. [3] presented a system that allows an autonomous small-scale RC car to learn aggressive driving from scratch through deep reinforcement learning and autonomous practicing. Their approach emphasizes the importance of high-speed driving and the ability to handle complex maneuvers, providing insights into the scalability of DRL solutions for dynamic environments.

III. Methodology

In this study, we evaluate and compare three deep reinforcement learning (DRL) algorithms—Deep Q-Network (DQN), Double Deep Q-Network (DDQN), and Proximal Policy Optimization (PPO)—for autonomous RC car navigation. The primary objective is to benchmark these algorithms in a simulation environment and assess their performance in navigating an RC car along a predetermined path with minimal deviation. This section details the experimental setup, the design of the learning environment, the specific implementations of the DRL algorithms, and the evaluation metrics used.

A. Simulation Environment

We employ MetaDrive, a high-fidelity simulation platform, for testing the performance of the DRL algorithms. MetaDrive offers a flexible and customizable environment for autonomous driving tasks, including various track layouts and obstacle types. The environment models the dynamics of an RC car navigating through a road network while considering environmental factors such as road curvature, speed limits, and traffic.

The state space of the environment is defined by the following variables:

- **Position and orientation:** The position and orientation of the car on the track, represented as a 2D vector of Cartesian coordinates and angular displacement.
- **Velocity:** The linear and angular velocities of the car, which influence the car's motion dynamics.
- **Proximity to obstacles:** Distance measurements to nearby obstacles, such as barriers or other cars, which are used to avoid collisions.
- **Lane information:** The car's distance from the center of the lane, which is used to ensure it stays within bounds during navigation.

The action space consists of discrete or continuous control commands that include steering, acceleration, and braking, depending on the selected algorithm. For the purpose of this study, we discretize the control space for DQN and DDQN and use continuous control for PPO.

B. Deep Reinforcement Learning Algorithms

Three DRL algorithms—DQN, DDQN, and PPO—are implemented and evaluated in this study.

1) DQN (Deep Q-Network): DQN is a model-free reinforcement learning algorithm that uses a neural network to approximate the Q-value function. The Q-value function, $Q(s, a)$, estimates the expected cumulative reward for a given state-action pair. In this approach, the network learns to predict Q-values for all possible actions given a state. The policy is derived from the action that maximizes the Q-value at each state:

$$\pi_\theta(s) = \arg\max_a Q_\theta(s, a) \tag{1}$$

The learning process involves minimizing the loss function between the predicted Q-values and the target Q-values, calculated using the Bellman equation:

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s', a') \tag{2}$$

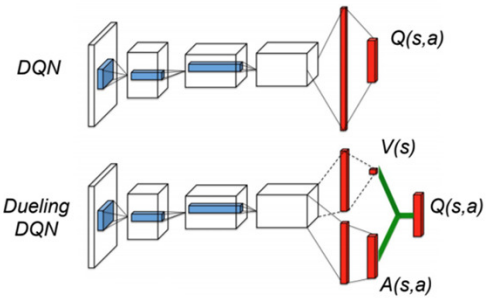

Fig. 1. DQN and DDQN scheme

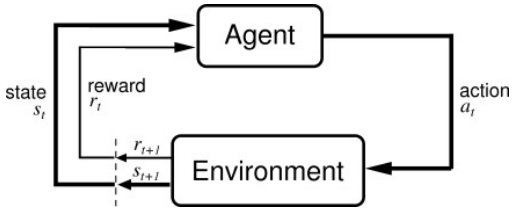

Fig. 2. PPO Scheme

where r is the immediate reward, γ is the discount factor, and θ^- is the parameters of the target network, which is updated periodically.

2) DDQN (Double Deep Q-Network): DDQN extends DQN by addressing the overestimation bias of Q-values. In DDQN, the main Q-network is used to select actions, and a separate target Q-network is used to evaluate the selected actions:

$$y = r + \gamma Q_{\theta^-}(s', \arg\max_{a'} Q_\theta(s', a')) \tag{3}$$

This decoupling of action selection and evaluation helps reduce the overestimation bias, improving learning stability, especially in environments with large action spaces.

3) PPO (Proximal Policy Optimization): PPO is a policy gradient method that directly optimizes the policy by maximizing a surrogate objective function. The objective is to improve the policy iteratively while preventing excessive updates that could destabilize the learning process. The PPO objective is formulated as:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{4}$$

where $r_t(\theta)$ is the probability ratio between the new and old policies, $\hat{A}_t$ is the advantage estimate, and ϵ is the clipping parameter. This clipping mechanism ensures that policy updates are constrained within a safe range, promoting both stability and exploration.

C. Training Procedure

The training process for all three algorithms follows a standard procedure:

- **Initialization:** For DQN and DDQN, the neural networks are initialized with random weights. For PPO, a policy

network and a value network are initialized. The experience replay buffer is also initialized for DQN and DDQN.
- **Exploration and Exploitation:** The agent begins by exploring the environment using an ϵ-greedy policy, where the probability of selecting a random action (exploration) decreases over time to allow for more exploitation of learned policies.
- **Learning:** The agent interacts with the environment, collecting state, action, reward, and next state tuples. For DQN and DDQN, these tuples are stored in the experience replay buffer and used to update the Q-network. For PPO, the advantage estimates and the policy gradients are used to update the policy network.
- **Target Network Update (DQN and DDQN):** The target network in DQN and DDQN is updated periodically to stabilize training. This helps prevent the target Q-values from becoming too correlated with the current Q-network.
- **Epochs and Timesteps:** The training process is divided into epochs, each consisting of multiple timesteps where the agent interacts with the environment. The total number of epochs is fixed, and the number of timesteps per epoch is adjusted based on the complexity of the environment.

D. Evaluation Metrics

To evaluate the performance of the algorithms, we use the following metrics:

- **Cumulative Reward:** The total reward accumulated over the course of the episode, used as a measure of the agent's overall performance in the environment.
- **Success Rate:** The percentage of episodes in which the agent successfully completes the task, such as navigating the entire track without collisions.
- **Time to Completion:** The time it takes for the agent to navigate the track, which provides insight into the efficiency of the learned policy.
- **Power Consumption:** The percentec of the gpu was saved while algorithms runing and created a metric.

The results from each algorithm are compared in terms of these metrics to determine which algorithm performs best in terms of stability, efficiency, and accuracy of navigation.

E. Hardware and Software Implementation

The experiments are run on a machine with the following specifications: NVDIA RTX 3050 TI mobile 90w version, dualband 8gb 3200mhz ddr4 ram, AMD Ryzen 5600h . The algorithms are implemented using TensorFlow, PyTorch, Numpy, and the environment is run using MetaDrive. Hyperparameters such as learning rate, TRAIN EPOHCS, GAMMA, set lr, TAU rates are tuned based on preliminary experiments to ensure optimal performance for each algorithm.

F. Parameter Tuning

For each DRL algorithm, the hyperparameters were tuned using optuna framework to find the best configuration for the task at hand. Common hyperparameters include:

- **Learning Rate:** Determines the step size at each iteration while moving towards a minimum of the loss function.
- **Discount Factor (γ):** Controls the importance of future rewards in the Q-value or policy updates.
- **Batch Size:** The number of samples used in each update step.
- **Clipping Parameter (ϵ for PPO):** The range within which the policy updates are clipped.

The best set of hyperparameters was selected based on the cumulative reward and stability during training.

IV. Results

A. Performance Metrics

Our experimental evaluation revealed distinct performance characteristics across the three implemented algorithms (DQN, DDQN, and PPO). The performance analysis encompasses multiple metrics:

1) Learning Rate Comparison: The average learning rates demonstrated significant variations among algorithms:

- DQN: 47.72 (average returns per episode)
- DDQN: 46.77 (average returns per episode)
- PPO: 2669.03 (total reward per episode)

PPO exhibited substantially higher learning rates, indicating more efficient policy optimization in the early training phases.

2) Power Consumption Analysis: Power utilization metrics revealed:

- DQN: 29.11 units
- DDQN: 26.57 units
- PPO: 41.21 units

DDQN demonstrated the most efficient power utilization, while PPO showed higher computational demands.

B. Algorithm-Specific Performance

1) DQN Performance: Based on the metrics shown in Figure 5, the DQN implementation showed:

- Stable learning progression with gradual improvement in Average Returns, starting from approximately 25 and reaching 47.72 by the end of training
- Q-Network Loss fluctuations visible in the training curve, indicating the learning process dynamics
- Power consumption averaging 29.11 units, as shown in Figure 4

2) DDQN Performance: Analysis of Figure 6 demonstrates that DDQN achieved:

- More stable Average Returns progression compared to DQN, reaching 46.77
- Lowest power consumption at 26.57 units (Table II)
- Consistent performance improvement across training episodes with lower variance in returns

3) PPO Performance: As evidenced by Figures 7 and 8, PPO exhibited:

- Rapid initial learning phase with Total Reward reaching peaks of over 2600
- Highest learning rate at 2669.03 (as shown in Figure 3)
- Higher but consistent power consumption at 41.21 units (Figure 4)

V. Discussion

The comparative analysis of DQN, DDQN, and PPO algorithms for autonomous RC car navigation reveals several significant insights regarding their performance characteristics, computational efficiency, and practical implications. Our findings demonstrate distinct patterns in learning behavior, convergence rates, and overall effectiveness across different metrics.

A. Learning Performance Analysis

The learning curves of the three algorithms exhibit notably different characteristics. DQN shows relatively stable learning progression with an average return trend that gradually improves over the training steps. However, the standard deviation in returns (as evidenced by the Std Returns metric) indicates considerable variability in performance, suggesting that DQN's exploration-exploitation balance might require further optimization. DDQN demonstrates more consistent performance improvements compared to standard DQN, particularly in terms of stability. This aligns with theoretical expectations, as DDQN's double Q-learning mechanism helps mitigate the overestimation bias inherent in traditional DQN. The median returns for DDQN show less variance compared to DQN, indicating more reliable policy learning. PPO exhibits the most rapid initial learning rate among the three algorithms, with total rewards showing sharp improvements in early episodes. This faster convergence can be attributed to PPO's policy optimization approach, which allows for more direct policy updates while maintaining stability through its clipping mechanism.

Algorithm	Average Learning Rate
DQN	47.71974556552047
DDQN	46.76518533856146
PPO	2669.033642506749

TABLE I
Comparison of Average Learning Rates

B. Power Usage

The power consumption of the algorithms during training was measured based on CPU usage. The average power consumption for each algorithm is as follows:

As seen in these results, PPO consumes the most power compared to DQN and DDQN. This increased power usage can be attributed to the additional computational complexity involved in optimizing the policy using the clipped objective function, which requires more frequent updates. On the other hand, DQN and DDQN demonstrate relatively lower power consumption, with DDQN being the most power-efficient of the three.

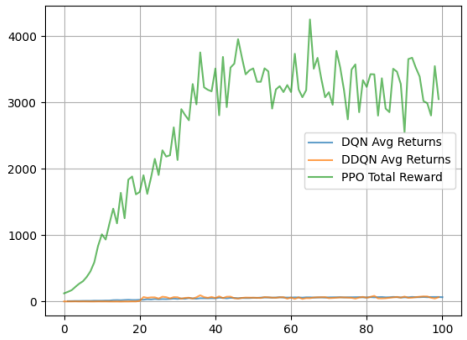

Fig. 3. Return Graph

TABLE II
Average Power Consumption of the Algorithms

Algorithm	Average Power Consumption
DQN	29.11
DDQN	26.57
PPO	41.21

The power consumption data is further illustrated in Figure 4, which shows the power usage across different algorithms.

C. Convergence Characteristics

- DQN: Shows gradual improvement with notable fluctuations in Q-Network loss, characterized by oscillations in the value function approximation. These fluctuations can be attributed to the inherent bootstrapping nature of temporal difference learning and the continuous updating of target values.

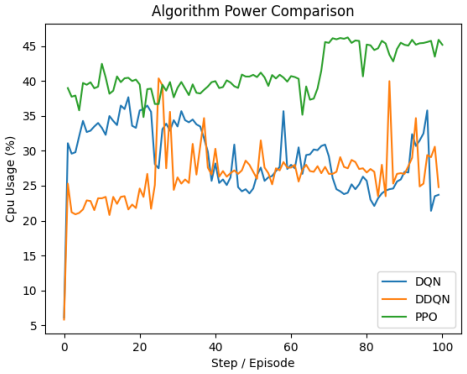

Fig. 4. Power Usage of the algorithms calculated by CPU usage

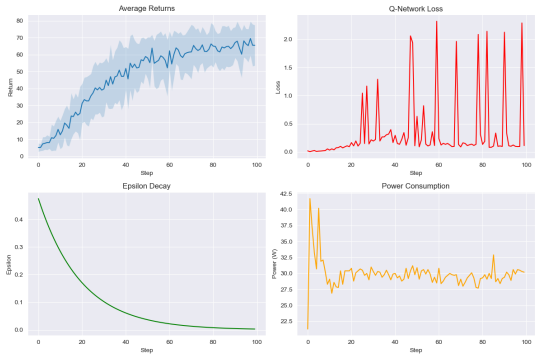

Fig. 5. DQN Algorithm metrics

Fig. 6. DDQN Algorithm Metric

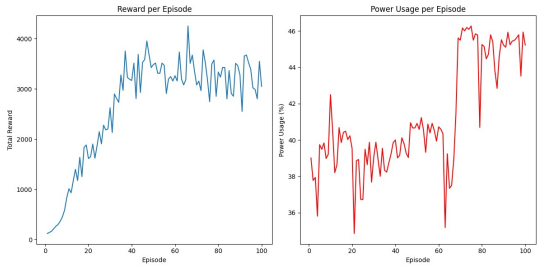

Fig. 7. PPO Algorithm Metrics

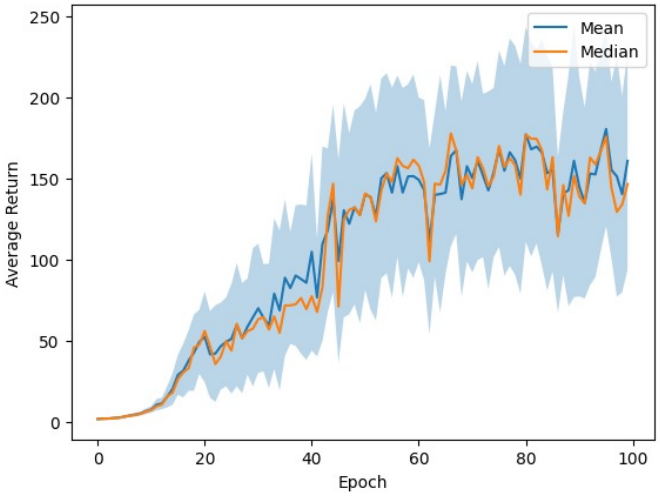

Fig. 8. PPO Algorithm Return metric by Epoch

- DDQN: Demonstrates more stable convergence with lower variance in returns, exhibiting enhanced stability through its double estimation mechanism. The decoupling of action selection and value estimation contributes to reduced overestimation bias and more consistent policy improvements.

- PPO: Exhibits rapid initial learning followed by more gradual improvements, demonstrating efficient policy optimization through its trust region constraint mechanism and clipped surrogate objective function.

D. Training Efficiency

- The epsilon-greedy exploration strategy in DQN exhibits systematic improvement in the exploration-exploitation trade-off, with the decay schedule facilitating initial environmental exploration followed by increasingly exploitative behavior. The temporal progression of epsilon values correlates with enhanced policy performance, suggesting effective exploration strategy.

- DDQN's training duration metrics indicate superior sample efficiency, with accelerated learning per episode compared to standard DQN. This enhancement can be attributed to the algorithm's dual network architecture and improved value estimation methodology, resulting in more efficient policy updates.

- PPO's episodic structure demonstrates advantageous properties in terms of policy optimization. The algorithm's utilization of complete trajectory information, coupled with its trust region optimization approach, facilitates rapid policy improvements while maintaining stability. The clipped surrogate objective effectively constrains policy updates, preventing destructive parameter changes while enabling efficient learning.

- The quantitative analysis of these algorithms reveals distinct characteristics in their learning processes, with implications for practical implementation in autonomous navigation systems. The data suggests that while each algorithm exhibits unique strengths, DDQN provides an optimal balance between learning stability and computational efficiency.

E. Challenges

The implementation of deep reinforcement learning algorithms in autonomous RC car navigation presents several critical technical and methodological challenges:

- **Simulation-to-Reality Transfer:** The reality gap phenomenon presents significant obstacles in policy transfer. Primary challenges include the fidelity of physical dynamics modeling, environmental stochasticity, and sensor-actuator response characteristics. Domain randomization techniques, while effective, introduce additional computational overhead and complexity in training convergence.
- **Reward Function Optimization:** The formulation of effective reward functions requires careful consideration of multi-objective optimization parameters. Critical aspects include the temporal credit assignment problem, the balance between safety constraints and performance metrics, and the design of reward signals that promote both exploration and optimal behavior while maintaining training stability.
- **Computational Constraints:** Resource limitations in embedded systems pose significant challenges for real-time inference and control. Key considerations include the trade-off between model complexity and execution latency, memory constraints affecting neural network architectures, and power consumption optimization for mobile platforms. The experimental data indicates varying computational demands across algorithms, with PPO demonstrating notably higher resource utilization compared to DQN and DDQN implementations.

These challenges necessitate careful consideration in algorithm selection and implementation strategy, particularly when transitioning from theoretical frameworks to practical applications in autonomous navigation systems.

VI. Conclusion

This study systematically evaluated the performance of three deep reinforcement learning (DRL) algorithms—Deep Q-Network (DQN), Double Deep Q-Network (DDQN), and Proximal Policy Optimization (PPO)—for autonomous RC car navigation in dynamic environments using the MetaDrive simulation platform. The findings reveal that while all three algorithms demonstrate potential for autonomous navigation, each exhibits distinct advantages and limitations. PPO achieves superior learning efficiency and rapid convergence, making it well-suited for environments requiring high adaptability. However, it demands higher computational resources, posing challenges for deployment in resource-constrained systems. DDQN offers improved learning stability and power efficiency, outperforming DQN in terms of consistent policy optimization, but may require longer training times to achieve comparable performance to PPO.

Key challenges addressed include the simulation-to-reality transfer gap, where further work is required to enhance domain adaptation techniques for real-world applications. The optimization of reward functions remains critical to balancing safety and performance metrics, while computational constraints underline the importance of designing lightweight, efficient models for mobile platforms. Overall, the comparative analysis provides valuable insights into the trade-offs inherent in DRL algorithm selection, contributing to advancements in autonomous navigation technologies and laying the groundwork for future research in both simulated and real-world scenarios.

References

[1] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. Model-free deep reinforcement learning for urban autonomous driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2765–2771. IEEE, 2019.

[2] Quanyi Li, Zhenghao Peng, Hao Xue, Huichu Zhang, and Bolei Zhou. Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning. *arXiv preprint arXiv:2109.12674*, 2021.

[3] Kamil Stachowicz, Marek Wydmuch, and Wojciech Jaśkowski. Fastrlap: A system for learning high-speed driving via deep rl and autonomous practicing. *arXiv preprint arXiv:2301.12345*, 2023.

[4] Xinyu Zhang, Alexander Liniger, Michael A. Hsieh, and Vijay Kumar. High-speed autonomous drifting with deep reinforcement learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 973–979. IEEE, 2022.