Evaluating Robustness of Deep Reinforcement Learning for Autonomous Driving

Effects of Domain Randomization on Training and Robustness

Ege Bayram[1]

Supervisor(s): Matthijs Spaan[1]**, Moritz Zanger**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 24, 2023

Name of the student: Ege Bayram
Final project course: CSE3000 Research Project
Thesis committee: Matthijs Spaan, Moritz Zanger, Elena Congeduti

Abstract

Deep reinforcement learning has been a topic of research in recent years and has been expanding into the domain of autonomous driving. As autonomous driving is likely to involve people, such as daily commuters, it is necessary to ensure the machine will perform well enough in real-life environments not to put anyone at risk. There exist possible approaches to make the transition from a simulation to real life easier, such as domain randomization. This paper uses OpenAI's CarRacing-v2 environment and the CARLA simulator [4] to investigate the effect of domain randomization on training efficiency and robustness for a Deep Q-Network algorithm for autonomous driving. The results show a decrease in training efficiency and higher variance during training for both environments. CARLA also indicates an overestimation during training. As for robustness testing, while visual domain randomization in CarRacing-v2 does not suggest a significant influence on robustness, the dynamic domain randomization in CARLA offers a positive influence toward robustness at the expense of some reward.

1 Introduction

The interest in neural networks in reinforcement learning has been on the rise with Gerald Tesauro's TD-Gammon program to play backgammon in 1992 [13] and the model introduced by Mnih et al. to play Atari in 2013 [9]. After witnessing such success in games, the interest in deep reinforcement learning (deep RL) expanded to other domains, such as robotics, autonomous driving, and autonomous navigation. With the field of autonomous driving increasing in fame with the latest achievements of companies such as Benz, Tesla, and Google competing for fully autonomous commercial vehicles, various autonomous driving system applications were made. With the previous success of reinforcement learning in handling complex raw input and its ability to learn and improve a policy without a manual design, the application of deep RL expanded to the autonomous driving domain as well [2]. The expansion brought new simulation environments and algorithm libraries helpful to use for researchers. This research also utilizes one such simulation, CARLA, a highly realistic and customizable simulation useful for autonomous driving research [4].

The expansion and new research of deep RL in autonomous driving also brought forth various drawbacks, namely the reality gap between the simulation and the real-world environment, and the lack of robustness of autonomous driving. The reality gap refers to the collection of differences between the simulation environment and the real world, as generally, simulations are not as complex or random as real-world environments can get. Some differences in CARLA simulation and real-world traffic environments may include various traffic signs, differences in the behavior of other drivers, and sudden traffic accidents or congestion. The robustness of an algorithm, on the other hand, refers to the stability of the performance under different environments, meaning whether the algorithm provides consistent results in various environments or performs better in some environments and worse in others. Since autonomous driving cars are likely to involve people, this gives rise to a critical issue where any mistake can be lethal, thus it is desired that the algorithm performs consistently in every environment than the latter.

This is why it is necessary to investigate the approaches to bridge the existing gap to create an algorithm stable enough for real-world use. The approach we focus on is domain randomization, one that introduces variability to simulations by randomizing the parameters of the environment during training in hopes that the real-world environment will be perceived as just another randomization for the algorithm [15]. By evaluating the training data of a model trained with domain randomization, we can check the training performance and efficiency of domain randomization compared to a model trained with just one set of parameters. Training performance in this context includes the comparison between the episodic returns of the agent and estimated rewards, and the efficiency is measured by how fast the agent starts learning and how fast the agent's episodic returns start converging. Subsequently, by testing the performance of the trained model on various environments, we can gain insight into domain randomization's influence on robustness compared to a base model.

In essence, the research question derived is "How do domain randomizations influence training and the robustness of final policies under various testing conditions?". Section 2 details the necessary background information about the Markov decision process, deep RL model, domain randomization, and related work. We then identify a research methodology, detailed in Section 3, to answer our question. Section 4 introduces the experimental results and provides an evaluation of the findings. The next section, section 5, discusses the possible ethical implications of the experiment and its findings, as well as the reproducibility of the research by others. Section 6 discusses the limitations encountered during the study. The paper ends with section 7, which contains conclusions and possible future work for improvement.

2 Background

This section introduces the necessary background information to understand the components of this work. We first introduce the Markov decision process, a decision-making model widely used in reinforcement learning, in subsection 2.1. Afterward, we delve into the deep Q-network (DQN) model used in the experiments in subsection 2.2. Then we discuss domain randomization in section 2.2, followed up by previous work in the literature related to our research in subsection 2.3.

2.1 Markov Decision Process (MDP)

Markov decision process is a framework introduced by Ronald Howard for optimal decision-making. It is denoted as a 4-tuple (S, A, P, R) where S is the state space, a collection of all possible configurations in the environment. A is the action space, also denoted as $A(s)$ to show all the actions

$$Q(s,a) \leftarrow Q(s,a) + \alpha \times [R + \gamma \times maxQ(s^{'}, a^{'}) - Q(s,a)]$$

Figure 1: Equation for updating Q-values according to Bellman optimality equation

that can be taken within a state. P is the probability distribution, or transition function, also shown as $P(s,a,s')$ for the probability of ending up in a new state s' if one takes an action a in state s. Lastly, R is the reward notation, also known as the reward function $R(a,s)$ to denote the expected reward if action a is taken in state s. MDP also introduces a policy function π, inputs a state s and outputs which action a should be taken for maximum reward. Maximizing the policy function for all states in the state space leads to optimal policy π^*, revealing the optimization objective of MDPs. The probability distribution P and rewards R are unknown for MDPs in reinforcement learning and the agents try to maximize a cumulative reward starting from an initial state s_0, thus aiming to learn an optimal policy from s_0, $\pi^*(s_0)$.

A partially observable MDP (POMDP) occurs when the current state s is unknown, and the model is unable to calculate $\pi(s)$, which is the case in our environments' outputs of single frames because the agent cannot gauge its direction of movement or current velocity. This can be mitigated by utilizing the last few frames, also known as frame stacking, to understand the current state of the agent. By utilizing frame stacking, the problem turns into an MDP again.

2.2 Deep Q-Networks (DQN)

Introduced by Mnih et al. in 2013, the deep Q-network algorithm utilizes a neural network trained with a variation of the Q-learning reinforcement learning algorithm to estimate the optimal action to take in a given state, also known as the action-state function or the Q-function [9]. The term Q-network comes from the neural network approximator used in the algorithm to estimate the action-state function. The following subsections will explore the key components of the DQN algorithm.

Q-Learning and Q-Network

Q-Learning is an off-policy temporal-difference learning algorithm to find an optimal target policy [6]. The algorithm inputs the state and action spaces in the environment and generates a Q-table, also known as a Q-matrix, that maps each state-action pair (s,a) to a Q-value. The algorithm then updates these Q-values iteratively following the Bellman optimality equation, a variation for updating Q-values is shown in Figure 1 [6]. This equation describes the Q-value for a state-action pair during time t Q(s_t, a_t) can be defined by a weighted sum of the current Q-value and the future value, where α is the learning rate, r_t is the reward received from taking action a in state s during time t, and γ is the discount factor for future rewards. Over time, these values can converge to the optimal value Q^* [12].

The main difference between Q-Learning described above and the Q-Network Mnih et al. introduces within DQN is the difference within the Q-function. While Q-Learning takes the state and action spaces in the environment and maps a Q-value to all possible pairs, Q-Network takes the current state and returns a Q-value for all actions using the neural network. The Q-Network itself is trained by minimizing the loss function, shown in Figure 2, updated at each iteration i.

$$L_i(\theta_i) = E_{s,a\sim p(.)}[(y_i - Q(s,a;\theta_i))^2]$$

Figure 2: Loss function for DQN introduced by [9]

Epsilon-Greedy Policy

DQN utilizes an ϵ-greedy policy, meaning that at each time t, the agent selects one of two possible actions: exploration, where the agent takes a random action, possibly with an unknown outcome to gain information; or it can use its previously gathered knowledge and act to gain rewards, also known as exploitation [7]. This concept is also known as the exploitation-exploration trade-off since if the agent keeps choosing exploitation it is likely to only reach a local maximum for its rewards, but if it continuously chooses exploration the agent will only choose random actions and will likely not reach an optimal reward [12]. Thus, the balance is important for the agent to both explore and utilize its knowledge. The agent determines which action to take based on its ϵ value, the probability of choosing exploration is ϵ whereas the probability of choosing exploitation is $1-\epsilon$. Initially, the agent prioritizes exploration as its knowledge is limited, and this gradually shifts towards prioritizing exploitation.

Experience Replay

The trial-and-error approach may be inefficient, as the network can forget its knowledge over time with each adjustment within the network until it goes through that transition pattern again, also known as the re-learning problem. To overcome this, an experience replay mechanism can be utilized, where the experiences, a tuple $(s,a,s^{'},r)$ denoting an action a taken in state s causing the agent to reach a new state $s^{'}$ and reinforcement r, are stored in memory during training. These experiences are then randomly sampled to re-experience to improve the training distribution and mitigate the re-learning problem [7], [9].

2.3 Domain Randomization (DR)

When the domain the agent is trained on, also known as the source domain, and the domain the agent is tested on, also known as the target domain, differ from each other, there exists a gap between these two domains that may lead to the agent underperforming in the target domain. Domain randomization is an approach used to close the gap between these domains by generalizing a model trained in a source domain such that it also performs at a desired level in the target domain. By controlling a set of parameters in the source domain and generating other domains by randomizing these parameters with a specified configuration, we train the policy parameter to generalize across various configurations [15]. A conceptual illustration for this can be seen in Figure 3, where the calibrated sim is the source domain and reality is the target domain. This figure depicts what is known as the Sim-to-Real gap, which is the gap between the domains when the source is a simulation environment and the target is the real world.

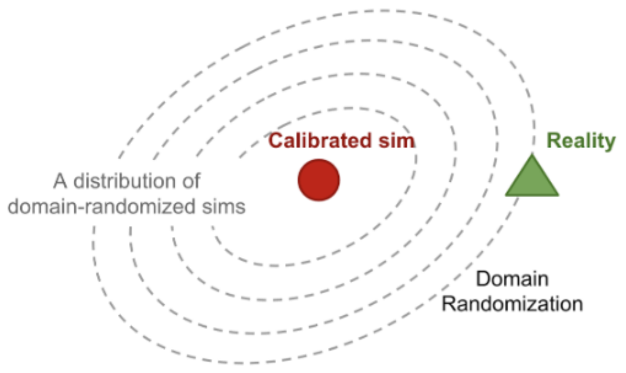

Figure 3: Conceptual illustration for domain randomization [15]

The set of controlled parameters can range from visual parameters, such as the work by Tobin et al., where they randomized the textures of objects, the location of the camera, or random observation noise added to images [14], to more dynamic parameters that will affect the agent's policies to some extent, for instance, Peng et al.'s experiment where they randomized the mass, friction, and damping values [10].

2.4 Related Work

Previous work done by Peng et al., Sadeghi and Levine, and Tobin et al. all showcase a successful generalization by domain randomization and increased robustness of the model, [10], [11], [14]. These papers indicate that both visual and dynamic randomization can increase the robustness of the model. However, Mehta et al. have also shown that domain randomization may also lead to worse performance than the baseline algorithm [8]. We will investigate the effects of domain randomization with both visual and dynamic parameters on training efficiency and robustness for autonomous driving since the previously mentioned papers employ these methods for different tasks. If successful, our paper could lead to future research in Sim-to-Real transfer for autonomous driving.

3 Methodology

It is necessary to come up with a systematic methodology to answer the research question "How do domain randomizations influence training and the robustness of final policies under various testing conditions?". This section will first discuss the environments and the setup, and then explain the concrete methodology in stages according to their main focus.

As we are investigating the visual and dynamic parameters separately, the research is also separated into two different stages. The first stage, consisting of visual parameters will be trained and tested in the CarRacing-v2[1] environment in OpenAI's Gym framework, and the dynamic parameters will be trained and tested using the CARLA simulator [4]. The CarRacing-v2 environment is a top-down racing environment with a single car and a Formula 1 race track generated randomly at each episode. As this environment is simpler than CARLA in terms of computation and variables, the data generated from this part will also be utilized as an early insight into the computationally expensive CARLA environment, which is more complex and similar to real-world traffic. Through CleanRL [5], a DQN implementation that was easy to use and alter with research-friendly features was available from the start. Some alterations were made to the algorithm to adapt it to the CARLA environment. We also utilized the gym-carla wrapper [1], a wrapper that adapts the CARLA environment to that of an OpenAI Gym third-party environment, to be able to apply the DQN implementation inside the CARLA simulator. The wrapper also introduces a parameter list for the environment, which makes parameter randomization easier. A full list of customizable parameters is provided in Appendix A.1.

[1]https://www.gymlibrary.dev/environments/box2d/car_racing

Stage One: Visual domain randomization in CarRacing-v2

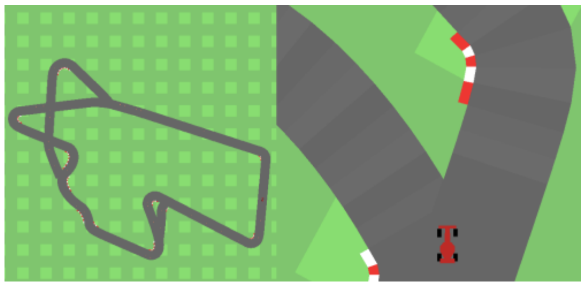

Figure 4: Observation view of CarRacing-v2 environment

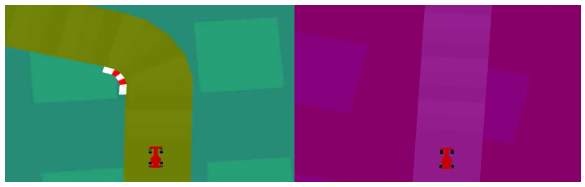

Figure 5: CarRacing-v2 environment with domain randomization examples

This stage makes up the first part of this study - investigating domain randomization with visual parameters and its effects on the training and robustness of the model using OpenAI's CarRacing-v2 environment. The steps involved in this stage are as follows:

- Get a working DQN algorithm for the CarRacing-v2 environment with the help of CleanRL.
- Train the algorithm for one million steps, once with the *domain_randomize* parameter set to True while making the environment, and once with False, which is the default value.
- Compare the training data in terms of overall episodic returns, how fast the variations start to learn, and how

fast they start to converge, as well as the variations' estimated Q-values.

- Run 10 evaluation episodes with the trained model for both variations and compare the results for robustness testing.

Stage Two: Dynamic domain randomization in CARLA

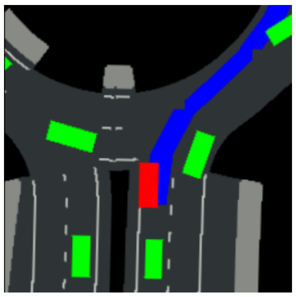

Figure 6: Birdeye view of CARLA simulator [1]

This stage makes up the second part of this study - investigating domain randomization with dynamic parameters and its effects on the training and robustness of the model using the CARLA simulator and the gym-carla wrapper. Since the gym-carla wrapper does not provide a randomize option by itself, we utilize the *domain-randomizer* [2] library provided by Mehta and Raparthy. Following the tutorial on the GitHub page allows one to configure their desired training randomizations to the extent of identifying a default value, a minimum multiplier, and a maximum multiplier where at each episode a random number between the two multiplier values changes the value of the parameter. The steps involved in this stage are as follows:

- Adapt the DQN algorithm into CARLA's environment.
- Identify training randomization parameters that will be different than the evaluation randomization. The training randomization parameters identified in this research are:
  - **Steering angle**, the value to control the angle of steering for when the agent decides to turn to a direction within a step
  - **Acceleration value**, the value to control the acceleration and deceleration of the vehicle within a step
  - **Out-of-lane threshold**, the threshold to detect whether the car is out of the lane to terminate the current episode, leading to possible subpar rewards
- Train the algorithm twice for five hundred thousand steps on CARLA. Once with domain randomization and once without.
- Compare the training data in terms of overall episodic returns, how fast the variations start to learn, and how fast they start to converge, as well as the variations' estimated Q-values.

[2]https://github.com/montrealrobotics/domain-randomizer

- Conduct a robustness test by running 10 evaluation episodes with differing evaluation randomization. The randomization for robustness testing defined in this research was to change the map. We used three maps provided by CARLA: Town03, Town04, and Town05.

4 Experiments

The experiments were designed and conducted with the methodology described in section 3. The experimental setup included using the DQN algorithm provided by CleanRL on two different environments with different complexities to train two different types of randomization parameters. The environments used were OpenAI's CarRacing-v2 and CARLA simulator version 0.9.13. The motivation behind using visual domain randomization (visual DR) in CarRacing-v2 and dynamic domain randomization (dynamic DR) in CARLA is the difference in complexity between the environments besides the limitation that their available parameters offered. As CarRacing-v2 is a simpler environment only consisting of a background, a race track, and a vehicle, visual randomization parameters could influence the agent more.

The following subsections provide results for visual domain randomization in CarRacing-v2 and dynamic domain randomization in CARLA.

4.1 Visual domain randomization in CarRacing-v2

The CarRacing-v2 environment already provides an argument to enable the domain randomization variant. By enabling this variation, the background and track color of the environment changes at each reset individually, meaning there are also cases where the colors are the same as can be seen in Figure 5. There is also the randomization factor where the generated race track is also random at every episode, which is enabled for both variations of the environment.

To compare the training efficiency of these models, we logged the shift in episodic lengths, episodic returns, and Q-values of the training data and plotted these for each timestep, visualized by Figures 7, 8, and 9. Firstly, the trends of Q-values and episodic returns show signs of converging, so it can be assumed that both models were able to train to some extent, although the oscillation for DR on episodic returns may indicate the agent could learn further if it had more time. The slight delay in Figure 8 for DR indicates that the model with domain randomization started learning later, which is expected since the agent is faced with multiple variations and has to train itself to adapt to these different circumstances. It is also observed that the smoothed episodic returns for DR are always lower than the base variation. This difference cannot be attributed to the episodic length, since even though each frame causes a -1 in reward, there is hardly any difference in episodic length for the algorithms. Therefore, it is safe to assume that the DR model underperforms compared to its counterpart in terms of rewards.

The raw data for episodic returns, indicated by the faded lines in the background, seem to have high variance because it oscillates throughout the training. Witnessing a similar trend in Q-values as well, this can be an implication that the agent

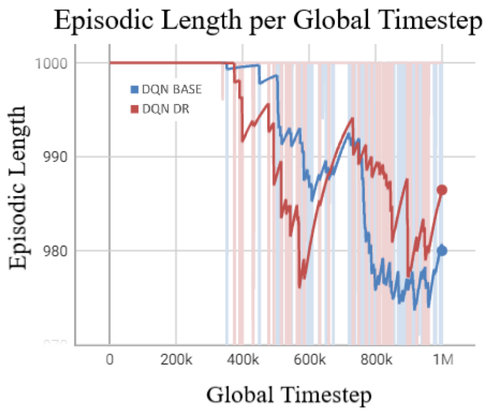

Figure 7: Episodic length over total timesteps for CarRacing-v2 models

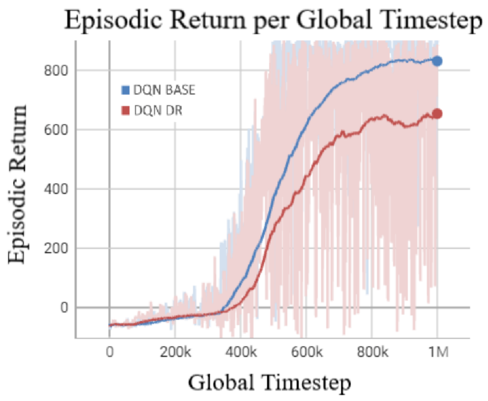

Figure 8: Episodic return over total timesteps for CarRacing-v2 models

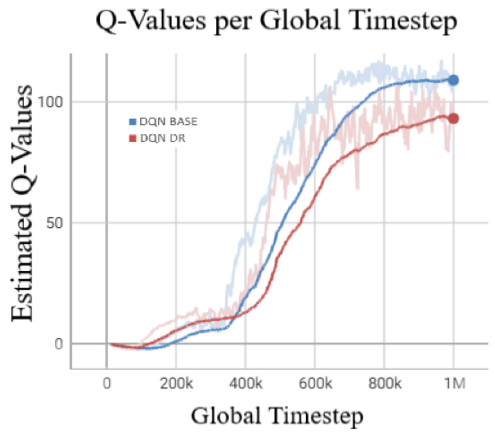

Figure 9: Q-Value over total timesteps for CarRacing-v2 models

has failed to generalize, thus causing oscillations in the data where it performs randomly based on generated colors. However, evaluating the data generated from the robustness testing is healthier to argue about the model's capabilities of generalizing.

Lastly, there is also a hint of overestimation of the DR model. When compared to the difference in episodic returns, we see that the difference in smoothed Q-values in Figure 9 is lower, as well as the DR variation surpassing the baseline until just before the four hundred thousand mark. Overall, we can suggest that the DR model underperforms compared to the model trained without randomization when it comes to training efficiency.

Episode	Base	DR
1	849.8461304	911.8059692
2	913.4052734	801.102417
3	862.8520508	815.0369873
4	900.8054199	868.5147705
5	805.5440674	845.0214233
6	896.2034302	465.7452393
7	840.352356	748.2772827
8	896.8123169	866.7800293
9	510.4972534	877.9499512
10	912.0064087	662.7657471
MEAN	838.8324707	786.2999817
STDEV	120.6565133	133.8916188

Table 1: Evaluation data for CarRacing-v2 models

For robustness testing, we individually ran both models for ten episodes after training. Domain randomization was disabled for the DR variation, and since the track maps are generated randomly at each episode, the agents were most likely evaluated on tracks they had no prior knowledge of. The collected data for both models can be observed in Figure 10 and Table 1. Overall, it can be argued that the generated episodic returns are similar, the only notable differences are in episodes 6 and 9, and a slightly smaller difference in episode 10. Table 1 indicates that the mean episodic return for DR is lower than the baseline while the standard deviation is higher. Judging by these ten episodes alone, it can be argued that due to the higher standard deviation, the model also underperforms in terms of stability and generalization when compared to the baseline algorithm, which aligns with the findings of Mehta et al. [8]. However, we can still reason that domain randomization increases stability and provides generalization to some extent when considering the standard deviation from the training data is heavily reduced, and the difference in episodic return with the baseline algorithm is also narrower which supports Tobin et al.'s findings [14].

4.2 Dynamic domain randomization in CARLA

CARLA simulator provides a variety of parameters for the domain to customize. In this experiment, we chose to randomize three variables: the acceleration value, the steering angle, and the out-of-lane threshold. These values are chosen to influence the agent's policies more directly during training. While the acceleration value and the steering angle directly

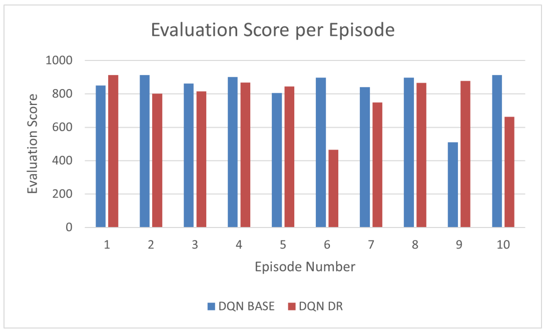

Figure 10: Evaluation scores over ten episodes for CarRacing-v2 models

influence the action the agent takes, the out-of-lane threshold can force the episode to terminate prematurely, resulting in subpar rewards, and influence the vehicle to keep away from the edges of the lane. The full list of CARLA parameters and the values we used for randomization are given in Appendix A.

Similar to subsection 4.1, we logged and plotted the change in episodic lengths, episodic returns, and Q-values of the training data over timesteps, visualized by Figures 11, 12, and 13. Judging by the episodic length and episodic return graphs, we can argue that the DR algorithm is still in the process of learning as the episodic returns do not show a sign of converging and the episodic length seems to be in an upward trend towards the end. Compared to DN, the baseline algorithm seems to be converging around the two hundred return value as the episodic length has converged and the episodic return seems to be starting to converge after it stopped its rise.

Another notable observation of the DR model is that the episodic return trends downwards twice. This could be due to randomization influencing the agent's policy. However, the Q-value of the algorithm is on the rise throughout training, thus it is hard to say that this caused the agent to change its policies drastically. Additionally, contrary to the comparison of episodic returns, DR outperforms the baseline algorithm when it comes to Q-values. Although it is trending downward at the end, the Q-value for DR is higher than the base model for almost three hundred thousand timesteps. This is an indication of the overestimation of the DR model, as the agent is estimating higher reward compared to the base model, but underperforming in terms of episodic returns for almost the whole duration of three hundred thousand timesteps. Overall, we can argue that similar to the CarRacing-v2 environment, the DR model underperforms compared to the base algorithm, except for a higher possible overestimation and a large oscillation in episodic return.

To test both models' robustness, we utilized three different maps provided by CARLA. Although the first of these maps, Town 03, is used for training, the agents do not possess any knowledge about the other two maps. We ran each model for ten episodes on each map and plotted the mean and standard deviation graphs visualized in Figures 14 and 15. We can observe that in general, the DR model performed worse than the baseline in terms of mean evaluation score.

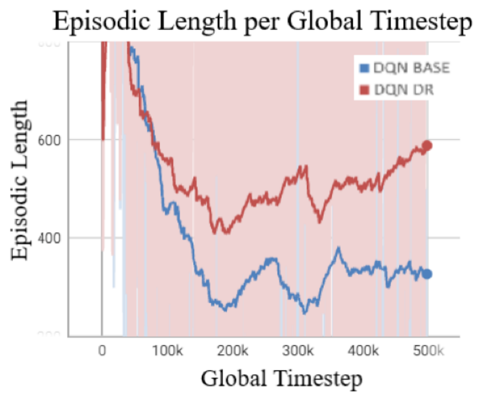

Figure 11: Episodic length over total timesteps for CARLA models

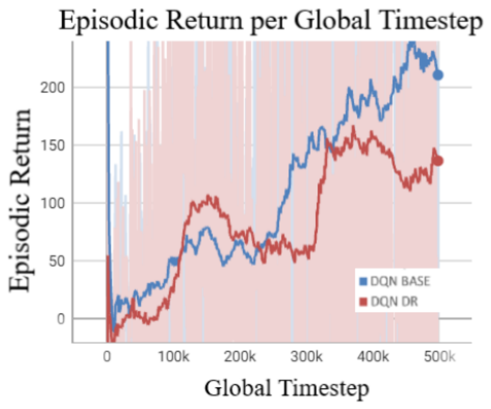

Figure 12: Episodic return over total timesteps for CARLA models

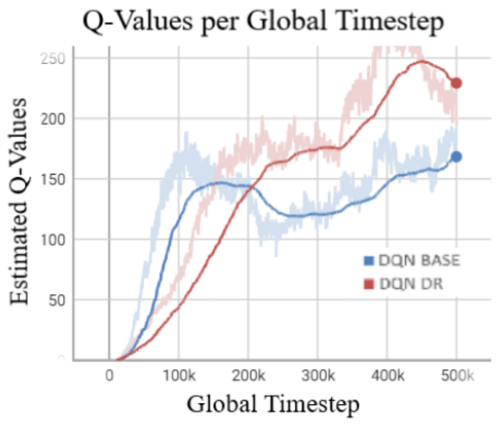

Figure 13: Q-Value over total timesteps for CARLA models

The performance in Town04 is especially low, with the base algorithm peaking and the DR model plunging. The difference between Town04 and the other towns is that it involves outskirt roads with junctions and curves. It might be possible that the curvy ring road within the environment resulted in a decrease in performance due to the change in acceleration values and steering angles. Even after considering the negative bias from the training as the DR agent was not able to converge and did not find the opportunity to optimize its Q-Network, we can argue the agent was not able to generalize for curvy roads and narrow junctions. Figure 15, on the other hand, shows that the DR algorithm had a notably less standard deviation of evaluation scores. This can indicate that the DR model is more consistent and stable in different environments, thus domain randomization positively influenced robustness. Overall, we can assume domain randomization positively influences the stability and consistency of the algorithm at the cost of performance; however, there also exists some generalization problems that may be due to the lack of policy training for the model. These findings align with Peng et al.'s in the sense that we see similar performance and increased robustness [10].

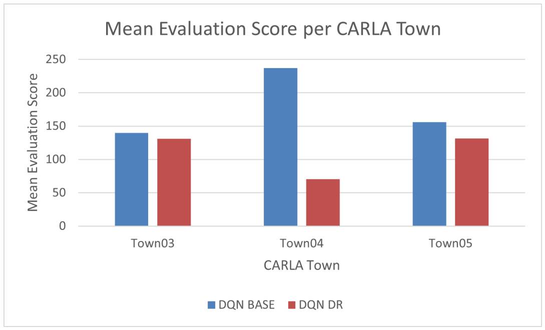

Figure 14: Mean evaluation scores for CARLA models in different towns

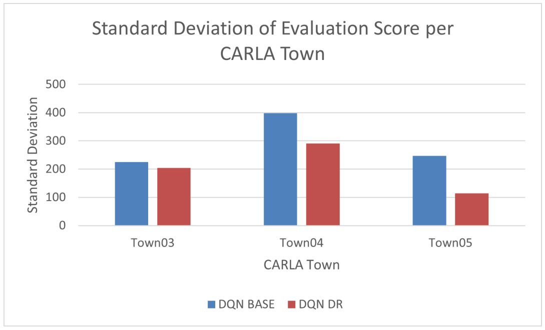

Figure 15: Standard deviation of the evaluation scores for CARLA models in different towns

5 Responsible Research

Although it is discussed that domain randomization is an approach used to bridge the gap between simulation and the real world, the purpose of this research was to investigate their effect on training efficiency and robustness in a simulation environment. The purpose of the experiment is by no means creating an autonomous driving vehicle suitable for use in the real world.

The setup and adaptation of the DQN algorithm were done in collaboration with other researchers in the same research group. We utilized a seed to be able to reproduce the randomness involved in the algorithm, such as the ϵ-greedy policy, or the randomized starting conditions. We ran the base algorithm several times and picked one of the seeds that performed on average to mitigate any possible positive or negative influence the seed can provide on the results. The seed we chose as the representative is "0", and future research can also utilize this seed to reproduce the randomness in the experiment to an extent. In addition, the training data for the baseline DQN model in the CarRacing-v2 environment was shared with one other researcher.

There is also the question of whether the experiment results are significant or not. Due to the time constraints of the research and the hardware constraints of the CARLA simulator, the training results for the experiment come from a single run instead of a combination of multiple runs. In that sense, we can argue that the training results are not significant enough, and it is advised for future research to take this into consideration. Instead, we could run the evaluation tests multiple times to get a better average result. The randomly generated track on CarRacing-v2 and the random starting conditions in CARLA provided enough randomness for the models to generate results not biased by chance. As the code consists of an algorithm from *CleanRL* [5], the *gym-carla* wrapper [1], and the *domain-randomizer*[3], it is easily reproducible.

6 Limitations

We faced several limitations during the experiment. One of the main limitations we encountered was the hardware constraints of the CARLA simulator. To mitigate the hardware constraints, we were allowed to use DelftBlue [3], the TU Delft supercomputer, for experiments involving the CARLA environment. This brought forth the second limitation, the time constraint. Due to congestion on DelftBlue at the start of May, it was not possible to test different parameters and their possible level of influence over the model. There was also a smaller congestion at the start of June, due to this the scheduled experiments were delayed for multiple days before starting. Combined with the fact that we were only allowed a 24-hour reservation for one experiment at a time and had to schedule it afterward to continue the experiment on the same model, the models were not trained until clear sign of convergence. It is possible that this led to a bias in CARLA models, especially for the model trained with domain randomization, and possibly affected the evaluation results.

7 Conclusions and Future Work

To summarize, the research desired to investigate the influence of domain randomizations on training and the robustness of final policies under various testing conditions. We approached this question in two stages by investigating visual domain randomizations in OpenAI's CarRacing-v2 environment and dynamic domain randomizations in the CARLA

[3]https://github.com/montrealrobotics/domain-randomizer

simulator. For visual domain randomization, the experiments indicate a decrease in training efficiency and a general underperformance when compared to the model trained without domain randomization. The evaluation tests performed on the trained models show no noticeable difference differences between each other, thus demonstrating that domain randomization does not contain heavy influence on the robustness, stability, and consistency of the model. On the other hand, we also witness an increase in stability when comparing the evaluation and training data of the DR model, as the standard deviation and the episodic return difference with the base model decrease significantly during evaluation.

For the dynamic domain randomization in CARLA, we see that there is a bigger difference in training efficiency between the models. While the DR model was able to converge in the CarRacing-v2 environment, the model could not converge during training in CARLA thus extending the necessary training time when compared to the baseline model, and we see a generally larger variance in the training data. We also observed some overestimation of the DR agent when comparing its Q-values and episodic returns. As for the robustness testing, we observe a lower standard deviation at the expense of a lower mean, indicating that domain randomization is able to provide more stability and consistency to the model by sacrificing some performance. There is also the influence of the negative bias from the model not being able to converge during training, thus it can be argued that there is a notable improvement by domain randomization on stability.

A possible improvement to this research could be extending the training time for the models so that their training converges and stops trending upward or downward for more optimal training. Other possible studies include changing the evaluation parameters for the robustness testing, as CARLA provides other parameters such as the number of vehicles and pedestrians on the map. It could also be interesting to change the training parameters for domain randomization and test the models with different parameters or ranges to see which parameters influence the model more. One final idea for future research is to investigate other approaches like auxiliary tasks or adversarial training and compare their influence on training and robustness with domain randomization.

References

[1] Jianyu Chen. gym-carla. https://github.com/cjy1992/gym-carla, 2020.

[2] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. Model-free deep reinforcement learning for urban autonomous driving, 2019.

[3] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[4] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio M. Lopez, and Vladlen Koltun. CARLA: an open urban driving simulator.

[5] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[6] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019.

[7] Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, USA, 1992. UMI Order No. GAX93-22750.

[8] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. Active domain randomization, 2019.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[10] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2018.

[11] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image, 2017.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[13] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, mar 1995.

[14] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

[15] Lilian Weng. Domain randomization for sim2real transfer. *lilianweng.github.io*, 2019.

A CARLA Parameters and Randomization Values

A.1 CARLA Parameters

'number_of_vehicles': 100,

'number_of_walkers': 0,

'display_size': 256, # screen size of bird-eye render

'max_past_step': 1, # the number of past steps to draw

'dt': 0.1, # time interval between two frames

'discrete': False # whether to use discrete control space

'discrete_acc': [-3.0, 0.0, 3.0], # discrete value of accelerations

'discrete_steer': [-0.2, 0.0, 0.2], # discrete value of steering angles

'continuous_accel_range': [-3.0, 3.0], # continuous acceleration range

'continuous_steer_range': [-0.3, 0.3], # continuous steering angle range

'ego_vehicle_filter': 'vehicle.lincoln*', # filter for defining ego vehicle

'port': 2000, # connection port

'town': 'Town03', # which town to simulate

'task_mode': 'random', # mode of the task, [random, roundabout (only for Town03)]

'max_time_episode': 1000, # maximum timesteps per episode

'max_waypt': 12, # maximum number of waypoints

'obs_range': 32, # observation range (meter)

'lidar_bin': 0.125, # bin size of lidar sensor (meter)

'd_behind': 12, # distance behind the ego vehicle (meter)

'out_lane_thres': 2.0, # threshold for out of lane

'desired_speed': 8, # desired speed (m/s)

'max_ego_spawn_times': 200, # maximum times to spawn ego vehicle

'display_route': True, # whether to render the desired route

'pixor_size': 64, # size of the pixor labels

'pixor': False, # whether to output PIXOR observation

A.2 Randomization Values

The values for discrete_acc and discrete_steer are given as floats and multiplied once because the lists for these parameters are symmetrical.

- 'out_lane_thres':
  - 'default': 2.0,
  - 'multiplier_min': 0.5,
  - 'multiplier_max': 2.0
- 'discrete_acc':
  - 'default': 3.0,
  - 'multiplier_min': 0.5,
  - 'multiplier_max': 2.0
- 'discrete_steer':
  - 'default': 0.2,
  - 'multiplier_min': 0.5,
  - 'multiplier_max': 2.0