Enhancing Visual Domain Randomization with Real Images for Sim-to-Real Transfer

András Béres and Bálint Gyires-Tóth

***Abstract*—In order to train reinforcement learning algorithms, a significant amount of experience is required, so it is common practice to train them in simulation, even when they are intended to be applied in the real world. To improve robustness, camera-based agents can be trained using visual domain randomization, which involves changing the visual characteristics of the simulator between training episodes in order to improve their resilience to visual changes in their environment.**

In this work, we propose a method, which includes real-world images alongside visual domain randomization in the reinforcement learning training procedure to further enhance the performance after sim-to-real transfer. We train variational autoencoders using both real and simulated frames, and the representations produced by the encoders are then used to train reinforcement learning agents.

The proposed method is evaluated against a variety of baselines, including direct and indirect visual domain randomization, end-to-end reinforcement learning, and supervised and unsupervised state representation learning.

By controlling a differential drive vehicle using only camera images, the method is tested in the Duckietown self-driving car environment. We demonstrate through our experimental results that our method improves learnt representation effectiveness and robustness by achieving the best performance of all tested methods.

***Index Terms*—Artificial intelligence, Neural networks, Reinforcement learning, Self-driving, Sim-to-real transfer**

I. INTRODUCTION

RECENTLY, reinforcement learning-based algorithms have demonstrated strong capabilities in challenging simulated environments, but real-world applications still pose challenges.

Typically, they require large amounts of experience, which can be obtained by training the agents in a simulator. Since simulators are imperfect and incomplete representations of reality, agents' performance typically decreases when transferred back into the real world. This is especially true for agents using cameras, due to the visual differences between the simulated and real environments.

It is advantageous to use cameras as sensors since they are inexpensive, easy to acquire, and can be used for a wide variety of purposes. They record a large amount of high-dimensional data, but algorithmically it is challenging to extract the relevant high-level information from them.

An agent that uses a camera sensor can be trained in a simulator by rendering an image of a simulated camera and using that as input. The difficulty of transferring to the real world stems from the fact that the diversity of images in a simulator is much lower than in the real world. There is a danger that the model learns some specific properties of the simulator (like the colors and textures of some objects), that will not be the same in reality, or will be much more diverse. In that case since these inputs are different from anything the network has seen, its outputs become unpredictable.

In order to increase the robustness of agents to visual changes in their environment, visual domain randomization can be used, which alters the visual characteristics of the simulator in a randomized way between training episodes.

One possibility to further improve performance after sim-to-real transfer would be to include real images in some way into the training procedure. This has the difficulty that while real world images are usually inexpensive to gather, reinforcement learning also requires corresponding rewards, which are more difficult to gather, since they would require precise estimation of the state, which is more error-prone and also more noisy due to imperfect sensors, compared to simulation.

As a result, in order to overcome this limitation while still retaining the ability to utilize real-world data, we propose using an unsupervised method for state representation learning. Our method does not require either rewards or labels in order to learn state representations. Our proposal is to train variational autoencoders (VAE) on both real and simulated frames, so that the training distribution incorporates real frames. Training reinforcement learning agents in simulators using the representations of these pretrained VAE encoders can be transferred and robustly applied to the real world.

II. RELATED WORK

In order to decrease the gap between the simulation and the real world, a number of techniques can be employed, as follows:

- More realistic simulator environments.
  - More realistic rendering and textures [1].
  - System identification and calibration [2]: more accurate dynamics parameters based on measurements.
  - Novel views of real 3D scenes using Neural Radiance Fields [3][4]
- Domain adaptation [5]: in order to reduce the performance difference between the simulated and real environment, certain statistics can be adjusted to make the

András Béres and Bálint Gyires-Tóth, Department of Networked Systems and Services, Budapest University of Technology and Economics, Budapest, Hungary (E-mail: beres@tmit.bme.hu, toth.b@tmit.bme.hu)

simulated and real environments more similar, auxiliary loss functions can be applied, or transfer learning can be used.

- Domain randomization [6][7]: A random perturbation of some parameters (e.g. visuals) of the simulated environment is performed in each training episode to broaden the range of environments where the agent performs correctly.
- Regularization:
  - Observation-noise [2]: making the agent more resilient to discrepancies in its observations can increase its robustness.
  - Action-noise [2]: can force the agent to plan more robustly or behave more conservatively.
  - Network regularization [8]: application of techniques typically used against overfitting in deep learning, such as L2 regularization [9], droupout [10], and parameter noise [11].

In the following subsections we introduce those methods, that are most relevant to this work.

A. Domain Randomization

In domain randomization, selected parameters of the simulator are randomly perturbed during every training episode. By training reinforcement learning agents in a variety of virtual environments, the range of environments they can perform well will be widened, increasing the likelihood of a successful simulation-to-real transfer.

The two main methods of domain randomization are visual domain randomization [6][7], where visual parameters are perturbed, such as textures, lightning, background, and dynamics randomization [12], where the parameters of the process dynamics are changed.

In the case of visual domain randomization and image observations, one could also use image augmentation methods instead of re-rendering the images. These however should not distort the perceived state of the simulator, which is observed by the agent. In first-person view environments such as car driving, random cropping the image would distort the agent's perception of its own position on the track, so we consider it observation-noise instead of visual domain randomization, this however was also shown recently [13][14] to be effective in regularizing the networks to improve training performance. A simple non-distorting image augmentation example is Gaussian pixel noise, but one could change the brightness, contrast or the saturation of the images as well.

These two methods are quite different, and promote generalization in different aspects. While a large diversity can be helpful in the case of visual domain randomization, it is usually detrimental for dynamics randomization. This means that the randomization ranges are important hyperparameters for both methods. In this work we investigate both of them, and will introduce the most relevant works in the following sections.

Visual domain randomization is commonly used for high-dimensional sensors, primarily camera-based tasks, although it may also be applied to LIDARs.

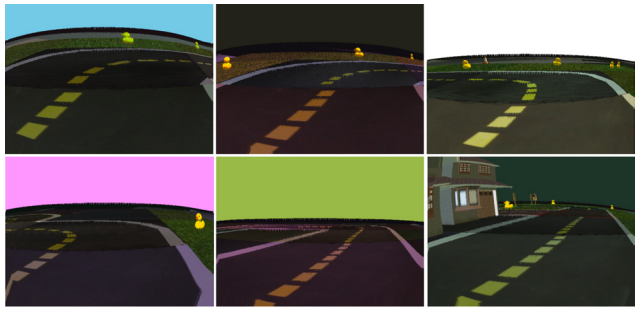

Fig. 1. Example frames with visual domain randomization from the Duckietown Gym self-driving simulated environment.

B. Direct Visual Domain Randomization

This is the most straightforward way to apply visual domain randomization: the randomized environments are used in the same way as the original one – their observations are directly used for training.

One of the first applications of the technique, in which the goal was to ensure generalization by visual diversity and not to make it visually more realistic, was for the task of indoor camera-based drone control [6]. The authors carried out the training in simulated indoor environments, in which they placed lightsources, furniture, closed and open doors in random positions and directions. They also randomly chose realistic wall textures. Though their network was pretrained on realistic images, they did not use any further real images during training, and their algorithm was capable of flying in the reality as well, with approximately one crash every minute.

The technique was also successfully applied in robotics for object localization [7]. The task of the network was to determine the positions of objects on a table, with other distracting objects present, based on camera images. The training was carried out without any real images, with a random amount of objects with randomized shape, texture and position, and with a random amount of lightsources with randomized direction, temperature and position. They also perturbed the position and direction of the camera, and the parameters of noise added to the images. They used multiple thousand non-realistic textures with randomized colors. Based on the ablation study, the randomized texture and camera positions had the highest impact, which is a finding that we have seen in multiple robotics applications.

The method has also been applied to object detection [15] as well, where the authors used it along with a wide range of image augmentation techniques. Findings show that their model has an accuracy similar to as if it has been trained on a highly realistic virtual dataset. In their case the randomized light sources and textures had the greatest impact on the result.

C. Indirect Visual Domain Randomization

Indirect approaches do not use the randomized environments for training, but for network regularization or domain adaptation instead.

Invariance regularized domain randomization [16] uses the following idea: the robustness of a policy can be measured by

calculating the average distance between the policy outputs for a randomized observation and the outputs for its original counterpart. So we can add this as a regularization term to the loss function, and use it for optimization, thereby ensuring robustness of the learned strategies.

A similar solution is the following, where the distances are not calculated between the outputs, but between the activations of the last hidden layers instead [17], which can be seen as a high level representation of the input. This helps to avoid the situation where the two parts of the loss function have opposite effects, therefore in this case increasing the strength of the regularization parameter does not cause a performance drop when comparing with [16], as it is shown in the appendix. Another work proposes this same method [18], however an interesting detail is that they use a randomly initialized convolutional layers for data augmentation.

Another method is to train an autoencoder to reconstruct original observations based on visually randomized observations [19], so that the encoder can be used to compress simulator frames to a representation that is invariant to visual changes of the environment.

Visual domain randomization can also be used for domain adaptation [20]. In this case a network is trained to generate a canonical observation (an observation that is similar to the observations of the original environment) based on the randomized observation. This network can also be used to adapt real observations, which we then train our agent on.

A drawback of all works mentioned above is, that they require *paired* canonical - randomized images, which makes incorporating real frames difficult. Real camera images can be considered as randomized observations, but finding their canonical versions is nontrivial and would require image-to-image translation. Another option [21] is to train a generative network to translate simulated images to the domain of real images instead, and use it during simulator-based training.

D. Sim-to-Real Transfer

It can generally be stated that model-free reinforcement learning algorithms are not using gathered experience efficiently, so they need several interactions with their environment to learn to complete certain tasks. That makes training in the real world slow, and since it also usually needs human supervision, it is generally too expensive and for some applications, such as self-diriving cars, even dangerous to train in the real world. A common solution to this problem is that the agents learn in simulated environments and are then transferred to the real world.

Sim-to-real transfer has already been successfully applied in robot arm manipulation [2], robot locomotion [22] and simple self-driving tasks [23].

However, our simulators can only be imperfect digital twins of reality, so the performance of agents is usually reduced after the transfer.This is called the **sim-to-real gap**, and one has to take it into account if they want to apply agents trained in simulation to the real world.

III. PROPOSED METHOD

Invariance-regularized methods (see Section II) require paired randomized and canonical images, which would require producing semantically equivalent versions of the real images – which is hard to produce. To overcome this difficulty, we chose to apply the method on top of direct domain randomization methods instead.

Since supervised state representation learning methods require labels (e.g. physical quantities) for each image, they would be noisy to measure and in some cases difficult to obtain in the real world. As an alternative, we used unsupervised representation learning, which requires only real images and no labels, so all what is needed is a camera to take photos.

Based on these considerations, we propose a two stage training method of the reinforcement learning agent.

1) In the first stage, direct visual domain randomization with unsupervised state representation learning is used. The visually randomized simulated images are extended with real images in the training dataset. For unsupervised state representation learning, we utilize variational autoencoders [24][25] with calibrated decoders [26].
2) The trained VAE encoder is then applied to encode the observations to its learned latent space, and the reinforcement learning agent is trained in this latent space. I.e. the simulator output frames are encoded by the pretrained VAE encoder, and these encoded observations are processed by the agent to predict the best possible actions.

The method is depicted on Figure 2.

During state representation learning, the goal is to learn representations that are (1) compact and informative about the environment state, making them useful for the control agent, and are (2) robust to visual changes, making the agent transferable into the real world.

The first criterion can be evaluated by training a reinforcement learning agent using the observations that are preprocessed by the VAE encoder with its weights frozen, and measuring the task performance in the simulator. We test the second criterion by transferring agents into the real world, and evaluating their performance.

Variational Autoencoder

For state representation learning, we decided to use variational autoencoders as their utility for learning image representations for reinforcement learning algorithms has been demonstrated in previous works [27], including self-driving, both in simulation [28] and the real world [29].

$$\mathbb{E}_{x\in X}[\mathbb{E}_{z\sim q(z|x)}[-\log p(x|z)] + D_{KL}(q(z|x)||p(z))] \quad (1)$$

Equation 1 shows the objective [30] of variational autoencoders, which can be optimized using the loss function in Equation 2, with X being the set of images in the dataset, x an image sample, $p(z)$ being the prior distribution of the latent variables, $q(z|x)$ being the posterior latent distribution an image is encoded into by the encoder, and $p(x|z)$ being the output distribution of pixel values, a reconstruction of the

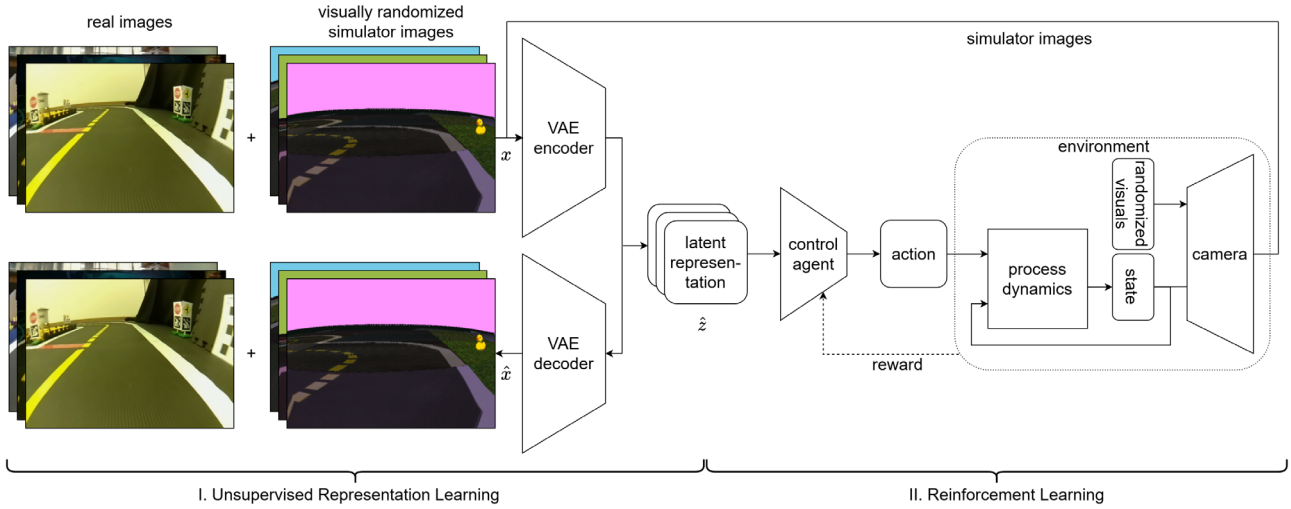

Fig. 2. High level overview of the proposed method. We use real images alongside visually randomized simulator images during unsupervised state representation learning, to improve robustness for sim-to-real transfer. In the first stage we use variational autoencoders for state representation learning, where an encoder is trained to compress the real and simulated images x into latent variables $\hat{z}$ and a decoder is trained to reconstruct them $\hat{x}$. In the second stage a control agent is trained with reinforcement learning and proximal policy optimization in simulator, where the observations are simulated frames preprocessed by the pretrained, frozen encoder network.

original image by the decoder, decoded using a sample z from the posterior latent distribution. In our method X contains both visually randomized simulated and real images.

$$L_{vae} = -\log p(x|z) + D_{KL}(q(z|x)||p(z)) \tag{2}$$

The loss consists of two terms. The negative log-likelihood term measures the reconstruction error between the original images and their reconstructed distributions, and the Kullbach-Leibler divergence term (KL-divergence), which measures the difference between the latent variable distributions the image is encoded into, and their priors.

Sometimes a β hyperparameter [31] is introduced, which scales the KL-divergence term to balance the relative strengths of the two loss terms. To eliminate the need of tuning this additional hyperparameter, we used pixelwise calibrated decoder distributions [26], which eliminate the need for it by scaling the reconstruction term instead, by setting the variance of its distribution based on the variance of the pixel values in the training data. For consistency we estimated the variances of the distributions by iterating over the whole training dataset once, and used it throughout the training, instead of estimating it on each minibatch, as was done in [26].

Following previous works, we use a unit Gaussian as the latent prior, and another Gaussian distribution with diagonal covariance matrix as the latent posterior. Using these assumptions the loss terms can be computed analytically, and the reconstruction loss term can be written as Equation 3, and the KL-divergence term as Equation 4.

$$-\log p(x|z) = \frac{(x-\hat{x})^2}{2\sigma_x^2} + \log \sigma_x + \log \sqrt{2\pi} \tag{3}$$

$$D_{KL}(q(z|x)||p(z)) = \frac{\hat{z}^2 + \hat{\sigma_z}^2}{2} - \log \hat{\sigma_z} - \frac{1}{2} \tag{4}$$

The latent posterior mean $\hat{z}$ and standard deviation $\hat{\sigma_z}$ are vectors predicted by the encoder, while the image posterior mean $\hat{x}$ is predicted by the decoder. The image posterior standard deviation, σ_x, is set beforehand. Removing the constants from the terms, we arrive at our final loss, shown in Equation 5, where N is the minibatch size, D the number of latent dimensions, X the set of images in the batch and Z the set of latent variables.

$$L_{vae} = \frac{1}{2N} \sum_{x \in X} [\frac{(x-\hat{x})^2}{\sigma_x^2} + \frac{1}{D} \sum_{z \in Z} [\hat{z}^2 + \hat{\sigma_z}^2 - \log(\hat{\sigma_z}^2)]] \tag{5}$$

IV. Evaluation

We implemented and evaluated the proposed method in the Duckietown self-driving car environment [32], on the camera-based lane following task. We used the Stable-Baselines3 [33] reinforcement learning library, and the proximal policy optimization (PPO) reinforcement learning algorithm [34] with a continuous action space.

We evaluated a wide range of baselines not using real images. The examined configurations are listed in Table I, and the baselines are shown in Figure 3, the bottom row is our proposed method. We also included end-to-end reinforcement learning as a baseline, without the application of a pretrained image encoder. Our hyperparameters are detailed in Appendix A, loss functions used for baseline methods are detailed in Appendix B and C.

A. The Duckietown Platform

The Duckietown self-driving platform consists of multiple main parts, one of which are the Duckiebots, which are small-sized autonomy-capable vehicles, that are controlled by a Raspberry Pi or an Nvidia Jetson Nano, and are equipped with

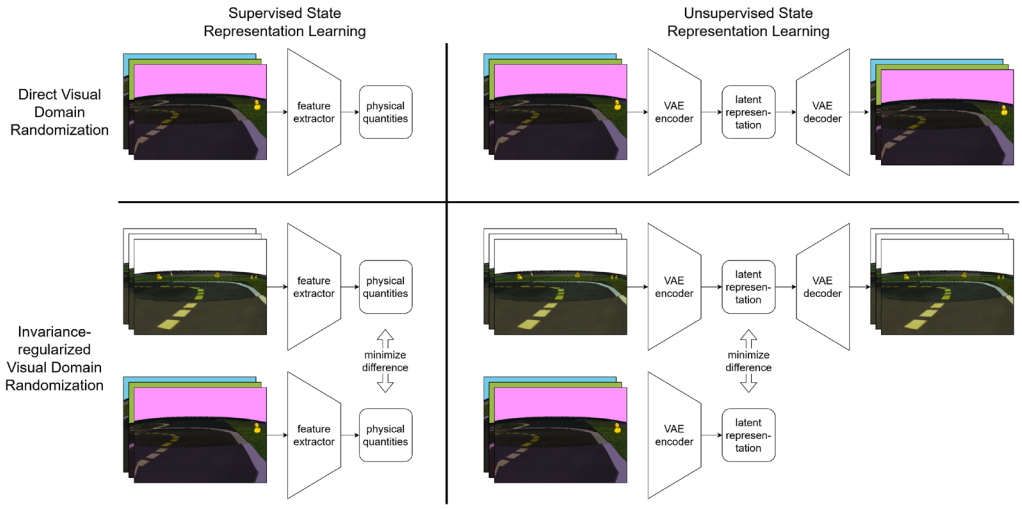

Fig. 3. Benchmarked baseline methods. Direct vs. invariance regularized domain randomization, and supervised vs. self-supervised state representaion learning.

TABLE I
OVERVIEW OF THE BENCHMARKED METHOD CONFIGURATIONS, OUR PROPOSED METHOD IS IN THE BOTTOM ROW IN BOLD.

Domain Randomization	Encoder Pretraining	Canonical Images	Randomized Images	Real Images
None	None	✓	✗	✗
None	Supervised	✓	✗	✗
None	Unsupervised	✓	✗	✗
Invariance-Reg	Supervised	✓	✓	✗
Invariance-Reg	Unsupervised	✓	✓	✗
Direct	None	✗	✓	✗
Direct	Supervised	✗	✓	✗
Direct	Unsupervised	✗	✓	✗
Direct	**Unsupervised**	✗	✓	✓

a single camera. If a Jetson Nano is present, the camera frames can be processed by it on-device, or alternatively they can be streamed over the network to a remote computer, which is common practice in camera-based robotics [35]. During evaluation we applied the latter option for greater throughput. Duckiebots are differential drive vehicles, they do not use a servo motor for steering, instead their motors are independent on their sides, and they can turn by driving their motors at different speeds.

Another part of the system is Duckietown, which is a small scale well-specified, real, physical driving environment, which can be used by the Duckiebots for driving, therefore their performance can be evaluated in a real environment.

The last main part is the Duckietown Gym, which is a self-driving car simulator, implementing the OpenAI Gym interface. The simulator contains multiple maps that provide tasks such as lane following, navigation in intersections, and pedestrian- (duckie-) and vehicle- (duckiebot-) avoidance. The simulator is capable of simulating multiple agents, opening the possibility for analyzing the joint behaviour of multiple traffic participants [36], however in this work we only consider the single-agent setting.

An important feature of the simulator is that it implements

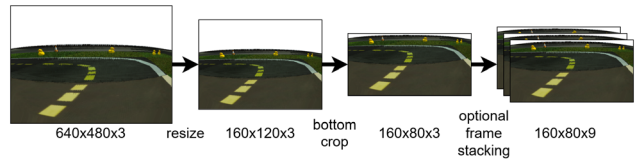

Fig. 4. An illustration of the preprocessing pipeline

visual domain randomization by optionally perturbing the following components:

- Position and color of the light source
- Camera position, angle, and field of view
- Color of the sky
- Texture and color of road tiles
- Amount, type, position and color of environment objects

We perturbed all of these components when training the reinforcement learning agent with visual domain randomization.

B. Observation Space

Following the work of [37], we downscaled the 640x480 input image by a factor of 4 on both sides to a resolution of 160x120, then we cropped out the upper third of the image, which generally only contained information about the background objects and the sky, which yielded an observation of size 160x80.

Theoretically, stacking multiple past frames can be useful, as this enables the network to infer information about its speed and angular velocity (which need at least 2 frames), and its acceleration and angular acceleration (which need at least 3 frames). These theoretical considerations have been reinforced by prior work [37], and it has also been experimentally shown that stacking more than 3 frames does not yield considerable benefits, therefore we stacked 3 past frames together for every observation for the RL agents when not using rotary encoders. Since we used colored images, the final size of the input image

observations became 160x80x9, as shown in Figure 4. We did not apply any other preprocessing on the input images, such as tresholding certain colors or filtering.

The alternative solution for sensing speed and angular velocity is to use the rotary encoders that the latest edition of Duckiebots are equipped with. This option brings its own set of challenges, and we leave exploring this option to future work.

C. Action Space

A differential robot is usually controlled by driving its motors on its sides at different speeds. In the case of the Duckiebot, one can control the duty cycles of the pulse width modulated (PWM) signals that drive its DC motors.

That means that the space of possible actions is two-dimensional and by each dimension it spans the range of $[-1.0; 1.0]$. This action space permits some actions that are not useful, for example we do not want the vehicle to drive backwards or to drive it much more slowly than what it is capable of.

Since the task of vehicle control and lane following is inherently continuous, we have chosen to use a continuous action space. We followed prior work [37], and have defined a 1-dimensional action space. The only thing the agent can directly influence is its steering angle, which is then mapped to two target speeds of its two motors. These speeds are chosen to be as high as possible while still having a difference that is proportional to the steering angle. This has the effect that when taking sharp turns the car has to slow down to be able to provide the needed difference between the wheel speeds.

$$u_{avg} = \min(u_{nom}, \frac{1}{1+|\phi|}) \tag{6}$$

$$u_{left} = clip(u_{avg}(1+\phi), -1, 1) \tag{7}$$

$$u_{right} = clip(u_{avg}(1-\phi), -1, 1) \tag{8}$$

The exact derivation is described in Equation 8, where u_{nom} is the desired maximal duty cycle (nominal duty cycle), u_{avg} is the average duty cycle of the two motors (this depends on the desired steering angle), u_{left} and u_{right} are the duty cycles of the corresponding motors, and ϕ is the desired steering angle, while $clip(value, min, max)$ is a function that clips its input to be between a minimal and maximal value.

For small values ϕ can actually be interpreted as a steering angle in radians, however for larger values it should be interpreted as a scalar value that is proportional to the angular velocity of the vehicle, with $|\phi| = 1$ meaning that either one of the motors stops completely while the other one runs at full speed, meaning that the vehicle goes at half of its maximal speed.

D. Reward Function

Reward functions for lane following may be based on throttle [28], speed [38], speed parallel to lane [39][40][41][42][29], traveled distance [43], and progress [38], and can include penalties for leaving the lane [28], distance from lane center [42][40][41], or collision [43][41].

Our initial experiments showed that with the default reward function provided by the Duckietown environment, the agents perform suboptimally. Though multiple different reward functions have been tested in previous work [44][45], it has been shown, that a speed-based reward function is already a strong baseline [45]. Based on these results, we have chosen to use a reward function that is physically motivated and is speed-based. In each timestep the reward of the agent is the speed at which it is progressing in its lane (with which speed it is completing the track). A more accurate description is that the reward is the speed of a virtual twin vehicle that moves exactly in the middle of the lane, is exactly parallel to it, and completes its route at the same rate as the actual car.

The exact formula of the reward function is shown in Equation 9, where v is the physical speed of the car, δ is the signed angle of the car and lane, r is the signed radius of the turn, c is the signed curvature of the turn ($c = 1/r$), and p is the signed distance of the car from the lane.

$$R = v_{progress} = v \cdot cos(\delta) \cdot \frac{r}{r+p} = v \cdot cos(\delta) \cdot \frac{1}{1+cp} \tag{9}$$

The formula can be understood in the following way: $v \cdot cos(\delta)$ is the component of the vehicle's speed that is parallel to the lane, $v \cdot cos(\delta)/(r+p)$ is the angular velocity of the vehicle in a turn, and $v \cdot cos(\delta)/(r+p) \cdot r$ is the circumferential velocity of the equivalent virtual vehicle in the turn, that is moving exactly on the middle of the lane.

This reward function has the advantage that it penalizes high angles and turns taken in the outer regions of the road, while it promotes high speeds and turns taken on the inner regions of the road. In practice the value is generally quite close to simply the speed of the vehicle, which has also been shown to be a plausible reward function [37].

When using this reward function however, care has to be taken to limit how much the car can leave its lane, otherwise it will tend to take left turns by going over to the other lane.

E. Dataset Collection

To be able to apply state representation learning efficiently, we generated a dataset of observations and corresponding physical parameters. Though one could use a streaming-type dataset, which is generated by the simulator, this would not only bottleneck the training speed, but the samples would not be statistically independent and identically distributed.

Based on these considerations, we generated and saved images of 200.000 scenes during the training of a baseline end-to-end convolutional reinforcement learning agent, and stored 3 different renderings for each image: a visually randomized, a canonical (non-randomized), and a segmented one, as shown in Figure 5. We also saved the corresponding speeds, angular velocities, lane angles, lane distances, and lane curvatures for future work. The generated dataset has a 22.6 GB storage size.

Four different maps were used, all being a part of the official Duckietown simulator (Duckietown Gym [32]): *4way, loop_empty, udem1 and zigzag_dists*.

Note that there are intersections on two out of these four maps, which we included on one hand to increase diversity, but also to help future efforts dealing with intersections.

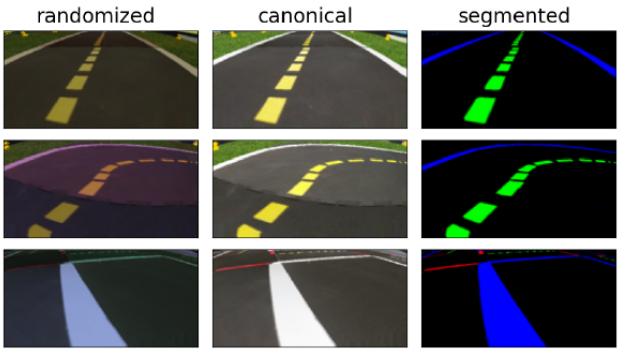

Fig. 5. Samples from the gathered offline dataset showing the 3 different renderings for each scene.

Using the dataset, the mean and variance observations for each image type have been determined on the training set before the training, to enable the usage of calibrated decoders [26].

We have created another dataset as well, which contains 19.000 real images, downloaded from online logs of Duckiebots from the Duckietown website [32]. We handpicked 3 videos of agents with reasonable performance and diverse lighting conditions, extracted and saved the frames from them. These images have been saved under the randomized images category, with no corresponding canonical or segmented frames, nor physical quantities.

F. Evaluation Metrics

We used the following metrics to monitor the performance of lane following agents during and after the training in simulation:

- Mean angle error: The mean absolute angle between vehicle and the lane
- Mean position error: The mean absolute distance between vehicle and the center of the lane
- Mean completion speed: the same as the mean reward per timestep
- The average length of the evaluation episodes in timesteps, which can be used to calculate the average completion rate, which is the ratio of average evaluation episode length and the maximal episode length

The averages have been calculated over the whole process of the evaluation, which generally consisted of multiple episodes: 10 during training, 50 during final evaluation.

During real world testing, these metrics would be difficult and error-prone to measure, so we used the following evaluation metrics:

- Mean number of traveled tiles: correlates with the distance traveled
- Mean survival time: the ratio of the evaluation time and the number of manual resets required to keep the agent on track

The real world metrics were measured for each algorithm for 60-60 seconds, on both the outer lane and the inner lane of a closed Duckietown track without intersections, with manual resets to the center of the lane if an agent attempted to leave its lane.

Note that the same metrics are used by the Duckietown AI Driving Olympics evaluation [46], but we report mean values instead of medians, which was more practical to measure manually real-time during real world evaluation.

TABLE II
MEAN EVALUATION METRICS OF THE VAE-BASED STATE REPRESENTATION LEARNING METHODS. THE PROPOSED METHOD IS IN THE BOTTOM ROW IN BOLD.

Algorithm	Reconstr. error [nats/dim]	KL-div. [nats/dim]	Inv. KL-div. [nats/dim]
VAE	0.125	5.5	-
VAE inv. reg.	0.126	3.9	8.3
VAE direct	0.160	5.4	-
VAE direct real	**0.087**	**4.9**	**-**

V. RESULTS

In this section we introduce the results of the first stage, and of the second stage (in simulation and real environments).

A. Representation Learning (first stage)

Though our main goal was to evaluate the quality and robustness of the learned state representations by training reinforcement learning agents using the pretrained encoders, we also report the pretraining performance metrics of our method and the other baselines.

The metrics of variational-autoencoder based methods are shown on Table II.

Overall the evaluation metrics are in a similar range across the methods, with the exception of the reconstruction error of our proposed method, which is noticeably lower than its counterparts. This could be caused by the fact that both real and simulated images are used for its training, so the distribution of training images is bimodal, while the loss parametrization only assumed a unimodal Gaussian pixel output distribution. This could have led to overestimated output image pixel variance values, making the loss reconstruction error lower under our imperfect assumption.

However our main goal was not to optimize the pretraining metrics, but to evaluate the quality and robustness of the learnt representations by using them for reinforcement learning (in the second stage), so this is what we present in the following sections.

B. Simulation Environment (second stage)

Table III and IV shows the results of state representation learning algorithms with their default control modules (trained using their representations of their input image type in the Duckietown Gym), evaluated in simulation, evaluated either without or with domain randomization.

Our results in the simulator without domain randomization (Table III) show that from the algorithms that did not use canonical (non-randomized) images for their training (see Table I), our proposed method performs the best. The usage

TABLE III
EVALUATION RESULTS IN THE SIMULATOR WITHOUT VISUAL DOMAIN RANDOMIZATION. COMPLETION RATIOS ABOVE 70% ARE UNDERLINED. THE PROPOSED METHOD IS IN THE BOTTOM ROW IN BOLD. SEE THE CORRESPONDING ROWS OF TABLE I FOR MORE DETAILS ON THE ALGORITHMS.

Algorithm	Abs. angle error [deg]	Abs. pos. error [cm]	Completion speed [m/s]	Completion ratio [%]
E2E	11.5	3.9	0.40	30.0
SUP	6.0	2.5	0.47	88.6
VAE	6.4	2.5	0.48	54.4
SUP inv. reg.	6.6	2.0	0.48	90.8
VAE inv. reg.	7.8	2.5	0.51	83.4
E2E direct	12.9	3.2	0.35	29.3
SUP direct	9.0	3.4	0.44	39.2
VAE direct	8.7	2.4	0.46	52.9
VAE dir. real	**7.6**	**2.8**	**0.47**	**73.9**

TABLE IV
EVALUATION RESULTS IN THE SIMULATOR WITH VISUAL DOMAIN RANDOMIZATION. COMPLETION RATIOS ABOVE 70% ARE UNDERLINED. THE PROPOSED METHOD IS IN THE BOTTOM ROW IN BOLD. SEE THE CORRESPONDING ROWS OF TABLE I FOR MORE DETAILS ON THE ALGORITHMS.

Algorithm	Abs. angle error [deg]	Abs. pos. error [cm]	Completion speed [m/s]	Completion ratio [%]
E2E	10.1	3.8	0.42	23.0
SUP	17.8	4.0	0.28	12.5
VAE	11.0	3.9	0.41	18.2
SUP inv. reg.	6.6	2.0	0.48	84.0
VAE inv. reg.	7.7	2.5	0.50	82.3
E2E direct	12.4	3.3	0.38	27.0
SUP direct	7.3	2.8	0.47	79.9
VAE direct	6.9	2.9	0.49	82.7
VAE dir. real	**6.0**	**2.5**	**0.50**	**88.1**

of real images might have enabled the network to generalize better to canonical images. However the two invariance-regularized methods do show a higher performance in this evaluation setting. The reason can be that the visual domain randomization in the Duckietown Gym might be too extreme, visual inspection of the images show that the images are usually much darker in comparison to the canonical images. Further evaluation will show however (Table IV, Table V) that they have worse generalization compared to our proposed method in the simulated environment with domain randomization, and also in reality.

End-to-end reinforcement learning seems to have underperformed in these experiments. As one can see on tables III, IV and V, end-to-end reinforcement learning did not manage to perform well in any of the evaluation settings.

Table IV shows our evaluation results in the simulator, with domain randomization. They show that all investigated state representation learning methods that are trained using some sort of visual domain randomization, direct or invariance regularized, are capable of solving the lane following task in the simulator with randomized visuals at near 80% or higher completion ratio. However our proposed method (bottom row) outperformed all of them, achieving the highest, 88% completion ratio.

TABLE V
EVALUATION RESULTS IN REALITY. SURVIVAL TIMES GREATER THAN OR EQUAL TO 20 ARE UNDERLINED. THE PROPOSED METHOD IS IN THE BOTTOM ROW IN BOLD. SEE THE CORRESPONDING ROWS OF TABLE I FOR MORE DETAILS ON THE ALGORITHMS.

Algorithm	Traveled tiles (outer)	Traveled tiles (inner)	Surv. time (outer) [s]	Surv. time (inner) [s]
E2E	30	30	5.5	4.6
SUP	0	0	0.0	0.0
VAE	22	26	5.0	4.6
SUP inv. reg.	41	44	60.0	30.0
VAE inv. reg.	43	48	no resets	no resets
E2E direct	30	29	7.5	6.7
SUP direct	35	41	10.0	20.0
VAE direct	40	49	30.0	no resets
VAE dir. real	**44**	**51**	**no resets**	**no resets**

C. Real Environment (second stage)

Table V shows the results of state representation learning algorithms evaluated in the real world. Our proposed method was able to achieve 60 seconds of driving in both directions without any resets, while also covering more distance than any other method. Since all methods were run at the same speeds (same DC motor pulse-width-modulation duty cycle of 50%), more traveled distance signals a smoother driving with fewer oscillations, which lines up with what we saw visually.

Our experiments also showed that none of the models trained without visual domain randomization (top three rows in Table V) were able to have good performance during real testing. On the other hand, from the six models trained using direct or invariance regularizing visual domain randomization, four were able to achieve an average survival time of 30 seconds or above.

The trends are that supervised state representation learning outperforms end-to-end reinforcement learning, but variational autoencoder-based unsupervised state representations perform the best. In terms of domain randomization, direct methods outperform ones in which it was not used at all, but invariance regularizing methods perform the best, with the exception being our proposed method, which used direct visual domain randomization, but also used real images for training.

We also took part with preliminary versions of the proposed method in the Urban League of the 5th and 6th editions of the AI Driving Olympics [47], and achieved first prize in the Lane Following category in the 5th edition [48], and achieved third and fourth place in the Lane Following with Intersections and with Vehicles categories respectively in the 6th edition [49].

VI. CONCLUSION

In this work we proposed a novel method for learning effective image representations for reinforcement learning, whose core idea is to train a variational autoencoder using visually randomized images from the simulator, but include images from the real world as well, as if it was just another visually different version of the simulator.

We evaluated the method in the Duckietown self-driving environment on the lane-following task, and our experimental results showed that the image representations of our proposed method improved the performance of the tested reinforcement

learning agents both in simulation and reality. This demonstrates the effectiveness and robustness of the representations learned by the proposed method.

We benchmarked our method against a wide range of baselines, and the proposed method performed among the best in all cases. Our experiments showed that using some type of visual domain randomization is necessary for a successful sim-to-real transfer. Variational autoencoder-based representations tended to outperform supervised representations, and both outperformed representations learned during end-to-end reinforcement learning. Also, for visual domain randomization, when using no real images, invariance regularization-based methods seemed to outperform direct methods.

Based on our results, we conclude that including real images in simulation-based reinforcement learning trainings is able to enhance the real world performance of the agent – when using the two-stage approach, proposed in this paper.

APPENDIX A
HYPERPARAMETERS

Tables VI and VII show the hyperparameters used for pretraining and finetuning, while tables VIII and VIII show the used encoder and decoder architectures. Note that the neural network architectures were chosen to be able to fit into the embedded hardware at a limited runtime.

TABLE VI
VARIATIONAL AUTOENCODER HYPERPARAMETERS

Name	Value
Learning rate	1e-3
Number of epochs	20
Width	64
Number of latent dimensions	8
σ parametrization	$\log \sigma$

TABLE VII
PPO REINFORCEMENT LEARNING ALGORITHM HYPERPARAMETERS

Name	Value
Optimization steps	64 * 2048
Learning rate	3e-4
Number of steps between updates	2048
Batch size	64
Optimization epochs	10
Time horizon (discount factor)	0.8s (0.96)
Gradient clip range	0.2
Entropy coefficient	0.0
Initial log standard deviation	-1.2

TABLE VIII
ENCODER ARCHITECTURE. ALL CONVOLUTIONS USED A KERNEL SIZE OF 3, STRIDE OF 2, LEFT-RIGHT ZERO PADDING OF 1.

Layer	Activation	Output dimensions
Conv	ReLU	width x 40 x 80
Conv	ReLU	width x 20 x 40
Conv	ReLU	width x 10 x 20
Conv	ReLU	width x 5 x 10
Linear		2 x latent dim

APPENDIX B
BASELINES WITH SUPERVISED FEATURE EXTRACTION

The loss functions of benchmarked baselines (shown in Figure 3) using supervised image encoders were determined following a similar logic to Section III, to have a fair comparison.

Since the scale of multiple physical outputs that we predict can be very different, we applied a method which works out-of-the box, and doesn't require further hyperparameter tuning. The squared errors of all the physical quantities are normalized by their variance as shown in Equation 10, which is calculated beforehand on the training data.

$$-\log p(y|x) = \frac{(y-\hat{y})^2}{2\sigma^2} + \log\sigma + \log\sqrt{2\pi} \qquad (10)$$

y is the ground truth label (vector of physical quantities to be estimated), $\hat{y}$ is produced by the image encoder, and σ is the standard deviation of the output distribution. Removing the constants we arrive at the mean squared error loss, up to a constant scaling factor, shown in Equation 11, with X and Y being the set of images and corresponding labels in a batch.

$$L_{sup} = \mathbb{E}_{x,y\in X,Y}[-\log p(y|x)] = \frac{1}{2N}\sum_{y\in Y}(y-\hat{y})^2 \qquad (11)$$

APPENDIX C
BASELINES WITH INVARIANCE REGULARIZATION

For those benchmarked baselines that use invariance regularization-based visual domain randomization (shown in Figure 3), the KL-divergences were calculated between the decoder output distributions that were produced using a canonical and a randomized image.

These KL-divergences are added as auxiliary losses without reweighting to the training loss, which is motivated by a related work [17] (Appendix B), which shows that the network's performance on the reference domain does not depend heavily on the weight of the invariance loss, if it is not the controller's output which is regularized, but an earlier layer. This is true in our case, since reinforcement learning agents (parametrized by multilayer perceptrons) are trained later on top of the image encoders.

For the variational autoencoder, the auxiliary loss is shown in Equation 12.

$$KL(q(z|x_r)||q(z|x)) = \frac{(\hat{z}_r-\hat{z})^2 + \hat{\sigma_r}^2}{2\hat{\sigma}^2} + \log\frac{\hat{\sigma}}{\hat{\sigma_r}} - \frac{1}{2} \qquad (12)$$

TABLE IX
DECODER ARCHITECTURE. ALL CONVOLUTIONS USED A KERNEL SIZE OF 3, STRIDE OF 1, LEFT-RIGHT ZERO PADDING OF 1. ALL UPSAMPLING LAYERS USED NEAREST NEIGHBOR UPSAMPLING WITH A SCALE FACTOR OF 2.

Layer	Activation	Output dimensions
Linear	ReLU	width x 5 x 10
Upsampling + Conv	ReLU	width x 10 x 20
Upsampling + Conv	ReLU	width x 20 x 40
Upsampling + Conv	ReLU	width x 40 x 80
Upsampling + Conv	Sigmoid	width x 80 x 160

The supervised case is shown in Equation 13. Using that visual domain randomization does not change the underlying physical quantities, and the output variance are predefined, the loss can be further simplified to a scaled mean-squared-error loss, shown in Equation 14.

$$KL(p(y|x_r)||p(y|x)) = \frac{(\hat{y}_r - \hat{y})^2 + \sigma_r^2}{2\sigma^2} + \log\frac{\sigma}{\sigma_r} - \frac{1}{2} \quad (13)$$

$$KL(p(y|x_r)||p(y|x)) = \frac{(\hat{y}_r - \hat{y})^2}{2\sigma^2} \quad (14)$$

In the above auxiliary loss equations x_r is the second input image, in our case the visually randomized one, which we want to be invariant to. $\hat{y}_r$ is produced by the image encoder and $\hat{\sigma_r}$ is the standard deviation for the encoder output distribution based on the randomized image.

With that, we get a similar loss formulation to the work on VAEs with consistency regularization [50], if the regularization strength hyperparameter is set to 1.0, but without the task of reconstructing the invariance-input images.

ACKNOWLEDGMENT

The research presented in this work has been supported by the PIA Project, a collaboration between Budapest University of Technology and Economics and Continental Hungary Ltd with the goal of supporting students' research in the field of deep learning and autonomous driving.

The work reported in this paper, carried out at BME, has been partly supported by the the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory.

REFERENCES

[1] S. James and E. Johns, "3d simulation for robot arm control with deep q-learning," *arXiv preprint arXiv:1609.03759*, 2016. DOI: 10.48550/arXiv.1609.03759

[2] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray et al., "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020. DOI: 10.1177/0278364919887447

[3] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021. DOI: 10.1145/3503250

[4] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics (ToG)*, vol. 41, no. 4, pp. 1–15, 2022. DOI: 10.1145/3528223.3530127

[5] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, pp. 135–153, 2018. DOI: 10.1016/j.neucom.2018.05.083

[6] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," in *Proceedings of Robotics: Science and Systems*, 2017. DOI: 10.15607/RSS.2017.XIII.034

[7] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133

[8] Z. Liu, X. Li, B. Kang, and T. Darrell, "Regularization matters in policy optimization-an empirical study on continuous control," in *International Conference on Learning Representations*, 2020. DOI: 10.48550/arXiv.1910.09191

[9] A. Y. Ng, "Feature selection, l1 vs. l2 regularization, and rotational invariance," in *Proceedings of the 21st International Conference on Machine Learning*. Association for Computing Machinery, 2004, p. 78. DOI: 10.1145/1015330.1015435

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from over-fitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. DOI: 10.5555/2627435.2670313

[11] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," in *International Conference on Learning Representations*, 2018. DOI: 10.48550/arXiv.1706.01905

[12] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8. DOI: 10.1109/ICRA.2018.8460528

[13] D. Yarats, I. Kostrikov, and R. Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," in *International Conference on Learning Representations*, 2021. DOI: 10.48550/arXiv.2004.13649

[14] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *Advances in Neural Information Processing Systems*, vol. 33, 2020. DOI: 10.48550/arXiv.2004.14990

[15] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 969–977. DOI: 10.1109/CVPRW.2018.00143

[16] M. Aractingi, C. Dance, J. Perez, and T. Silander, "Improving the generalization of visual navigation policies using invariance regularization," *36th International Conference on Machine Learning, Workshop RL4RealLife*, 2019.

[17] R. B. Slaoui, W. R. Clements, J. N. Foerster, and S. Toth, "Robust domain randomization for reinforcement learning," *arXiv preprint arXiv:1910.10537*, 2019.

[18] K. Lee, K. Lee, J. Shin, and H. Lee, "Network randomization: A simple technique for generalization in deep reinforcement learning," in *8th International Conference on Learning Representations*, 2020. DOI: 10.48550/arXiv.1910.05396

[19] A. Amiranashvili, M. Argus, L. Hermann, W. Burgard, and T. Brox, "Pre-training of deep rl agents for improved learning under do- main randomization," *arXiv preprint arXiv:2104.14386*, 2021. DOI: 10.48550/arXiv.2104.14386

[20] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. DOI: 10.1109/CVPR.2019.01291

[21] M. Tim, M. Szemenyei, and R. Moni, "Simulation to real domain adaptation for lane segmentation," in *2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR)*, 2020, pp. 1–6. DOI: 10.1109/ISMCR51255.2020.9263406

[22] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018. DOI: 10.15607/RSS.2018.XIV.010

[23] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall, "Learning to drive from simulation without real world labels," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 4818–4824. DOI: 10.1109/ICRA.2019.8793668

[24] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013. DOI: 10.48550/arXiv.1312.6114

[25] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 32, no. 2. PMLR, 2014, pp. 1278–1286. DOI: 10.5555/3044805.3045035

[26] O. Rybkin, K. Daniilidis, and S. Levine, "Simple and effective vae training with calibrated decoders," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 139. PMLR, 2021, pp. 9179–9189. DOI: 10.48550/arXiv.2006.13202

[27] D. Ha and J. Schmidhuber, "World models," in *Advances in Neural Information Processing Systems*, vol. 31, 2018. **DOI**: 10.5281/zenodo.1207631

[28] B. Prakash, M. Horton, N. R. Waytowich, W. D. Hairston, T. Oates, and T. Mohsenin, "On the use of deep autoencoders for efficient embedded reinforcement learning," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 507–512. **DOI**: 10.1145/3299874.3319493

[29] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254. **DOI**: 10.1109/ICRA.2019.8793742

[30] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016. **DOI**: 10.48550/arXiv.1606.05908

[31] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," *International Conference on Learning Representations*, 2017.

[32] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta, and L. Paull, "Duckietown environments for openai gym," https://github.com/duckietown/gym-duckietown, 2018.

[33] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," https://github.com/DLR-RM/stable-baselines3, 2019.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. **DOI**: 10.48550/arXiv.1707.06347

[35] M. Balogh and A. Vidács, "Optimizing camera stream transport in cloud-based industrial robotic systems," *Infocommunications Journal*, vol. XIV, no. 1, pp. 36–42, March 2022. **DOI**: 10.36244/ICJ.2022.1.5

[36] G. Hollósi, C. Lukovszki, M. Bancsics, and G. Magyar, "Traffic swarm behaviour: Machine learning and game theory in behaviour analysis," *Infocommunications Journal*, vol. XIII, no. 4, pp. 19–27, December 2021. **DOI**: 10.36244/ICJ.2021.4.3

[37] A. Kalapos, "Applying transfer learning to autonomous driving task," Master's thesis, Budapest University of Technology and Economics, 2020.

[38] A. Santara, S. Rudra, S. A. Buridi, M. Kaushik, A. Naik, B. Kaul, and B. Ravindran, "MADRaS: Multi agent driving simulator," *Journal of Artificial Intelligence Research*, vol. 70, pp. 1517–1555, apr 2021. **DOI**: 10.1613/jair.1.12531

[39] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lil- licrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 1928–1937. **DOI**: 10.48550/arXiv.1602.01783

[40] E. Perot, M. Jaritz, M. Toromanoff, and R. de Charette, "End-to-end driving in a realistic racing game with deep reinforcement learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017. **DOI**: 10.1109/CVPRW.2017.64

[41] Z. W. Xinlei Pan, Yurong You and C. Lu, "Virtual to real reinforcement learning for autonomous driving," in *Proceedings of the British Machine Vision Conference (BMVC)*, G. B. Tae-Kyun Kim, Stefanos Zafeiriou and K. Mikolajczyk, Eds. BMVA Press, September 2017, pp. 11.1– 11.13. **DOI**: 10.5244/C.31.11. ISBN 1-901725-60-X

[42] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2070–2075, 2018. **DOI**: 10.1109/ICRA.2018.8460934

[43] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16. **DOI**: 10.48550/arXiv.1711.03938

[44] P. Almasi, R. Moni, and B. Gyires-Toth, "Robust reinforcement learning-based autonomous driving agent for simulation and real world," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8. **DOI**: 10.1109/IJCNN48605.2020.9207497

[45] A. Kalapos, C. Gór, R. Moni, and I. Harmati, "Sim-to-realreinforcement learning applied to end-to-end vehicle control," in *23rd International Symposium on Measurement and Control in Robotics (ISMCR)*, 2020, pp. 1–6. **DOI**: 10.1109/ISMCR51255.2020.9263751

[46] Duckietown Foundation. "AI Driving Olympics Performance Metrics". [Online]. Available: https://docs.duckietown.org/daffy/AIDO/out/measuring_performance.html

[47] ——. Ai driving olympics. [Online]. Available: https://driving-olympics.ai/

[48] ——. Ai driving olympics 5: Urban league winners. [Online]. Available: https://www.duckietown.org/archives/66156

[49] ——. Ai driving olympics 6: Urban league. [Online]. Available: https://www.duckietown.org/archives/85031

[50] S. Sinha and A. B. Dieng, "Consistency regularization for variational auto-encoders," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 12 943–12 954. **DOI**: 10.48550/arXiv.2105.14859

András Béres received his BSc and MSc degrees from the Budapest University of Technology and Economics (BME) as an electrical engineer, and currently works as a deep learning engineer at the Continental AI Development Center in Budapest. His interests range from deep learning through robotics to embedded systems.

Bálint Gyires-Tóth is an associate professor at BME. He conducts research on fundamental and applied machine learning since 2007. With his leadership, the first Hungarian hidden Markov-model based Text-To-Speech (TTS) system was introduced in 2008. He obtained his PhD degree with summa cum laude in January 2014. Since then, his primary research field is deep learning. His main research interests are sequential data modelling with deep learning, self-supervised learning and deep reinforcement learning. He also participates in applied deep learning projects, including time series modelling, anomaly detection, computer vision and conversational AI. He was involved in various successful research and commercial projects. In 2017 he was certified as NVidia Deep Learning Institute (DLI) Instructor and University Ambassador. His latest AI- related research achievements contribute to the recently launched Artificial Intelligence Systems National Laboratory as a subproject leader.