Sim-to-Sim-to-Real: Utilizing high and low-fidelity simulators for predicting Sim-to-Real Transfer and Analysis using small-scale autonomous vehicles

by

Jurriaan Buitenweg

In partial fulfilment of the requirements for the degree of Master of Science at Delft University of Technology
Track: Artificial Intelligence

Faculty: EEMCS
Department: Multimedia
Program: Computer Science
Delft University of Technology
Delft, NL

Thesis advisor: Dr. Cynthia Liem
Daily supervisor: Antony Bartlett
External commitee member: Dr. Annibale Panichella

To be defended publicly on 27-06-2025

Introduction

This thesis is split up into 2 sections. 1 contains the scientific paper which highlights all important background, related work, methodology, approach, results, discussion and conclusion. The paper is aimed towards computer scientists, familiar with AI, specifically, (Deep) reinforcement learning and image processing techniques. For readers not familiar with these topic, section 2 contains all material needed to understand the fundamental topics.

Table of Contents

List of Abbreviations

Sim2Real	Simulation-to-reality
AV	Autonomous Vehicles
AI	Artificial Intelligence
(D)RL	(Deep) Reinforcement Learning.
CNN	Convolutional Neural Network.
RGB	Red Green Blue
PPO	Proximal Policy Optimization
SB	StableBaselines3
CARLA	Car Learning to Act
DT	Duckietown

Chapter 1

Scientific Paper

Sim-to-Sim-to-Real: Utilizing high and low-fidelity simulators for predicting Sim-to-Real Transfer and Analysis using small-scale autonomous vehicles

Jurriaan Buitenweg
Delft University of Technology
j.r.buitenweg@student.tudelft.nl

Thesis Supervisor: Dr. Cynthia Liem
Daily Supervisor: Antony Bartlett

***Abstract*—The Sim2Real gap poses significant challenges for testing autonomous vehicles, often becoming apparent only during high-risk real-world deployments. This research proposes a novel pipeline that leverages both high-fidelity (CARLA) and low-fidelity (Gym-Duckietown) simulators to estimate this gap prior to deployment. The results reveal a strong correlation between performance in Gym-Duckietown and real-world outcomes, suggesting it can serve as potential estimation for real world performance and the Sim2Real gap. Nonetheless, real-world testing remains an essential part of the validation process. Future work should build on these findings to further explore and validate the approach.**

I. Introduction

By 2035, autonomous driving is projected to generate $300 billion to $400 billion in revenue, making it one of the most anticipated technological advancements of this generation [1]. Despite progress in their development, evaluating these systems remains a significant challenge due to safety concerns, high costs, and strict safety requirements [2, 3].

To mitigate these risks, autonomous vehicle (AV) models are typically trained in simulated environments. However, models that perform well in simulation often degrade significantly in the real world, a phenomenon known as the simulation-to-reality (Sim2Real) gap. This gap arises from discrepancies in visual inputs, environmental conditions, dynamics, or dissimilarities between the simulator and the real world.

Although many studies focus on reducing the Sim2Real gap during the training process with techniques like domain randomization [4], relatively little research has been conducted on identifying or quantifying this gap prior to deployment in the real world. Previous work has explored modifying simulation environments to closely resemble real-world conditions [5], which improves the simulator's ability to estimate the Sim2Real gap. However, this approach is costly and relies on having a highly customizable simulator.

The current study proposes a novel approach to estimate the Sim2Real gap using a three-stage evaluation pipeline. A Deep Reinforcement Learning (DRL) model is first trained in CARLA, a high-fidelity simulator, using different image types (RGB, Grayscale, and Segmentation masks), pre-processing methods and network architectures. The trained model is then evaluated in Gym-Duckietown (Gym-DT), a lower-fidelity simulator, resembling the real-world enviroment and finally deployed on a physical Duckiebot robot. This gradual transition allows us to assess performance degradation at each stage to determine whether an intermediate simulator can serve as a proxy for estimating the Sim2Real gap itself. Furthermore, models trained in CARLA are analyzed to see if they have learned similar feature representations compared to those that perform well in Gym-DT and the real world. If this is the case, similarity metrics can be used to make deductions about Sim2Real capabilities. This results in the following research questions

1) To what extent can performance in a low-fidelity simulator, resembling the real-world environment, be used as a predictive metric for the Sim2Real gap for models trained in high-fidelity simulation environments?
2) To what extent can similarity of learned feature representations between autonomous driving models be an indication of real-world performance?

The structure of this paper is as follows: Chapter II provides a review of background research, focusing on the simulators used in the experiments and existing methods addressing the Sim2Real gap. Chapter III discusses related work. Chapter IV, V discuss methodology and approach. Chapter VI presents result. These are then analyzed and discussed in chapter VII and concluded in chapter IX.

II. Background

In this section, the technologies that underlie the work are discussed. This includes information about autonomous vehicles, image representations and (Deep) Reinforcement Learning.

A. Autonomous Vehicles

In this research, the term autonomous vehicle (AV) specifically refers to Level 5 autonomy as defined by the SAE [6] (Table I), which represents full driving automation where no human driver is required under any circumstances. These fully autonomous vehicles present significantly greater challenges compared to lower levels of autonomy [7]. Without human intervention, they demand real-time control outputs to handle all driving scenarios independently. Additionally, training policies must be far more strict, as errors can be life-threatening in the absence of human oversight and control [3]. Lastly, the amount of training data required is much higher, as fully autonomous vehicles must be capable of navigating any type of road in any environment. This also introduces the importance of simulators because real world training in different environments is very costly.

B. Reinforcement Learning

Reinforcement Learning (RL) addresses the problem of enabling a computational agent to make sequential decisions through trial and error to maximize a reward function (Figure 1). An RL model is often visualized as a Markov Decision Process (MDP) [8], characterized by the following components:

- S: The state space.
- A: The set of possible actions.
- $P_a(s, s') = P(S_{t+1} = s' \mid S_t = s, A_t = a)$: The transition probability, representing the likelihood of moving from state s to s' under action a at time t.
- $R_a(s, s')$: The reward received when transitioning from state s to s' under action a. This is determined by a reward function.

A basic reinforcement learning agent interacts with its environment in discrete time steps. At each time step t, the agent receives the current state S_t and reward R_t. It then chooses an action A_t from the set of available actions, which is subsequently sent to the environment. The environment moves to a new state S_{t+1} and the reward R_{t+1} associated with the transition (S_t, A_t, S_{t+1}) is determined. The goal of a reinforcement learning agent is to learn a *policy*:

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1], \quad \pi(s, a) = \Pr(A_t = a \mid S_t = s)$$

that **maximizes** the expected cumulative reward.

In RL, an important distinction exists between *on-policy* and *off-policy* algorithms. On-policy algorithms require the evaluation or improvement of the policy that is actively collecting data, while off-policy algorithms can learn from data generated by an arbitrary policy. Additionally, the action space in reinforcement learning (RL) can be categorized into discrete and continuous types. In the discrete action space, the model learns a policy that outputs

Level	Name	Description
0	**No Automation**	The driver is in complete control of the vehicle at all times.
1	**Driver Assistance**	The vehicle can assist the driver in either speed (e.g., cruise control) or lane position (e.g lane guidance).
2	**Improved Driver Assistance**	The vehicle can assist the driver in speed (e.g., cruise control) and lane position (e.g lane guidance).
3	**Occasional Self-Driving**	The vehicle can take control of both speed and lane position in some situations, such as on limited-access freeways.
4	**Limited Self-Driving**	The vehicle is in full control in some situations, monitors the road and traffic, and will inform the driver when they must take control.
5	**Full Self-Driving Under All Conditions**	The vehicle can operate without a human driver or occupants.

TABLE I: The Levels of Vehicle Autonomy [6]

a probability distribution over a finite set of possible actions. Each action is assigned a probability, and the agent selects an action based on this distribution. In the continuous action space, the model outputs a probability distribution over a continuous range of values. An action is then sampled from this distribution, allowing for a more fluid and precise representation of actions.

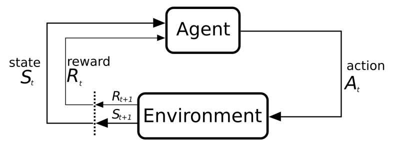


Fig. 1: General representation of a reinforcement learning algorithm.

C. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is similar to traditional reinforcement learning, except that it incorporates a neural network to learn the policy. This integration leverages the power of deep neural networks to learn complex policies directly from raw data. Neural networks in DRL often include feature extraction layers, such as convolutional layers in image-based tasks which automatically identify and transform relevant input patterns into lower-dimensional representations. These extracted features are crucial, as they allow the network to focus on the most informative aspects of the environment, ultimately improving the accuracy and stability of the agent's decisions. The final feature extraction layer plays a key role by producing a compact, high-level summary of the input data, essentially serving as the foundation for decision making in DRL models. Increasing the dimensionality of the feature space can give the model access to richer representations, potentially improving performance in complex environments. However, it can also increase computational cost and the risk of overfitting, especially when training data is limited. DRL is particularly useful in tasks involving self-driving cars, where environments are highly complex and demand adaptive policies [9].

A well-performing DRL on-policy algorithm is Proximal Policy Optimization (PPO) [10]. This algorithm, developed by OpenAI, is a robust and widely used deep reinforcement learning algorithm. It addresses the core challenge of how to make the most significant possible improvements and encourage exploration within a policy using the available data, while avoiding the risk of performance collapse typically caused by the training process becoming trapped in a local minimum. PPO achieves this by employing a clipped objective function to constrain policy updates, ensuring a balance between exploration and stability. This approach avoids the instability often associated with excessively large updates while maintaining high sample efficiency. When using images as input, PPO employs convolutional neural networks (CNNs) to extract visual features, which are then transformed into a latent representation. This representation is passed to the actor-critic network to generate action probability outputs. CNNs consist of multiple layers, where the lower layers typically detect low-level features such as edges, and the higher layers capture more abstract, task-specific patterns. A crucial hyperparameter in a CNN is the dimensionality of the final feature representation layer (often a fully connected layer). This layer has impact on the capacity of the network to represent complex features, influencing its ability to perform well on the target task.

D. *Simulators*

Simulators are digital programs that allow models to be trained in a controlled environment. They can be categorized into low- and high-fidelity simulators. High-fidelity simulators replicate real-world conditions in great detail, providing both complexity and configurational freedom. In contrast, low-fidelity simulators focus only on the essential components of an environment.

For autonomous vehicles (AVs), a high-fidelity simulator might simulate an entire city, complete with traffic, weather conditions, and dynamic changes. On the other hand, a low-fidelity simulator could consist of just a road with markings, without any surrounding elements. Examples of different simulators are:

1) CARLA: CARLA [11] is an open-source, high-fidelity driving simulator designed to facilitate the development and evaluation of autonomous driving systems. CARLA offers an extensive suite of features, including realistic environmental components such as other vehicles, pedestrians, and diverse traffic scenarios. CARLA also allows for gathering data from multiple sensors, including RGB, and segmentation masks through OpenDrive Data [12].

2) Gym-Duckietown: Gym-duckietown (Gym-DT) was developed using using OpenAI's gymnasium framework [13, 14] to serve as a simulator for real-life duckietowns (Figure 2). Gym-DT is fast, open, low-fidelity simulator suitable for developing and evaluation reinforcement learning models. An example of a Gym-DT environment can be seen in Figure 3.

Duckietown is a widely recognized framework for hands-on learning and research in robotics. It provides an easy to use and inexpensive way to do experiments involving autonomous driving robots (Figure 2) in simulation and in the real world.

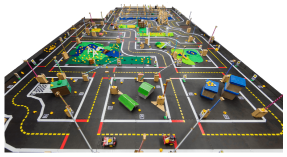


Fig. 2: A real life duckietown environment

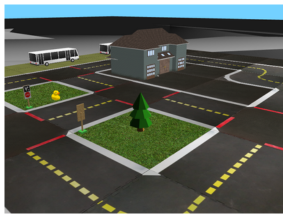


Fig. 3: A simulated duckietown environment

E. *Sim2Real*

Simulations are not perfect copies of reality, therefore models trained in simulators usually perform worse after being deployed in the real world. This performance difference is known as the sim-to-real (Sim2Real) gap. This phenomenon can be caused by different factors, including but not restricted to different colors between environments, physics, lighting conditions or unexpected encounters. Additionally, the simulator itself may fail to accurately capture critical aspects of the real-world environment.

F. *Domain Randomization*

Domain randomization [4] is a technique used to improve a model's robustness and generalization capabilities and reduce the Sim2Real gap. The technique consists of varying certain aspects of the training environment or observations. This method introduces controlled randomness into the observations or the environment, forcing the model to learn invariant features that are not specific to any particular instance. In the context of visual tasks, for example, visual domain randomization involves altering the appearance of images, such as changing color spaces, textures, lighting, or backgrounds of features that should not matter to the model. By applying a set of random perturbations to these visual parameters, the model becomes less sensitive to specific environmental conditions, enhancing its ability to generalize to unseen scenarios. Domain randomization reduces overfitting and increases robustness to real-world variability [4].

G. *Input representations*

In DRL models for AVs, visual inputs are usually images. The image can be a camera image in a certain color space (RGB or Grayscale), or a result of a semantic segmentation network.

1) Camera images: RGB images are captured directly from a camera and offer complex and detailed representations of an environment. However, their complexity can be challenging for training robust deep reinforcement learning (DRL) models. Environmental changes such as lighting variations or weather conditions in simulation can significantly change key visual features within RGB inputs, further complicating the generalization of learned policies [4].

Grayscale images contain only a single intensity channel, they reduce the input dimensionality for neural networks, allowing models to train faster and with fewer parameters, an advantage when real-time decisions are critical. In many driving scenarios, essential information such as lane markings, road edges, and obstacles can be accurately perceived through contrast and shape rather than color, making grayscale sufficient for tasks like lane following or collision avoidance.

2) Semantic Segmentation: Semantic segmentation masks has long been researched as input data to machine learning models, and provide great results [15]. Semantic segmentation masks are the result of a process called semantic segmentation. This technique classifies each pixel in an image as one of a finite set of classes and assigns a specific color to each class (figure 4). This process provides an image representation which is generalizable across multiple environments and reduces the complexity significantly, making for more robust models when used as input. The drawback of this technique is that it can be time intensive to obtain the segmentation masks because of per pixel classification and might need a lot of groundtruth data to teach a model to do this correctly.

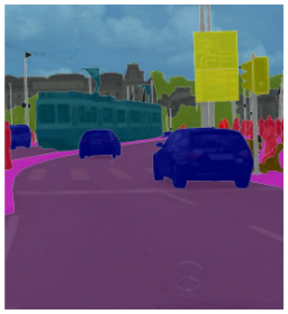


Fig. 4: Segmentation image example

A fast end-end segmentation network is FastSCNN [16] (Figure 5). End-End means that you take in an input images and your desired output image is achieved by a single pass through the network.

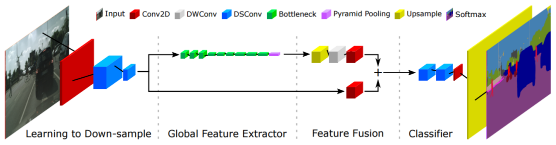


Fig. 5: Network architecture of Fast SCNN [16]

III. Related Work

A. *Sim2Real prediction through simulation*

Sim2Real predictivity through a simulator was previously explored in the domain of robotic arm manipulation [5] . By using a metric called the Sim2Real correlation coefficient, the study quantifies how well a simulator can predict the real-world capabilities of a trained model. This metric is further employed to guide the tuning of simulator parameters such as visuals and dynamics to improve Sim2Real predictivity.

The work in this paper builds upon this systematic approach by applying it to the domain of self-driving cars. Additionally, while the previous work focuses on models trained within the same simulation where evaluation occurs, this study expands the scope by evaluating models trained across different simulators, allowing the assessment of generalization and robustness in Sim2Real transfer. This is particularly useful because there are many autonomous vehicle simulators available, each with varying fidelity and characteristics, so evaluating across different simulators better reflects the diversity of real-world conditions and helps identify models that generalize well beyond a single simulation environment.

B. Quantifying the Sim2Real Autonomous Driving Simulations by using attention map Similarity

Sim2Real transfer can also be quantified by comparing attention maps, as demonstrated in [17]. Their method measures the similarity between the histograms of the middle self-attention maps generated by models when processing images from the simulator and the real world. A higher histogram similarity suggests that the model is extracting similar features across both environments and thus also achieves similar performance.

While this approach offers a scientifically sound way to quantify the Sim2Real gap, histogram comparison ignores spatial information within the attention maps. In contrast, the work in this paper employs a similarity metric that accounts for spatial differences, providing a more detailed comparison of feature extraction between simulated and real-world inputs. This is useful because even if latent representations differ between models, they may still capture the same underlying features.

IV. Methodology

In this section, the general methodology used to do the experiments is discussed, breaking it down into three distinct sections: Training, Testing and Evaluation.

A. Training

Models are trained using different types of input image representations, each with varying levels of visual complexity:

- RGB – High complexity
- Grayscale – Medium complexity
- Segmentation masks – Low complexity

For each input type, models are trained while varying one of the following components that could influence generalization performance:

- Domain randomization: Enhances generalization by making the models more robust to environmental variations [4].
- Cropping: Cropping input images can improve performance and robustness by reducing irrelevant background features, effectively narrowing the feature space [18].
- Model architecture: Expanding the final feature extraction layer enables the model to learn a wider range of informative features, improving its learning capability, but also increasing the risk of overfitting [19].

Note that when using segmentation masks, domain randomization becomes unnecessary. This is because segmentation masks inherently provide a simplified and environment-independent representation, already abstracting away visual variations in the background.

All models are trained in the CARLA simulator. CARLA is selected due to its high rendering fidelity, flexibility, and ease of integration with reinforcement learning pipelines [20]. Although alternatives such as Gazebo [21] offer better vehicle dynamics simulation, they pose significant integration challenges for DRL and are out of scope for this study. For each image type, a baseline model is trained in Gym-DT using domain randomization and cropping. This model demonstrates validated performance in both the Gym-DT environment and the real world, making it a strong benchmark. Gym-DT is selected because it closely resembles real-world testing scenarios that utilize Duckietown products.

B. Testing

Post-training, the models are evaluated in both CARLA and Gym-DT where performance metrics are gathered. After evaluation in the two simulators, the final step involves deploying all models onto a real-world duckiebot to assess real-world performance. This approach allows for evaluating whether performance in Gym-DT serves as a reliable indicator of real-world performance for models trained in CARLA. While Duckiebots are used in this study, the methodology is applicable to any small-scale autonomous vehicle with a similar observation and action space.

C. Evaluation

Evaluation is done on three different fronts: performance, Sim2Real transferability and model similarity.

- Performance: Model performance is assessed based on its ability to navigate predefined environments, comparing it against a baseline model.
- Sim2Real Correlation: To quantify how well a simulator predicts real-world performance, the correlation between simulated and real-world results is analyzed.
- Feature Similarity: This metric is used to investigate whether models that achieve strong performance in CARLA learn features similar to those associated with high performance in Gym-DT and real-world environments.

V. Approach

This section outlines the technical implementations used to obtain input images, configure training environments, and train the DRL models. Technical methods used in the training process are discussed, followed by the testing setup and evaluation metrics.

A. Domain randomization

Domain randomization (DR) is applied in the training process. In CARLA, this is done by randomizing weather conditions on each reset. This has impact on the overall colors and features of the environment. In for the baseline Gym-DT model, DR is applied by randomizing environment background each time the environment has been resetted. A reset occurs when the vehicle crashes in the training process or a training episode reaches its maximum length.

B. Camera Images

Camera images are obtained via callback functions using the APIs of the respective simulators. Raw RGB images are used directly for RGB input, while grayscale images are generated by converting the RGB images using OpenCV [22]. In real-world deployment, images are received through ROS communication with the Duckiebot [23]. All images are resized to C×120×160 and, if required by the specific model, cropped to C×80×160, where C denotes the number of channels (3 for RGB and 1 for grayscale). Finally, images are normalized by diving all pixel values by 255.

C. Semantic Segmentation Masks

In CARLA, OpenDRIVE [12] data is leveraged to directly generate semantic segmentation masks. This metadata is embedded within the materials that the vehicle interacts with (e.g., road, sidewalk), enabling perfect segmentation without the need for a trained model. This approach supports training across various environments, such as asphalt and dirt roads.

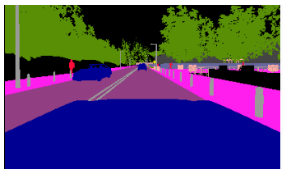


Fig. 6: Sample segmentation mask from the CARLA simulator

In Gym-DT, segmentation masks are approximated by modifying environment textures to represent different classes (e.g., lane, background), removing the need for a segmentation model during simulation. This helps isolate the cause of poor performance, as issues related to a segmentation network can be ruled out.

However, for real-world deployment, a real-time segmentation model is needed. The FastSCNN architecture is used, which is trained using the Duckietown Segmentation Dataset [24] to perform semantic segmentation on RGB inputs from the duckiebot (figure 7).

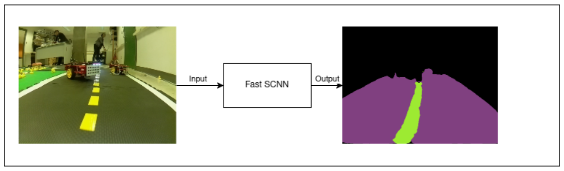


Fig. 7: Illustration of the input and output of the Fast SCNN model.

For simplicity in our experiments, segmentation masks consist of three classes: background (black), lane (purple), and lane markings (green). The segmentation mask, in the form of an RGB image, is then converted into a class label image of either 3x120x160 or 3x80x160, depending on whether it's cropped. Finally, segmentation masks are normalized by diving all pixel values by the number of classes, in this case 3.

D. Training Algorithm

All models are trained using the Proximal Policy Optimization (PPO) algorithm from the Stable Baselines3 library [25]. PPO is selected for its robustness to hyperparameter choices and its fast convergence. This is beneficial when training multiple models with different input modalities.

Models with domain randomization (DR) are trained for 2 million timesteps, while those without DR are trained for 1 million timesteps. Since models trained without DR are known to converge more quickly [26], this decision was made to reduce computational cost and training time.

An exponential decaying learning rate schedule is used and the reward function utilized for training is defined as:

$$\text{penalty}_{\text{lane}} = \min\left(\frac{d}{d_{\text{max}}},\ 1\right) \tag{1}$$

$$\text{penalty}_{\text{angle}} = \frac{1-\cos(\theta)}{2} \tag{2}$$

$$\text{reward} = 1 - 0.8 \cdot \text{penalty}_{\text{lane}} - 0.2 \cdot \text{penalty}_{\text{angle}} \tag{3}$$

where:

- θ: angle between the vehicle's heading and the tangent of the right lane
- d: lateral distance from the center of the vehicle to the lane tangent
- d_{max}: The maximum allowed deviation, this is used for normalization purposes.

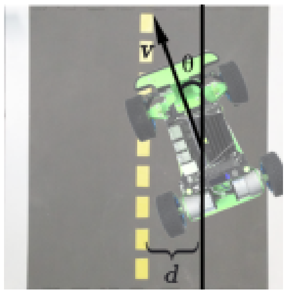


Fig. 8: Variables used in the reward function [27]

When the car exceeds d_{max}, the environment is reset.

Only steering control is learned during training, while the forward speed is kept constant. Steering control entails how much the car should steer left or right. This is expressed in a single float value between -1 and 1 where -1 indicates a full left turn, and 1 a full right turn. All training parameters can be seen in table II. The values used are based on commonly adopted hyperparameters for PPO.

Parameter	Value
learning_rate	`exponential(3e-4, 1e-5, 3)`
n_steps	2048
batch_size	128
n_epochs	10
gamma	0.99
gae_lambda	0.95
clip_range	0.2
ent_coef	1e-4
vf_coef	0.5
max_grad_norm	0.5
seed	42

TABLE II: Hyperparameter configuration. $exponential(x, y, d)$ indicates an exponential decay from x to y with rate d.

E. Feature extractor

As the feature extractor for PPO, the standard NatureCNN architecture provided by Stable Baselines3 (Figure 9) is used. In our experiments, the final linear layer is modified to have a size of 512 instead of the default 256. This adjustment allows for investigation whether increased feature learning capacity leads to improved generalization and whether such improvements are observable in the intermediate simulation environment and in the real world.

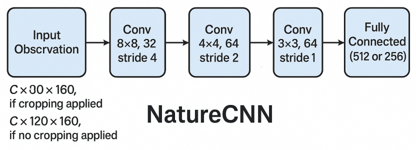


Fig. 9: NatureCNN architecture

F. *Environments*

Models in CARLA are trained to drive along the outer lap of the Town02 map [1] (Figure10). This map was chosen because it provides an environment for testing basic lane-following maneuvers, including driving straight as well as making left and right turns. The baseline model is trained on the loop map in Gym-Duckietown (Gym-DT), offering a consistent comparison point across environments.

All models are evaluated in three settings: CARLA, Gym-Duckietown, and a real-world Duckiebot setup (Figure 10). In the CARLA environment, evaluation takes place in Town02, with randomized starting positions selected from various spawn points along the outer loop. This encourages robust and diverse lap-driving behavior.

In Gym-DT, each episode starts at a random position on one of the straight road segments of the loop map. To introduce controlled variability, the vehicle's initial heading angle is randomly perturbed within the range $[-\pi/8, \pi/8]$, ensuring variation without making the task infeasible.

The real-world Duckiebot setup closely replicates the Gym-DT layout. This alignment is designed to maximize the transferability of insights and performance between the simulated and physical environments. The robot is placed on one of the straight road parts randomly, balancing inner and outer laps.

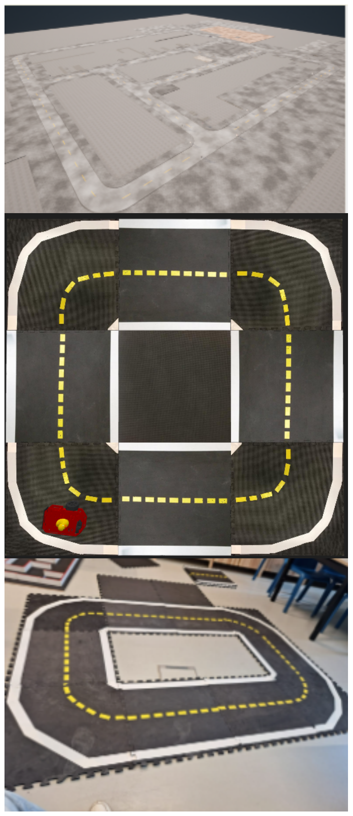


Fig. 10: Evaluation environments: CARLA (top), Gym-DT (middle), and real Duckiebot setup (bottom). In CARLA, the vehicle drives a lap around the outer road.

G. *Metrics and Procedure*

Below, the key metrics used in our analysis are defined. First, performance metrics are discussed, which help evaluate how well models perform across different environments. Next, Sim2Real correlation metrics are introduced, which address our research question of whether an intermediate simulator can serve as a reliable predictor of the Sim2Real gap. Finally, similarity metrics are pre-

[1]https://carla.readthedocs.io/en/latest/map_town02/

sented, used to assess whether models trained in different simulators develop similar perceptions of their environment.

1) Performance metrics: Each model is evaluated in simulation over a total of 100K timesteps in both CARLA and Gym-DT. During this period, the number of laps completed without crashing and the overall mean reward are recorded.

- Average percentage of completion: This metric represents the overall success rate of the vehicle in navigating the environment, calculated as the percentage of trials in which the vehicle completes one full lap. A successful navigation is defined as completing a full circuit of the track without failure, making this metric a direct measure of task completion performance.
- Mean reward: In the simulated environments (CARLA and Gym-DT), the mean reward is used as a proxy for lane alignment and driving quality.

For real-world testing, the mean reward is not available; instead, only the average percentage of completion is reported, computed over 10 trials, balancing inner- and outer-lane trials.

2) Sim2Real correlation: The Sim2Real Correlation coefficient between CARLA and the real-world and gym-DT and the real-world is computed. The Sim2Real Coefficient (SRCC) is an indicator used to evaluate how well performance in a simulated environment predicts performance in the real world [15]. A higher SRCC (up to 1) suggests that the simulator performance closely correlates with real-world performance, indicating a smaller Sim2Real gap. In contrast, a SRCC close to 0 implies a poor correlation between the performance in simulation and reality and a larger Sim2Real gap. The SRCC is calculated by taking the sample pearson correlation between average percentage of completion in simulation and the real world.

Since the SRCC is based on Pearson correlation, it is undefined (N.A) when there is a lack of variation in the results. Therefore, the mean difference in average percentage of completion is reported to provide an additional measure of performance comparison.

3) Similarity metrics: To compare how similarly different models extract features, Centered Kernel Alignment (CKA) is used. CKA is a reliable method for comparing internal representations across neural networks, even when those representations are rotated or scaled differently [28]. This makes it well suited for identifying shared patterns learned by models trained from different starting points.

In this research, CKA is computed between each layer of a model trained with domain randomization (DR) and a model trained without it. The goal is to compare whether models that generalize across environments learn similar features to those that are fine-tuned for a specific environment, and whether this similarity corresponds to performance in that environment. If a strong correlation is found, CKA could be used as an indicator of how well a model might perform in a given environment without needing to deploy it in the real world.

To ensure a reliable CKA comparison between models, the exact same input data is used across evaluations. Pre-recorded videos from both Gym-DT and a lap around the Duckiebot map are served as the input for all models during the analysis.

VI. Results

A. Performance & correlation results

Tables V, VI and VII summarize the performance results across environments for RGB, grayscale, and segmentation inputs, respectively. These tables report the percentage of task completion. Highlighted values represent the models with the highest completion percentage for each corresponding environment. For an extended overview, including the mean rewards per model, please refer to Appendix A1. In Table III mean difference in average completion rate are visualized, and in table IV, the SRCC is computed over average completion rates of the models.

Image Type	**CARLA**	**Gym-DT**
RGB	66.25	**10.6**
Grayscale	75.8	**14.9**
Segmentation	47.5	**1.5**

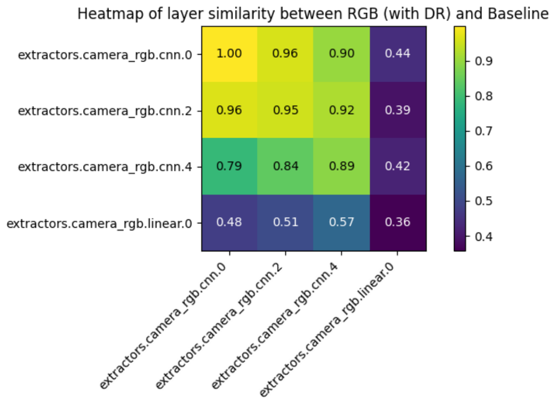
TABLE III: Mean difference in performance when comparing CARLA and Gym-DT to real world performance for each image input type.

Image Type	**CARLA**	**Gym-DT**
RGB	N.A	**0.80**
Grayscale	-0.008	**0.41**
Segmentation	N.A	**0.89**

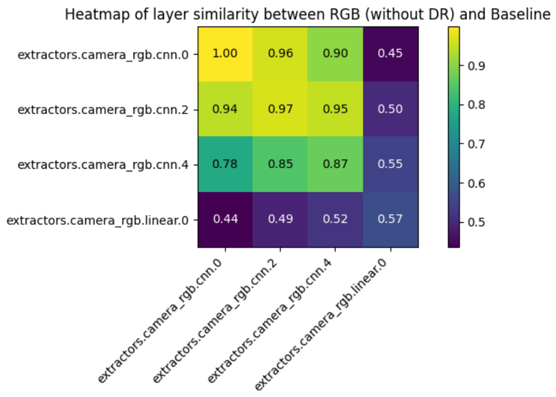
TABLE IV: Sim2Real Correlation Coefficient (SRCC) across CARLA and Gym-Duckietown for each image input type. A cell is N.A if SRCC computation was not possible due to constant values in one of the performance sets.

B. Similarity results

Figure 11 displays heatmaps that illustrate the similarity between feature extraction results of layer of models trained with various input types and a baseline model trained in the Gym-Duckietown environment and duckiebot environment. Additional heatmaps comparing grayscale models can be found in Appendix A2.



(a) RGB with domain randomization



(b) RGB without domain randomization

Fig. 11: Heatmaps comparing feature learning similarity using CKA between models trained in CARLA with (a) and without (b) domain randomization (DR) and a model trained in Gym-DT, evaluated on a real-world test scenario. On the vertical axis, layers are ordered from lower (top) to higher (bottom), while on the horizontal axis, layers range from lowest (left) to highest (right). For more information about the architecture, see figure 9.

TABLE V: Accuracy with RGB input across different environments, grouped by architecture

Architecture	DR	Cropping	Train Env.	CARLA	Gym-DT	Duckiebot
256	✓	✓	Duckietown	0%	99%	100%
	✓	✓	CARLA	100%	**75%**	**80%**
	✓	–	CARLA	100%	31%	40%
	–	✓	CARLA	100%	30%	30%
	–	–	CARLA	100%	20%	0%
512	✓	✓	Duckietown	0%	70%	70%
	✓	✓	CARLA	100%	**42%**	**40%**
	✓	–	CARLA	100%	31%	30%
	–	✓	CARLA	100%	20%	50%
	–	–	CARLA	100%	18%	0%

TABLE VI: Accuracy with Grayscale input across different environments, grouped by architecture

Architecture	DR	Cropping	Train Env.	CARLA	Gym-DT	Duckiebot
256	✓	✓	Duckietown	0%	84%	10%
	✓	✓	CARLA	100%	**51%**	10%
	✓	–	CARLA	73%	28%	**20%**
	–	✓	CARLA	100%	0%	0%
	–	–	CARLA	100%	0%	0%
512	✓	✓	Duckietown	0%	49%	20%
	✓	✓	CARLA	74%	0%	0%
	✓	–	CARLA	90%	**25%**	**50%**
	–	✓	CARLA	100%	23%	50%
	–	–	CARLA	100%	18%	0%

TABLE VII: Accuracy with Segmentation input across different environments, grouped by architecture

Architecture	Cropping	Train Env.	CARLA	Gym-DT	Duckiebot
256	✓	Duckietown	42%	65%	70%
	✓	CARLA	100%	**61%**	**60%**
	-	CARLA	100%	50%	50%
512	✓	Duckietown	54%	60%	50%
	✓	CARLA	100%	**55%**	**50%**
	-	CARLA	100%	50%	50%

VII. Discussion

The results in Tables V, VI, and VII demonstrate that it is feasible to train models in a high-fidelity simulator like CARLA and still achieve performance in both the low-fidelity Gym-Duckietown (Gym-DT) environment and the real world. That being said, the reverse does not hold: using CARLA to evaluate models trained in a low-fidelity simulator proved ineffective, as the baseline model failed to perform in the high-fidelity CARLA environment. The first is a promising finding, as it indicates that models pretrained in high-fidelity simulators can have their generalization capabilities effectively assessed using an intermediate simulator.

Especially using the combination of RGB input images together with domain randomization and cropping resulted in performance comparable to the baseline. Models trained using grayscale input images, performed less compared to RGB, this could be to a lack of feature complexity. Increasing final layer dimensions seem to decrease overall performance of models in both Gym-DT and the real world, this can be the consequence of overfitting to the CARLA environment.

The Sim2Real correlation coefficient (SRCC) values indicate that performance in Gym-Duckietown provides a more reliable estimation of Sim2Real transferability than performance in CARLA. This trend remains consistent regardless of the model architecture or preprocessing technique used. In other words, the better a model performs in Gym-DT, the more likely it is to perform well in the real world. Although the SRCC could not be computed for RGB and segmentation due to a lack of variation in the CARLA simulator results, the mean difference reveals a similar trend: performance in Gym-DT more closely aligns with real-world performance than that observed in CARLA. These insights have several practical implications:

1) Performance in an intermediate low-fidelity simulator gives an early estimation of a model's real world performance without requiring access to a full-scale physical vehicle, an expensive, time-consuming, and potentially dangerous resource.
2) Developers can use the intermediate simulator to identify and debug model failures before proceeding to real-world testing. Unlike real-world evaluations, where multiple environmental and dynamic factors can interfere, simulation provides controlled conditions to isolate and resolve specific issues.

For instance, it is observed that a model exhibiting difficulty in executing right turns in Gym-DT also failed similarly in real-world tests. This shows how an intermediate simulator can act as a filter to catch predictable failure modes in a safe and cost-effective way.

Furthermore, segmentation models trained in CARLA exhibited particularly strong transferability. This is useful because CARLA supports training using OpenDRIVE metadata, an inexpensive alternative to manually labeled datasets. These findings magnify the value of using structured metadata like OpenDRIVE to train autonomous driving models efficiently, reducing both cost and manual effort. A drawback of this method is that in the real-world, it still requires a semantic segmentation network, which can greatly impact Sim2Real transfer.

A. Similarity results

The heatmaps in Figure 11 show that, despite differences in training environments, models trained on self-driving tasks tend to learn similar features in their convolutional layers, particularly in the bottom layers. This is likely because these layers capture low-level visual features such as edges and shapes, which remain consistent across environments.

Interestingly, even without the use of domain randomization, models trained in CARLA learn feature representations that are closely aligned with those of models trained in Gym-DT. One might expect this similarity in learned features to translate into similar performance across environments; however, this was not observed. This suggests that feature representation similarity alone is not a reliable indicator of Sim2Real performance.

The most significant differences between models emerge in the linear (fully connected) layers, especially when evaluated on real-world data. This points to the linear layers playing a critical role in bridging the gap between simulation and reality, and may explain discrepancies in performance despite shared convolutional features.

VIII. Limitations

Since this is the first study to explore the use of an intermediate simulator as an indicator for Sim2Real performance in autonomous driving, several limitations exist.

First, this work utilizes only Gym-Duckietown and Duckiebots to assess the Sim2Real gap. To determine the broader applicability of this method, future studies should include additional test scenarios involving different vehicles and simulation platforms.

Second, model training has a significant impact on performance. In this study, the Proximal Policy Optimization (PPO) algorithm is used. However, other reinforcement learning algorithms such as Soft Actor-Critic (SAC) [29] and Twin Delayed Deep Deterministic Policy Gradient (TD3) [30], may result in different levels of performance and generalization. Future work should investigate whether the proposed method generalizes across a variety of RL algorithms.

Third, during experimentation, it became clear that vehicle dynamics such as wheel slippage and differences in turning radius have a significant impact on real-world model performance and contribute notably to the Sim2Real gap. Although this study did not incorporate vehicle dynamics, these factors are critical. Future work should explore methods to analyze them prior to real-world deployment.

Moreover, this study simplifies the control problem by only having the models output steering action. To better reflect real-world driving scenarios and assess compatibility for full-scale AV's, future research should expand the action space to include throttle and braking control.

Finally, to assess the practical and industrial relevance of this approach, the same method should be applied on full-scale autonomous vehicles in a safe environment to assess industry usefulness.

IX. Conclusion

In this research, a novel methodology is proposed that uses an intermediate low-fidelity simulator to estimate the Sim2Real gap in autonomous driving models. Testing self-driving car models in simulation is significantly safer and more cost-effective than deploying them directly in real-world environments on real-scale vehicles.

Multiple models are trained with proximal policy optimization (PPO), varying configurations (applying domain randomization, cropping or changing model architecture) in the high-fidelity CARLA simulator, aiming to explore factors that affect generalization and Sim2Real transfer. These models were then evaluated both in the intermediate Gym-Duckietown (Gym-DT) simulator and on a physical Duckiebot. Results are gathered on performance (average percentage of completion & mean reward) and used to analyze the correlation between model performance in the intermediate simulator and in the real world.

The findings suggest that it is feasible to train models in a high-fidelity simulator such as CARLA and use a low-fidelity simulator to estimate real-world performance, thereby providing an approximation of the Sim2Real gap. However, results obtained in the intermediate simulator are not sufficiently reliable to eliminate the need for real-world testing.

Training in a low-fidelity simulator like Duckietown and evaluating in CARLA proved to be much less effective. This indicates that the proposed method is well-suited for high-to-low fidelity transfer like discussed above, but not the reverse. Future work should look at broadening this methodology by incorporating multiple training algorithms, simulators and environments. Future work should aim to investigate if this methods generalizes when using other simulators and environments, as well as using full-scale autonomous vehicles.

References

[1] Johannes Deichmann. *The future of autonomous vehicles (AV) — McKinsey*. URL: https : / / www . mckinsey . com / industries / automotive - and - assembly / our - insights /

autonomous-drivings-future-convenient-and-connected#/ (visited on 11/21/2024).

[2] Margarita Martínez-Díaz and Francesc Soriguera. “Autonomous vehicles: theoretical and practical challenges”. In: *Transportation Research Procedia*. XIII Conference on Transport Engineering, CIT2018 33 (Jan. 2018), pp. 275–282. ISSN: 2352-1465. DOI: 10.1016/j.trpro.2018.10.103. URL: https://www.sciencedirect.com/science/article/pii/S2352146518302606 (visited on 04/06/2025).

[3] B. Padmaja et al. “Exploration of issues, challenges and latest developments in autonomous cars”. In: *Journal of Big Data* 10.1 (May 2023), p. 61. ISSN: 2196-1115. DOI: 10.1186/s40537-023-00701-y. URL: https://doi.org/10.1186/s40537-023-00701-y (visited on 04/08/2025).

[4] Josh Tobin et al. *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*. arXiv:1703.06907 [cs]. Mar. 2017. DOI: 10.48550/arXiv.1703.06907. URL: http://arxiv.org/abs/1703.06907 (visited on 04/01/2025).

[5] Abhishek Kadian et al. “Sim2Real Predictivity: Does Evaluation in Simulation Predict Real-World Performance?” In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020). arXiv:1912.06321 [cs], pp. 6670–6677. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2020.3013848. URL: http://arxiv.org/abs/1912.06321 (visited on 05/12/2025).

[6] T Vesselenyi. *Autonomous vehicles: classification, technology and evolution*. en. URL: https://www.researchgate.net/publication/353858131_Autonomous_vehicles_classification_technology_and_evolution (visited on 06/16/2025).

[7] patale. *Autonomous Vehicle Levels & Trends*. en. URL: https://www.researchgate.net/publication/384729301_Autonomous_Vehicle_Levels_Trends (visited on 04/06/2025).

[8] Martijn Otterlo and Marco Wiering. “Reinforcement Learning and Markov Decision Processes”. In: *Reinforcement Learning: State of the Art* (Jan. 2012). ISBN: 978-3-642-27644-6, pp. 3–42. DOI: 10.1007/978-3-642-27645-3_1.

[9] B. Ravi Kiran et al. *Deep Reinforcement Learning for Autonomous Driving: A Survey*. arXiv:2002.00444 [cs]. Jan. 2021. DOI: 10.48550/arXiv.2002.00444. URL: http://arxiv.org/abs/2002.00444 (visited on 04/06/2025).

[10] John Schulman et al. *Proximal Policy Optimization Algorithms*. arXiv:1707.06347 [cs]. Aug. 2017. DOI: 10.48550/arXiv.1707.06347. URL: http://arxiv.org/abs/1707.06347 (visited on 12/11/2024).

[11] Alexey Dosovitskiy et al. *CARLA: An Open Urban Driving Simulator*. arXiv:1711.03938. Nov. 2017. DOI: 10.48550/arXiv.1711.03938. URL: http://arxiv.org/abs/1711.03938 (visited on 11/25/2024).

[12] Vivien Potó et al. “OpenDRIVE Standard for Road Description”. In: Nov. 2019.

[13] Maxime Chevalier-Boisvert et al. *Duckietown Environments for OpenAI Gym*. Publication Title: GitHub repository. 2018. URL: https://github.com/duckietown/gym-duckietown.

[14] Greg Brockman et al. *OpenAI Gym*. arXiv:1606.01540 [cs]. June 2016. DOI: 10.48550/arXiv.1606.01540. URL: http://arxiv.org/abs/1606.01540 (visited on 04/06/2025).

[15] John So et al. *Sim-to-Real via Sim-to-Seg: End-to-end Off-road Autonomous Driving Without Real Data*. arXiv:2210.14721 [cs]. Oct. 2022. DOI: 10.48550/arXiv.2210.14721. URL: http://arxiv.org/abs/2210.14721 (visited on 03/31/2025).

[16] Rudra P. K. Poudel, Stephan Liwicki, and Roberto Cipolla. *Fast-SCNN: Fast Semantic Segmentation Network*. arXiv:1902.04502 [cs]. Feb. 2019. DOI: 10.48550/arXiv.1902.04502. URL: http://arxiv.org/abs/1902.04502 (visited on 05/12/2025).

[17] Seongjeong Park et al. *A Study on Quantifying Sim2Real Image Gap in Autonomous Driving Simulations Using Lane Segmentation Attention Map Similarity*.

arXiv:2306.10491 [cs]. June 2023. DOI: 10 . 48550 / arXiv . 2306 . 10491. URL: http://arxiv.org/abs/2306.10491 (visited on 01/16/2025).

[18] Junhui Liang, Ying Liu, and Vladimir Vlassov. "The Impact of Background Removal on Performance of Neural Networks for Fashion Image Classification and Segmentation". In: *2023 Congress in Computer Science, Computer Engineering, &amp; Applied Computing (CSCE)*. arXiv:2308.09764 [cs]. July 2023, pp. 1960–1968. DOI: 10.1109/CSCE60160.2023.00323. URL: http://arxiv.org/abs/2308.09764 (visited on 06/11/2025).

[19] S. H. Shabbeer Basha et al. "Impact of Fully Connected Layers on Performance of Convolutional Neural Networks for Image Classification". en. In: *Neurocomputing* 378 (Feb. 2020). arXiv:1902.02771 [cs], pp. 112–119. ISSN: 09252312. DOI: 10.1016/j.neucom.2019.10.008. URL: http://arxiv.org/abs/1902.02771 (visited on 06/20/2025).

[20] Prabhjot Kaur et al. *A Survey on Simulators for Testing Self-Driving Cars*. arXiv:2101.05337. Jan. 2021. DOI: 10 . 48550 / arXiv . 2101 . 05337. URL: http://arxiv.org/abs/2101.05337 (visited on 11/25/2024).

[21] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. Sept. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727. URL: https://ieeexplore.ieee.org/document/1389727 (visited on 11/25/2024).

[22] Ivan Culjak et al. "A brief introduction to OpenCV". In: *2012 Proceedings of the 35th International Convention MIPRO*. May 2012, pp. 1725–1730. URL: https://ieeexplore.ieee.org/document/6240859 (visited on 06/20/2025).

[23] Morgan Quigley et al. "ROS: an open-source Robot Operating System". en. In: ().

[24] Hamudi Naanaa. *hamnaanaa/Multiclass-Semantic-Segmentation-Duckietown-Dataset*. original-date: 2023-01-11T00:16:14Z. Mar. 2025. URL: https://github.com/hamnaanaa/Multiclass-Semantic-Segmentation-Duckietown-Dataset (visited on 05/12/2025).

[25] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[26] Rawal Khirodkar and Kris M. Kitani. *Adversarial Domain Randomization*. en. arXiv:1812.00491 [cs]. Aug. 2021. DOI: 10.48550/arXiv.1812.00491. URL: http://arxiv.org/abs/1812.00491 (visited on 06/20/2025).

[27] Thomas P. A. Wiggers and Arnoud Visser. "Learning to Drive Fast on a DuckieTown Highway". en. In: *Intelligent Autonomous Systems 16*. Ed. by Marcelo H. Ang Jr et al. Vol. 412. Series Title: Lecture Notes in Networks and Systems. Cham: Springer International Publishing, 2022, pp. 183–194. ISBN: 978-3-030-95891-6 978-3-030-95892-3. DOI: 10.1007/978-3-030-95892-3_14. URL: https://link.springer.com/10.1007/978-3-030-95892-3_14 (visited on 12/13/2024).

[28] Simon Kornblith et al. *Similarity of Neural Network Representations Revisited*. arXiv:1905.00414 [cs]. July 2019. DOI: 10 . 48550 / arXiv . 1905 . 00414. URL: http://arxiv.org/abs/1905.00414 (visited on 05/12/2025).

[29] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. en. Jan. 2018. URL: https://arxiv.org/abs/1801.01290v2 (visited on 06/20/2025).

[30] Stephen Dankwa and Wenfeng Zheng. "Twin-Delayed DDPG: A Deep Reinforcement Learning Technique to Model a Continuous Movement of an Intelligent Robot Agent". en. In: *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*. Vancouver BC Canada: ACM, Aug. 2019, pp. 1–5. ISBN: 978-

1-4503-7625-9. DOI: 10 . 1145 / 3387168 . 3387199. URL: https : / / dl . acm . org / doi / 10 . 1145 / 3387168 . 3387199 (visited on 06/20/2025).

.

Chapter 2

Supplementary Material

2.1 Deep Learning

Machine learning models can be supervised and unsupervised. Supervised means that there needs to be a "ground-truth" label for every training sample. Unsupervised means this is not necessary. In this case, the model can learns from its own input. Supervised learning is much harder to do since data collection is difficult, that being said, the performance of the model sometimes exceeds unsupervised methods because of more exact predictions and feature extraction [1].

Deep learning is a subgenre of machine learning that uses complex structures that mimic the human brain. These structures are called "neural networks". Compared to traditional machine learning, the way these networks predict an output can be hard to traced and neural networks are sometimes considered "black box", meaning you do not know how an output is predicted.

A traditional deep learning model consists of weights, where each weight is a real number, and some summation function at the end of it. A very basic neural network is a perceptron (Figure 2.1). Here, inputs are summed at put through a non-linear node. In most cases, a ReLU function is used($ReLU(x) = (max(0, x))$). Deep learning models learn by a process called "backpropagation".

This method uses a loss function, which quantifies the difference between the predicted outputs of a machine learning algorithm and the actual target values. The loss function is used to tune the weights. By calculating the gradient of the loss function with respect to the neural network's weights, it can adjust these weights to by a small step (gradient decent) to improve model performance step by step. The size of these steps is often depicted by the learning rate (ϵ) parameter, and is usually scheduled to take larger steps in the beginning and smaller steps towards the end of the learning cycle.

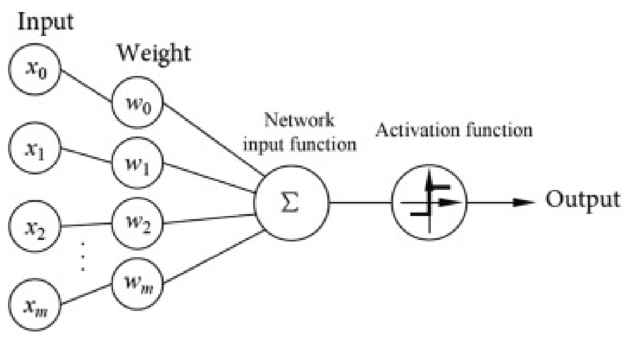


Figure 2.1: The perceptron neural network

Convolutional Neural Networks

Later came convolution neural networks (CNNs). These are neural network like described above except that the weights learned are equal to filter values used for convolutions. CNNs perform very well on images since these convolutions can be used to extract patterns from images. Convolutional Neural Networks are composed of convolutional layers (Figure 2.3).

Convolution

CNNs make use of the convolution operator ($*$). Convolution is defined by sliding a kernel, which can also be seen as a filter, over a matrix of values (i.e pixel intensities). At each step, element wise multiplication is performed and summed.(Figure 2.2).

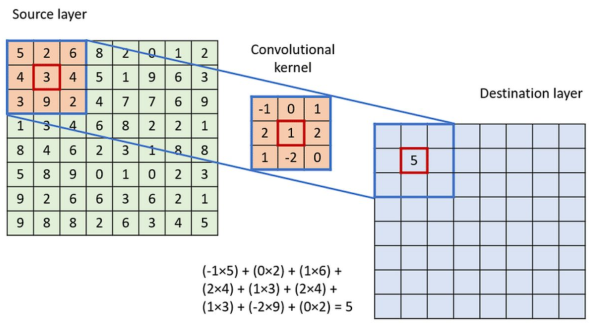


Figure 2.2: An example of the convolution operator

The reason that CNNs are so popular nowadays, is because this operator is optimized by GPU's because of the matrix multiplication technique.

CNN layers are typically arranged sequentially, where early layers capture basic features such as edges and textures, while deeper layers extract increasingly complex patterns and structures. After each convolutional layer, a pooling operation (such as max pooling) is often applied to reduce spatial dimensions, summarizing the most important information and improving computational efficiency. Depending on the task, the CNN ends with one or more fully connected (FC) layers, which transform the extracted features into a final prediction. For classification tasks, the network outputs a set of scores (logits) that are converted into probabilities using a softmax activation function. These probabilities represent the likelihood of the input belonging to each class.

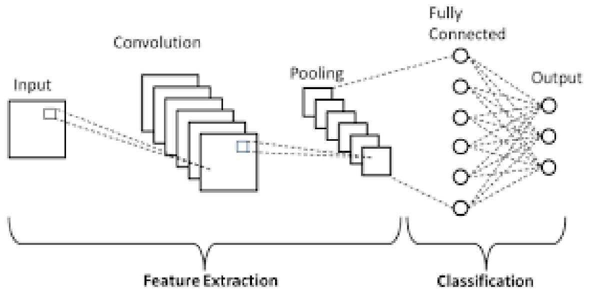


Figure 2.3: An example Convolutional Neural Network (CNN)

2.2 Reinforcement Learning

Reinforcement Learning (RL) [2] is an outlier compared to supervised and unsupervised learning, since its neither supervised or unsupervised. It relies on the ability to monitor the response to the actions of the learning agent. For this matter,it is widely used in Robotics since response to real time data is important in these fields.

To eloborate RL terminology: It consists of an Agent, this can be a car or a robot, which tries to learn a policy from a set of observations during training to solve a given problem (i.e lane following). It can then use this policy to predict an action when another observation comes in. The learning is done through a reward function (where generally a negative rewards means that the action has a negative impact, where a positive reward means it has a positive impact), mathematically speaking:

- S: The state space.
- A: The set of possible actions.
- $P_a(s, s') = P(S_{t+1} = s' \mid S_t = s, A_t = a)$: The transition probability, representing the likelihood of moving from state s to s' under action a at time t.
- $R_a(s, s')$: The reward received when transitioning from state s to s' under action a. This is determined by a reward function.
- $V(s)$: The value function outputting the expected future reward from a given state.
- $Q(a, s)$: The Q function outputting the expected future reward from taking action a in state s.

A basic reinforcement learning agent interacts with its environment in discrete time steps. At each time step t, the agent receives the current state S_t and reward R_t. It then chooses an action A_t from the set of available actions, which is subsequently sent to the environment.

The environment moves to a new state S_{t+1} and the reward R_{t+1} associated with the transition (S_t, A_t, S_{t+1}) is determined. The goal of a reinforcement learning agent is to learn a *policy*:

$$\pi : \mathcal{S} \times \mathcal{A} \to [0,1], \quad \pi(s,a) = \Pr(A_t = a \mid S_t = s)$$

that **maximizes** the expected cumulative reward.

In RL, an important distinction exists between *on-policy* and *off-policy* algorithms. On-policy algorithms require the evaluation or improvement of the policy that is actively collecting data, while off-policy algorithms can learn from data generated by an arbitrary policy. Additionally, the action space in reinforcement learning (RL) can be categorized into discrete and continuous types. In the discrete action space, the model learns a policy that outputs a probability distribution over a finite set of possible actions. Each action is assigned a probability, and the agent selects an action based on this distribution. In the continuous action space, the model outputs a probability distribution over a continuous range of values. An action is then sampled from this distribution, allowing for a more fluid and precise representation of actions.

In Reinforcement Learning, finding a balance between stabilizing a policy and the freedom of exploration is very important. You want a policy to convolve, but also, that it tries to explore new options. This make RL, but also DRL algorithms extremely sensitive to hyperparameters. Reinforcement learning algorithms learn by value, policy, or a combination of both (actor-critic).

Value based

These methods learn a value function, which estimates how good a state (or action) is. They rely on the theory of Markov decision processes, where optimality is defined as: A policy is optimal if it achieves the best-expected discounted return from any initial state. Values are computed using the bellman equation. The Bellman equation for the state-value function $V(s)$ is given by:

$$V^{\pi}(s) = \mathbb{E}\left[r + \gamma V^{\pi}(s') \mid s\right]$$

γ denotes the discount factor. The larger the discout factor, the faster previous rewards degradate over time.

Policy based

These methods learn a policy (a direct mapping from states to actions). This means that the agent learns a function $\pi(a|s)$ that tells it the probability of taking each action in each state. These methods use policy gradients to adjust the policy parameters based on how much reward the agent is getting.

Deep Reinforcement Learning

Deep Reinforcement Leaning (DRL) is similar to reinforcement learning, excepts that it uses a neural network to learn its policy. More specifically, a neural network is used to approximate the value function and CNNs are used for feature extraction.

Policy Gradient Methods

Policy Gradient Methods [3] are a class of reinforcement learning algorithms that directly optimize the policy mapping from states to action probabilities by adjusting its parameters through gradient ascent. Unlike value-based methods that derive a policy from a value function, policy gradient methods represent the policy using a neural network in most cases and improve it by computing gradients of expected reward with respect to those parameters. These gradients are estimated using sampled trajectories and used to make the policy more likely to choose actions that lead to higher rewards. This approach is especially useful for handling high-dimensional or continuous action spaces.

Proximal Policy Optimization (PPO)

PPO [4] teaches an agent to improve its policy gradually, without making huge, risky changes all at once. It does this by comparing the new policy to the old one and limiting how much it can change in a single update. This is done using a “clipping” technique that stops updates if they would push the policy too far from the current one. This helps prevent the agent from getting worse while trying to get better.

PPO works well with deep neural networks, can handle complex environments (like video games or robots), and is one of the most widely used DRL algorithms because it is both simple to implement and very effective. It is also not as sensitive to hyperparameters, compared to other DRL algorithms.

2.3 Image Processing Methods

Cropping

Cropping involves cutting out a portion of an image to retain only the most relevant visual information. This process is commonly used to remove unnecessary background or focus on specific objects or features within the image.

Normalization

Normalization is a crucial preprocessing step in machine learning and artificial intelligence [5]. It involves scaling numerical data to fall within a specific range, typically [0, 1] or [-1, 1]. This is important because many machine learning algorithms perform better and converge faster when input features are on a similar scale.

References

[1] Tishan, “Understanding the Difference Between Supervised and Unsupervised Learning Techniques.” [Online]. Available: https://www.researchgate.net/publication/373979805_Understanding_the_Difference_Between_Supervised_and_Unsupervised_Learning_Techniques

[2] M. Otterlo and M. Wiering, “Reinforcement Learning and Markov Decision Processes,” *Reinforcement Learning: State of the Art*, pp. 3–42, Jan. 2012, iSBN: 978-3-642-27644-6.

[3] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy Gradient Methods for Reinforcement Learning with Function Approximation,” in *Advances in Neural Information Processing Systems*, vol. 12. MIT Press, 1999. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html

[4] B. Padmaja, C. V. K. N. S. N. Moorthy, N. Venkateswarulu, and M. M. Bala, “Exploration of issues, challenges and latest developments in autonomous cars,” *Journal of Big Data*, vol. 10, no. 1, p. 61, May 2023. [Online]. Available: https://doi.org/10.1186/s40537-023-00701-y

[5] J. Yu and K. Spiliopoulos, “Normalization effects on deep neural networks,” Sep. 2022, arXiv:2209.01018 [cs]. [Online]. Available: http://arxiv.org/abs/2209.01018

[6] J. Shlens, “Notes on Kullback-Leibler Divergence and Likelihood,” Apr. 2014, arXiv:1404.2000 [cs]. [Online]. Available: http://arxiv.org/abs/1404.2000

APPENDICES

Appendix A

Supplementary Results

A.1 Result tables

Table A.1: Accuracy with RGB input across different environments, grouped by architecture

Arch	**Domain Rand.**	**Cropping**	**Train Env.**	**CARLA**	**Gym-DT**	**Duckiebot**
256	✓	✓	Duckietown	0%	99%, R=0.74	100%
	✓	✓	CARLA	100%, R=0.93	**75%, R=0.38**	**80%**
	✓	–	CARLA	100%, R=0.93	31%, R=0.52	40%
	–	✓	CARLA	100%, R=0.93	30%, R=0.32	30%
	–	–	CARLA	100%, R=0.94	20%, R=0.46	0%
512	✓	✓	Duckietown	0%	70%, R=0.68	70%
	✓	✓	CARLA	100%, R=0.93	**42%**, R=0.28	**40%**
	✓	–	CARLA	100%, R=0.93	31%, R=0.55	30%
	–	✓	CARLA	100%, R=0.93	20%, R=0.49	50%
	–	–	CARLA	100%, R=0.94	18%, R=0.48	0%

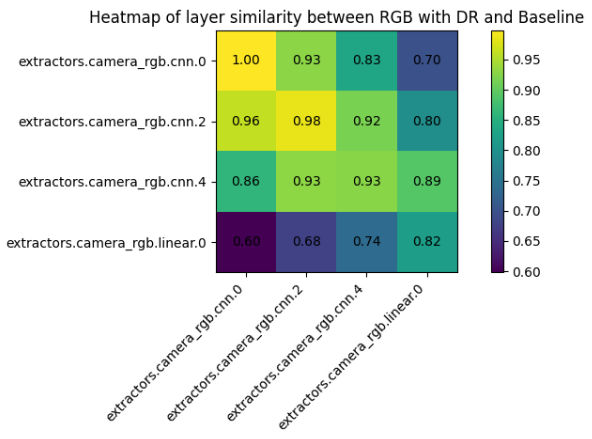
Table A.2: Accuracy with Grayscale input across different environments, grouped by architecture

Arch	Domain Rand.	Cropping	Train Env.	CARLA	Gym-DT	Duckiebot
256	✓	✓	Duckietown	0%	84%, R=0.66	10%
	✓	✓	CARLA	100%, R=0.92	**51%**, R=0.38	10%
	✓	–	CARLA	73%, R=0.93	28%, R=0.41	**20%**
	–	✓	CARLA	100%, R=0.93	0%	0%
	–	–	CARLA	100%, R=0.94	0%	0%
512	✓	✓	Duckietown	0%	49%, R=0.57	20%
	✓	✓	CARLA	74%, R=0.93	0%	0%
	✓	–	CARLA	90%, R=0.93	**25%**, R=0.46	**50%**
	–	✓	CARLA	100%, R=0.93	23%, R=0.49	50%
	–	–	CARLA	100%, R=0.93	18%, R=0.48	0%

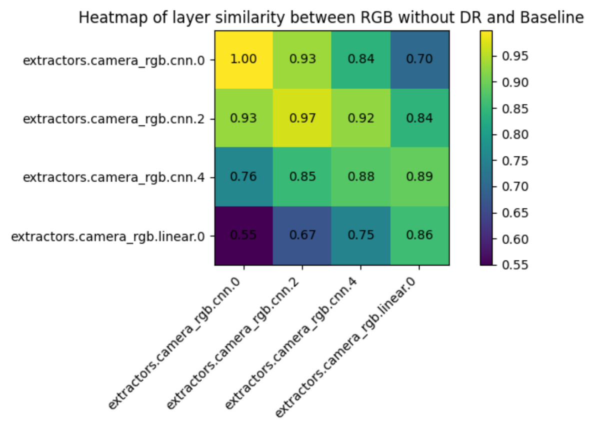
Table A.3: Accuracy with Segmentation input across different environments, grouped by architecture

Arch	Cropping	Train Env.	CARLA	Gym-DT	Duckiebot
256	✓	Duckietown	42%, R=0.88	65%, R=0.49	70%
	✓	CARLA	100%, R=0.93	**61%**, R=0.44	**60%**
	–	CARLA	100%, R=0.93	50%, R=0.43	50%
512	✓	Duckietown	54%, R=0.89	60%, R=0.44	50%
	✓	CARLA	100%, R=0.93	**55%**, R=0.45	**50%**
	–	CARLA	100%, R=0.93	50%, R=0.43	50%

A.2 Heatmaps

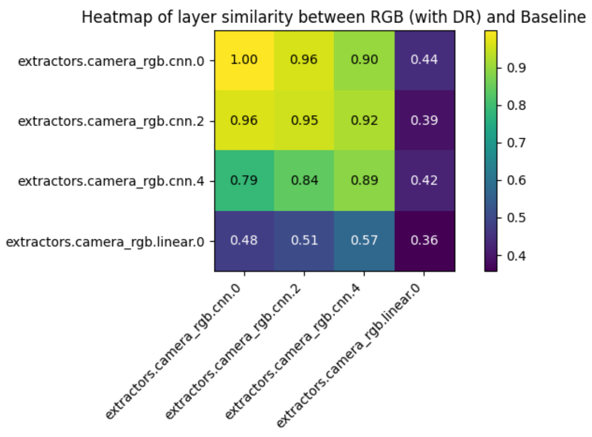


(a) RGB with domain randomization

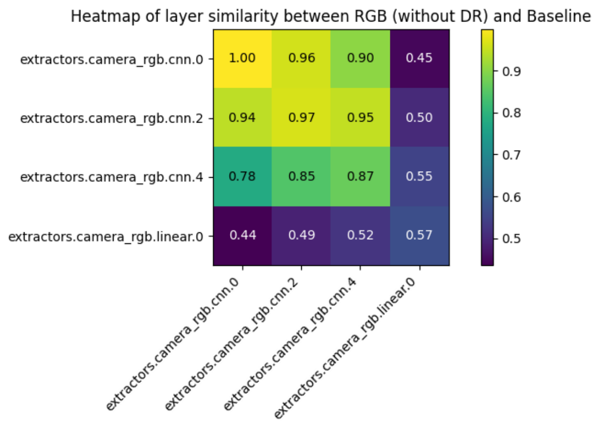


(b) RGB without domain randomization

Figure A.1: Heatmaps comparing feature extraction result similarity in models trained in CARLA with or without DR to a model trained in Gym-DT on a simulated test case in Gym-DT, models are trained using cropped RGB input images.

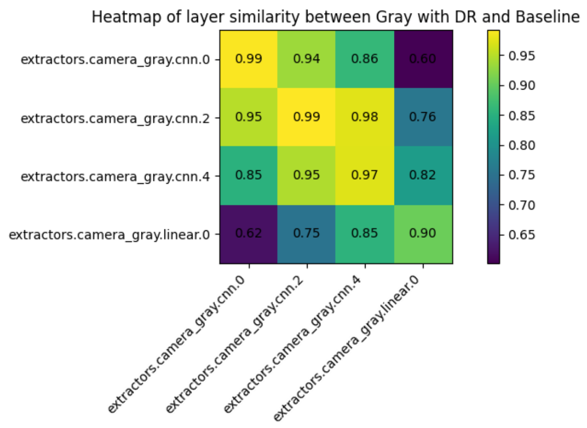


(a) RGB with domain randomization

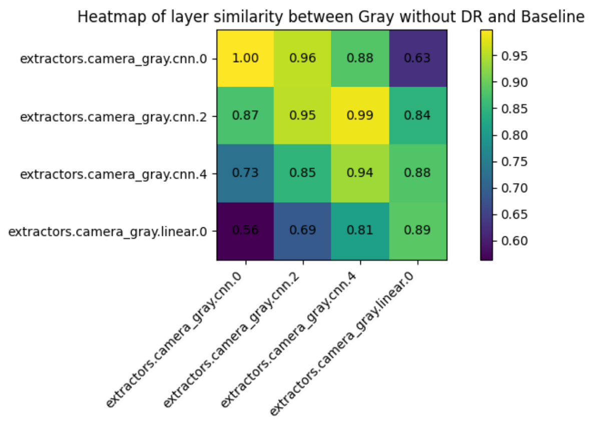


(b) RGB without domain randomization

Figure A.2: Heatmaps comparing feature extraction result similarity in models trained in CARLA with or without DR to a model trained in gym-DT on a real-world testcase (duckiebot), models are trained using cropped RGB input images.

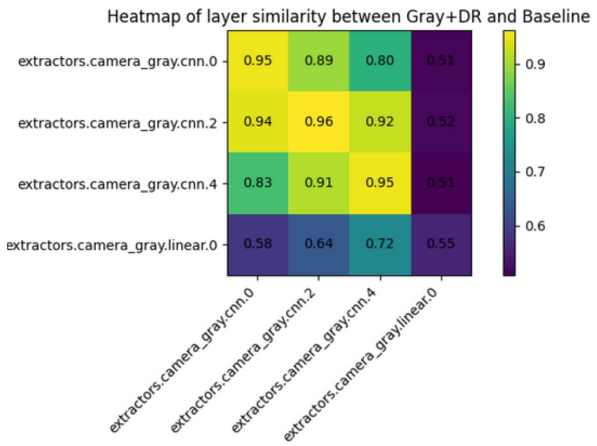


(a) RGB with domain randomization

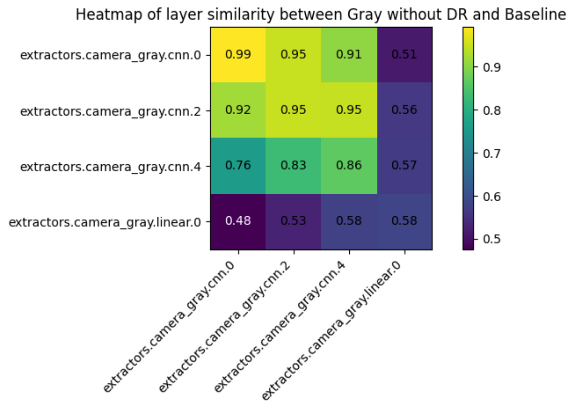


(b) RGB without domain randomization

Figure A.3: Heatmaps comparing feature extraction result similarity in models trained in CARLA with or without DR to a model trained in gym-DT on a simulated test case in Gym-DT, models are trained using cropped Grayscale input images.



(a) RGB with domain randomization



(b) RGB without domain randomization

Figure A.4: Heatmaps comparing feature extraction result similarity in models trained in CARLA with or without DR to a model trained in gym-DT on a real-world test case (duckiebot), models are trained using cropped Grayscale input images.

A.3 Additional Metrics

Next to performance metrics, we have also gathered data on Kullback–Leibler (KL) divergence between action distributions outputted by the models and empirical metrics relating to training times.

KL Divergence in action distributions

Kullback–Leibler (KL) divergence [6] measures the distance between distributions. Since our models output continuous actions, they will output a probability distribution where an action is sampled from. If we compute the average KL divergence over a set of action for two models, we can see how similar the model outputs are.

Below we summarize the KL Divergence between a model trained with domain randomization to a model without and the respective baseline for the RGB, grayscale and segmentation mask input image types. Unfortunately, we could not link KL divergence to model performance

Model1/Model2	No DR	Baseline
DR	6.37	3.3

(a) RGB on gym-dt testcase

Model1/Model2	No DR	Baseline
DR	1.34	1.97

(b) RGB on duckiebot testcase

Model1/Model2	No DR	Baseline
DR	648849	132859

(c) Gray on gym-dt testcase

Model1/Model2	No DR	Baseline
DR	56271	870064

(d) Gray on duckiebot testcase

Figure A.5: KL Divergence between models with and without DR and the respective baseline on both a simulated and real life testcase for RGB & Grayscale image types.

Empirical analysis

We also looked at training times of different image types and preprocessing methods to connect computational demand to performance. We looked at training times in terms of the total timesteps until convergence, meaning the model has learned to perform the task of lane following.

We first compare training times across different image types. Models trained with domain randomization (DR) generally require more timesteps to converge, typically around 1.5 million, compared to approximately 500,000 timesteps for models trained without DR (figure A.6). While DR-trained models consistently achieve better performance in both simulated and real-world settings, segmentation emerges as an exception: segmentation models do not require additional training time and achieve comparable performance to those trained with DR. Model architecture and cropping does not significantly influence this trend.

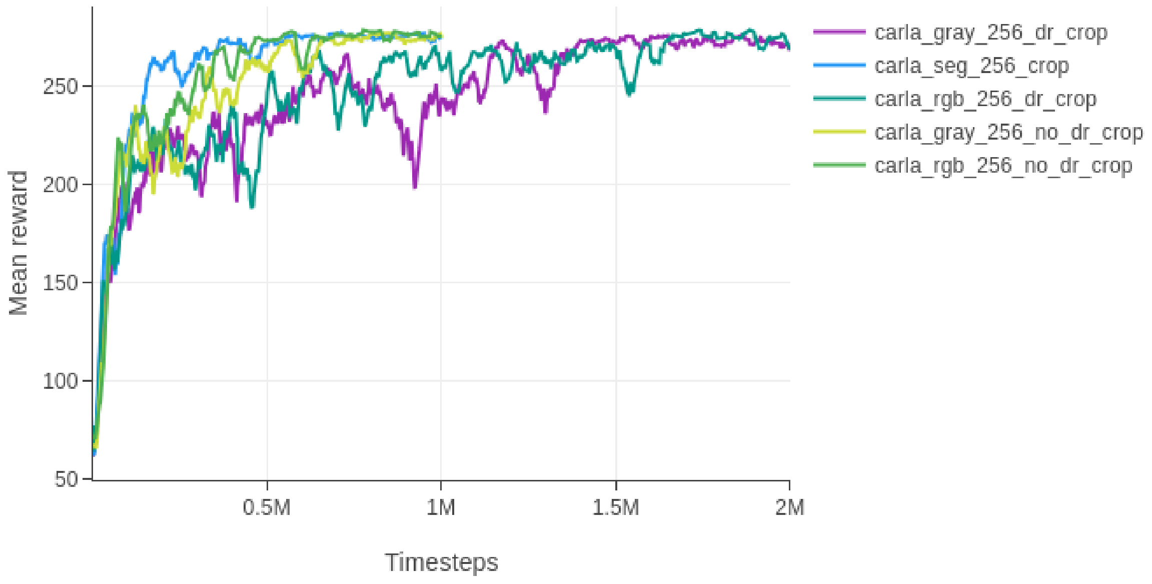


Figure A.6: Overall comparison in training timesteps between models using different image types. All models in this image apply cropping. Model names can be read as {training_env}_{input_type}_{model_architecture}_{dr_applied}

Appendix B

Python Implementation

B.1 Libraries

We used the following libraries:

- PyTorch 1.13.1
- Carla 0.9.15
- duckietown-gym-daffy 6.2.0
- stable-baselines3 2.0.0

B.2 Code

The complete code implementation can be found at https://github.com/Guthax/Sim2Sim2Real

Gymnasium Environments

To run the software, we depend on the following environments:

- CARLA: https://carla.org/
- gym-duckietown: https://github.com/duckietown/gym-duckietown