Learning to see in the disco: Image canonization for robots

Dominik Walder[1]

***Abstract*— Autonomous robots are expected to navigate a previously unknown environment and accomplish a given task. To succeed in this task, the robot needs specific information of its environment. But often, sensors used by the robot to perceive the world provide excessive information, so called *nuisances*. The actions of the robots should not depend on these nuisances, since they are not informative for the task. We address this problem by creating representations of the environment that are invariant to nuisances. By transforming the sensory input to the robot before it "sees" it, we remove nuisance dependency in the perception pipeline of the robot. This approach is modular and can be applied to any existing system. We use recent advances in neural image-to-image translation to learn nuisance invariant mappings from simulation data, without requiring any data annotation. The learned networks are used to increase the robustness of an existing robot control pipeline in Duckietown, show-casing the effectiveness of the approach.**

I. Introduction

A robot attains information about its environment through its sensors. Often this information is plentiful, and can be categorized into two distinct types: *actionable information* which is the minimal, but sufficient information needed to fulfill the *task* assigned to the robot and *nuisances* which is any other information that is not informative for the task [1]. Common representations of the world, such as images, provide very rich information making them applicable for many different tasks. However, that often means that a large amount of the information are nuisances and a major part of the complexity contained in the information is due to nuisances [2]. A robot computes its actions from a representation and for an optimal agent, these computations must be invariant to nuisances [3]. Humans are a great example for nuisance invariant agents: it does not matter if at day, during night, in snow or a thunder storm, a human will always find his way along a path.

For robotic agents, we can achieve invariance not only by designing a good agent, but as well by how we let it perceive the world. If the representation of the world the agent "sees" is invariant to certain nuisances, the agent will be too. The common approach to achieve invariance to nuisances in representations is to choose a feature representation with certain invariance properties [4], [5]. As Deep Neural Networks (DNN) have started to outperform traditional methods in computer vision [6], they have been used for improved feature extraction as well [7]. In some cases, they have been trained to extract task-specific features that by construction only hold actionable information [8].

[1]The authors are with the Institute for Dynamic Systems and Control (IDSC), ETH Zurich, Switzerland. Please address correspondence to {dwalder, jzilly}@ethz.ch

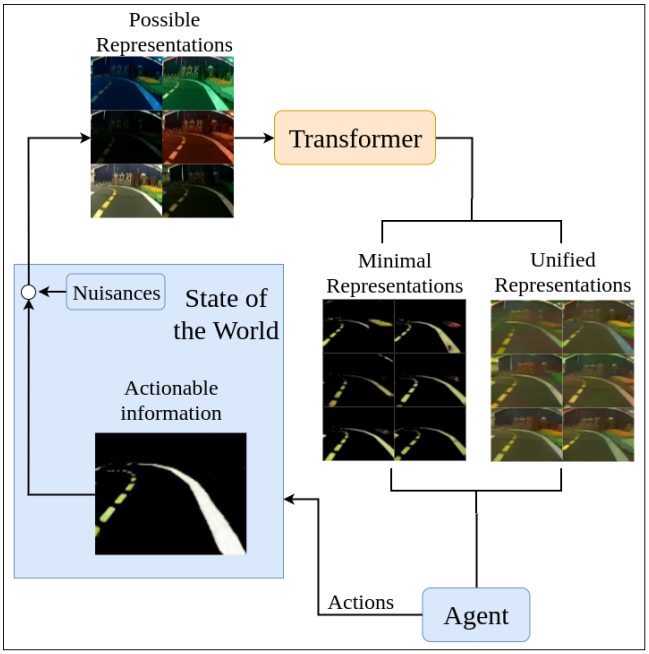

Fig. 1. Improving the robustness of a lane following agent by image canonization. The state of the world consists of actionable information (lane markings) and possible nuisances. There are many possible representations for the same actionable information. A transformer maps all representations into a space where there is only a unique representation for a certain actionable information. We propose two frameworks: one for minimal representations and one for representations close to reality.

While the latter is a promising avenue for robotics, it shares one main disadvantage with all feature-based methods: the new representation lies in a different space with a different dimensionality which creates the need for a new or adapted controller.

To make a modular framework that can be applied onto an existing controller, we must not change the dimensionality of the representation. In this work we use images as representations of the world, so we are interested in image-to-image transformations. So even though we can not change the target representation, we can restrict the information contained in it. There have been two particularly interesting approaches in computer vision that involve mapping into restricted representations: *neural style transfer* [9] and *domain transfer* [10] which is sometimes also referred to as domain adaption [11]. Style transfer maps input images into a certain style that is independent of the input, so the output image is invariant to the input style. If the style is a nuisance, which it is for many tasks in robotics, the output is invariant to the style nuisance. Domain transfer is a more general framework

which defines the output representation mainly through the training data [12], [13].

Both style and domain transfer are used in this work to investigate their effectiveness in image canonization. We target the use in robotic applications, with an emphasis on robust transformations. In addition to numerical validation, we test the proposed approach on a physical robot through using the Duckietown platform [14]. It provides an inexpensive scenario to realistically test algorithms for autonomous robots, particularly self-driving cars. Additionally, the open-source Duckietown software repository[1] provides existing controllers, allowing us to demonstrate the modularity and generality of our approach.

II. Related work

Actionable information We use the concept of actionable information and nuisances [2], as it closely follows our model of how data is represented. There exists more comprehensive theoretical analysis of actionable information and nuisances [1], [15]. A theoretical approach allows for a mathematical definition of specific nuisances. This facilitates an analysis on how to invert or remove them [3]. A key obstacle in theoretical reasoning, is that nuisances and actionable information are often deeply nested and strongly coupled. It can therefore be beneficial to learn such mappings, as especially neural networks trained by stochastic gradient descent tend to learn representations invariant to nuisances in the input data [16].

Neural style transfer In neural style transfer, DNNs are used to generate images where the content comes from one source image (the content source) and the style from another source image (the style source) [9]. Both the style and content of an image can be represented by the activations of deep neural networks. More precisely, the style can be represented by the feature correlations (Gram matrices) of several layers of a DNN previously trained on object recognition. The content of an image can be represented directly by the activations of an earlier layer in the same DNN. The earliest style transfer framework [9] solves an expensive optimization problem for every newly generated image. Later work then introduced models that learned specific styles at training time [17], [18]. To transform an image, a single forward pass through the network is used, reducing computation time by several orders of magnitude. With this innovation, neural style transfer became feasible for real time use. More advanced models are able to learn multiple styles at once [19] or read the style source at test time via adaptive instance normalization [20]. Further control over perceptual factors such as scale and colour have also been introduced [21], as well as a more theoretical analysis of style transfer [22]. We use the architecture of [17], as it provides a stable training procedure and fast test time.

Domain transfer Generative Adversarial Networks (GAN) [23] rely on a combination of a discriminator and a generator. The training process is closely related to a two player game, where the generator tries to generate fake images from a certain distribution (possibly using a prior) while the discriminator attempts to discern the generated images from those drawn from the real distribution. This training framework, where one network learns from the other, has hugely improved artificial image creation [24], [25] and has since been used in most image domain transfer frameworks. The first domain transfer framework introduced was able to learn one-way transition in an unsupervised manner [10]. Later, frameworks for unsupervised image-to-image translation were developed for two way translation [26], [13], [27], [28], for multiple domains [29], [30], [31] and for video retargeting with temporal consistency [32]. The advantage of these frameworks is that unpaired image datasets are easy to create, as no ground truth joint distribution is required. Simultaneously, frameworks that make use of paired data have also been developed [12], [33]. These datasets are harder to acquire as you need image pairs across the domains, however the training procedure is simplified. A special case of paired training data is when the model is forced to learn a minimal intermediate representation of the actionable information [34]. In this work we focus on directly learning the new representation, leaving the application of this framework in robotics for future work.

Simulation Training data can be expensive to annotate, and hence simulation presents a promising alternative. Different methods have been proposed to close the gap between reality and simulation via deep learning [35], [36]. An especially promising method is data augmentation, where the training data from the simulation is augmented in so many different ways, that the learned network generalizes well to reality [37].

Distance measures There exists a broad range of scores to evaluate the quality of generated images [38]. One of the most widely used for generative adversarial networks is the Inception Score [39]. It correlates with human judgment on how realistic generated images look, but cannot be used to compute the distance to any image distribution other than real images. The *Fréchet Inception Distance* (FID) [40] in contrary computes the distance between any two arbitrary image distributions. Additionally, it has been shown to be more robust towards noise and more sensitive towards large artifacts in images [38]. Both are desirable features for a distance measure for our study, as we will use it to compute the similarity between two datasets.

III. Preliminaries

A. Problem Formulation

The general problem setup is displayed in Fig. 2. We have an agent $\mathcal{C}$ that acts as a mapping from sensor input x to control actions y. We assume that the agent has a task and denote any action that leads to success in this task by y_s. The goal of the agent is then:

$$\mathcal{C}(x) = y_s \tag{1}$$

[1] https://github.com/duckietown/Software

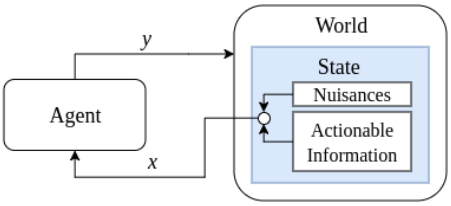

Fig. 2. An agent receives sensor input x from and output's control actions y to the world. The sensor input x is a composition of nuisances and actionable information and hence it might be difficult to decompose it.

x is a composition of nuisances $\mathcal{N}$ and the actionable information $\mathcal{I}_A$, $x = (\mathcal{N}, \mathcal{I}_A)$. By definition, the nuisances do not contain any information relevant for the task, the optimal controller is therefor invariant to nuisances [3]:

$$\mathcal{C}(\mathcal{I}_A, \mathcal{N}) = \mathcal{C}(\mathcal{I}_A) = y_s \tag{2}$$

A controller which lacks this invariance can be made robust if the sensory input x is transformed into z such that the output nuisances do not depend on the input nuisances and the controller is invariant to the output nuisances:

$$\begin{aligned} G(\mathcal{I}_A, \mathcal{N}_x) &= (\mathcal{I}_A, \mathcal{N}_z) \quad \text{with } \mathcal{N}_z \perp\!\!\!\perp \mathcal{N}_x \\ \mathcal{C}(\mathcal{I}_A, \mathcal{N}_z) &= \mathcal{C}(\mathcal{I}_A) = y_s \end{aligned} \tag{3}$$

where $(\mathcal{I}_A, \mathcal{N}_x) = x$ and $(\mathcal{I}_A, \mathcal{N}_z) = z$. In this work, we present different approaches on how to learn such a function G.

B. Duckietown: World, Agent and Task

Our main results are of experimental nature. We use a Duckiebot as our agent and its task is to follow a lane in Duckietown. Please refer to the website `https://www.duckietown.org/` or the main publication [14] for more complete information about both the town and the bot. We will only present the most relevant and important details here.

The Duckiebot has a Raspberry Pi as a computation unit and a camera as its only sensor. There is an open source control pipeline for lane following in Duckietown, which will be used as is. The controller mostly corresponds to what is presented in [14], but as it is actively worked on and improved there might be some slight differences. We use the version found in the Duckietown software repository `https://github.com/duckietown/Software/`[2]. This controller controls the Duckiebots relative position with respect to the lane markings such that it always drives in the middle of the right hand side lane. In Duckietown, the colors of lane markings are very distinct and the controller makes use of that fact during the computation of its relative position. It is therefore mandatory for any transformation that we apply to the input of the pipeline to retain both lane markings and their colors.

We used a light system in Duckietown which is adjustable over the whole RGB color spectrum. Additionally, the intensity can be changed. Although not completely, Duckietown is mostly protected from other light sources. This allows us to insert artificial nuisances in the form of light changes.

[2]commit: 4289ee3923fc9004e3bf0b4be67afae7d5c2e6da

C. Image-to-Image Transformations

The only sensor used by the Duckiebot is a camera, so our input mappings are image-to-image transformations. There are certain properties of images (viewpoint, illumination, content, ...) [15] that can be identified as possible nuisances which then helps to define useful transformations to invert them [3]. Nuisances and actionable information are often tightly knitted together and separating them is a complex task, this is particularly true for images. Neural networks have shown to be extremely effective in learning such complex mappings, and we will use them to learn mappings that create nuisance invariant representations. We would like to emphasize, as seen in Eq. 3, that it is not necessary for the new representation z to contain no nuisances as the controller is often already invariant to a particular set of nuisances. The learned mappings can be seen as a reduction of variability in data - mapping from a large set of nuisances into a much smaller set which we know that the controller is invariant against.

In contrary to most of the recent research in image-to-image transformation, our models are used in a robotic application. The modularity of our approach also brings additional responsibility: as a crucial part of a control pipeline, the transformations add an additional point of failure. It is therefore mandatory that the transformations are robust themselves. No mapping should alter the actionable information, as it would make it impossible for the controller to act correctly.

IV. Method

In this section we present two methods to create representations that are invariant to certain nuisances. For both methods, the use-case is the Duckiebot, living in Duckietown. Both methods are however generally applicable to robotic control pipelines using images and can be used as such with little adaption. All model architectures as well as hyperparameter settings are described in the supplementary material.

A. Neural Style Transfer

Style is in many applications a nuisance. While we can imagine that there are many styles that would make a controller fail to read the content out of an image, it is reasonable to assume that there exists at least one to which the controller is invariant. A neural style transfer network learns to map any image into a single style. If the controller is invariant towards this single style, the style transfer model can be used to create invariance of the pipeline towards many styles.

To train the network, two different losses are used: one for style l_{style} and one for features $l_{feature}$. The style loss is computed from the generated image $\hat{z}$ and a style target z_s and the feature loss is computed from the generated image and the input image x. Please refer to our code for the

exact implementation. The final loss L for training the image generation network is then:

$$L = \lambda_{style} * l_{stlye}(\hat{z}, z_s) + \lambda_{feature} * l_{feature}(\hat{z}, x) \quad (4)$$

We found that training on a single style target z_s can lead to overfitting, leading to artifacts in the generated image. To address this problem, we introduced training on a set of styles $\{z_s\}$ containing N different style targets. For every training iteration we sample a set $\{x\}$ of N input images from the training set and generate stylized images from it: $\{\hat{z}\} = \{\hat{z} = G(x) \mid \forall x \in \{x\}\}$. We then aggregate the loss in a mini-batch:

$$L = \sum_{i=0}^{N} \lambda_{style} * l_{style}(\hat{z}^i, z_s^i) + \lambda_{feature} * l_{feature}(\hat{z}^i, x^i) \quad (5)$$

where i denotes the i-th element of the corresponding set, $\{\hat{y}\}$ and $\{y_s\}$.

We experimented with two different approaches to create $\{y_s\}$. We either chose N images prior to the training or we sampled them from a large style dataset ($\gg N$) before every iteration, effectively training on a whole style dataset.

B. Domain Transfer

Whilst style transfer learns a specific image property which is the style of an image (or set of images), domain transfer learns to map between two domains by exploring the distributions of these domains. It requires more domain specific data to train but can, theoretically, learn arbitrary transformations.

Given two image domains X_A and X_B, we are interested in the mapping $G_{A\to B}(x_A) = x_B$. We distinguish between two different learning approaches: paired, where the joint distribution $P_{X_A,X_B}(x_A, x_B)$ is known at training time and unpaired, where only the marginal distributions $P_{X_A}(x_A)$ and $P_{X_B}(x_B)$ are known and the network needs to learn the joint distribution.

Unpaired domain transfer To enforce correct learning of the joint distribution when training on unpaired datasets, auxiliary constraints are usually introduced. A very common one is the cycle-consistency constraint $x_A = G_{B\to A}(G_{A\to B}(x_A))$. Although there are other constraints in literature [10], [41], [11], we found the cycle-consistency constraint to show the most promising results [26], [13].

It allows for the computation of the difference between the original image x_A and its reconstructed counterpart and so that the network can be trained on this difference. It does however, set an important restriction on the two image domains: it must be possible to encode most information contained in X_A in X_B and vice versa. In other words, $G_{A\to B}$ and $G_{B\to A}$ must both be invertible, meaning they must not remove information, but only transform it. In practice this restriction might be softened, as neural networks approximate functions, so transformations that are almost invertible might suffice. Thus, the two image domains should not be too distant from each other. A natural choice is transferring from reality to simulation, especially as Duckietown has an open source simulation available [42]. It is reasonable to assume that invariance towards nuisances occurring in simulation is easier to achieve than invariance to real nuisances. The question that remains is how real nuisances are translated into the simulation domain by the learned model. Some possible answers to that question are given in Sec. VI, however a comprehensive analysis of this topic is not within the scope of this work.

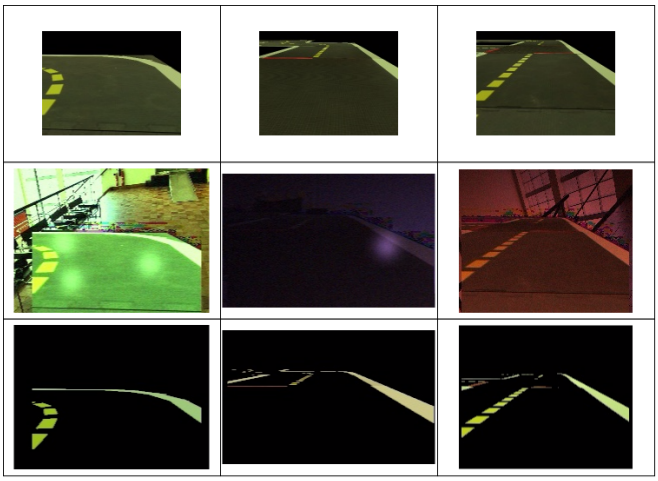

Fig. 3. Examples of randomized transformations on training images for the pix2pix framework. *Top:* road images extracted from simulation. *Middle:* randomly augmented road images. *Bottom:* extracted lane markings. The middle and the lower row are the two image domains with which the pix2pix framework are trained on.

Paired domain transfer Knowing the ground truth joint distribution simplifies training, especially as it gives two main advantages over unpaired frameworks:

- For every transformation $G(x_A) = \hat{x}_B$ we know the ground truth x_B. That provides a powerful tool to train the network and no additional constraints are necessary [12].
- $G_{A\to B}$ does not need to be invertible, as we do not require $G_{B\to A}$.

So we can learn a transformation that removes more nuisances, possibly even all nuisances. Recall that the Duckiebot lane follower requires colored lane markings for navigating the roads of Duckietown, meaning that these lane markings are the minimal representation. However, to learn a transformation from a real image to an image only containing lane markings, we would need a dataset with ground truth image pairs. Creating such a dataset by hand is an expensive task that results in a high initial cost to deploying our method and it is thus best avoided. As an alternative, we propose to create a fake joint distribution from simulation and use data augmentation to make the trained network robust enough to work in reality too. This was inspired by recent research [37] which showed that going from simulation to reality is possible even for very complex tasks. Having a fake joint distribution retains the above mentioned main advantages of

paired datasets.

V. Experiments

In this section we provide the experimental setup that was used to validate our proposed methods. We split our experiments into two phases. First, we trained networks and evaluated their effectiveness by human eye. This step was especially useful for neural style transfer, as it is unclear what a good style image is. A small set of images of Duckietown that were taken with a smartphone camera were used to evaluate if a network performed well enough to go the second experimental stage. This second stage consisted of numerical evaluation on data variability reduction as well as a set of experiments in Duckietown [14]. The purpose of this was to evaluate how much we can improve the robustness of an existing controller by transforming the input image.

A. Network Training

Neural style transfer To train a neural style network, a style target z_s is needed. We used simulation images as they are readily available and contain all the important features (colored lane markings). Furthermore, they also contain fewer distracting objects than real images. To determine what a good style image is, we experimented with the number of styles by cropping certain parts of the style images, manually enhancing colors and changing the complexity of the features contained in the image (road with background, only road or only lane markings). Our model closely follows the architecture proposed in [17][3]. We used instance [43] instead of batch normalization after the convolution layers, as previous work found it to be superior for style transfer models [44]. As suggested by the authors of the original architecture, we used the Microsoft COCO dataset as the source for training images [45].

Domain transfer For the two domains used in unpaired domain transfer we used a dataset taken from Duckietown logs and a dataset collected through simulation, respectively. We then tried to learn a mapping from real images to simulation with the UNIT framework [26]. The UNIT framework was chosen as the authors reported successful transformations from real to simulation images.

For paired domain transfer, as described in Sec. IV-B, simulation datasets were used to train the pix2pix framework [12]. To create this fake distribution, we used the official simulation of Duckietown [42][4]. The simulation was modified, such that it only outputs the road, without any background or objects. We then let an agent in the simulation follow the lane and extracted the simulated road images that it sees. To create the fake real images we performed a series of random transformations to the road images. See the supplementary material for the code and Fig. 3 for examples. The lane marking images were created by thresholding the road image. This is possible because the colors in the simulation are very distinct and it is rather easy to discern the white, yellow and red lane markings from the gray road. We created a dataset of 746 randomized image pairs. As the data generation process is very cheap, it would be possible to increase this number without significant additional cost. However, for our purposes this small dataset was sufficient. This shows once more how neural networks, especially GANs, can generalize well even if the learning problem is heavily over-parameterized [12], [13].

[3]We based the model on a open source implementation found in `https://github.com/pytorch/examples`

[4]The gym was still under development during our work. Our version was frozen at 8th July 2018.

B. Data variability reduction

From here on we assume that we have a set of transformations G for which we conduct all further experiments.

TABLE I

The six different lighting conditions used for experiments in Duckietown. We can reproduce all of these lighting conditions with the light system in Duckietown.

name	color	brightness
blue	blue	100%
green	green	100%
red	red	100%
b_white	white	100%
m_white	white	30%
l_white	white	10%

Fréchet Inception Distance The Fréchet Inception Distance (FID) was used to compute the similarity between two datasets. Let us denote a set of images by D, a lighting condition by L, the set of all lighting conditions shown in Tab. I by $\{L\}$ and the set of all transformations by $\{G\}$. We then collect a dataset of images in Duckietown for every light condition D_L $\forall L \in \{L\}$. Then, we transform every so collected dataset by every transformation $G \in \{G\}$, creating D_L^G. Now let $f_{FID} : (D_A, D_B) \rightarrow \mathbb{R}$ be the function that computes the FID, where (D_A, D_B) is a pair of datasets. Since we do not merely want to compute the distance between a single pair of datasets, but instead between a collection of several datasets, we propose the following scheme: Let C_L be the set of all possible combinations of two lighting conditions drawn from $\{L\}$. The average FID for a given transformation G is then computed as follows:

$$FID = \frac{1}{N} \sum_{(L^A, L^B) \in C_L} f_{FID}(D_{L^A}^G, D_{L^B}^G) \tag{6}$$

where N is the size of C_L. Note that we use the FID as a distance measure, but this procedure can be used with any measure that can compute similarity between datasets. It provides the average distance or similarity between a collection of datasets. If this distance is high, or the similarity low, it suggests that there is a large variability within the datasets in the collection. Our goal is to provide a transformation that is invariant to environmental influences like light. We therefore want a high similarity between all datasets, even if they are collected under different environmental conditions.

Entropy In addition to the FID, we report the average entropy [46] of images before and after they have been

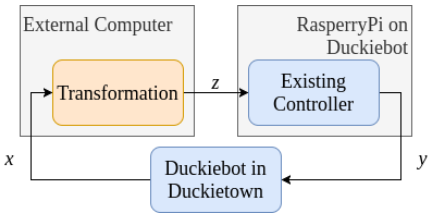

Fig. 4. The control pipeline used for the Duckiebot. x is the image from the Duckiebot camera, z is its transformed counterpart and y are the actions computed from the Duckietown lane following controller.

transformed. We compute the entropy from the histograms of every color channel separately and then take the average. Although this measure does not capture actual visual information, it informs about potential information contained in the image.

C. Tests in Duckietown

The experiments conducted on physical robots in Duckietown are explained in the following section. The setup provided aims to test if improvements to the robustness of the robot can be achieved with the transformations proposed.

Fig. 4 shows the pipeline used for lane following. We used the existing controller as described in Sec. III-B. A small set of changes had been implemented:

- The image was rerouted to a laptop for processing before sending it to the control pipeline. Since the Duckiebot runs on a Raspberry Pi, we could not compute the forward passes for the neural networks locally.
- The image size was reduced to 120x160 instead of 480x640 before it was sent to the laptop to decrease introduced delays. Originally, the image size was reduced in two steps: first to 192x256 at the color correction step and then to 120x160 before the line detection step.

Apart from that, the controller was left as it is. It was previously tuned to work well under laboratory conditions (bright, white, diffuse light from above) and we did not change any of the tuning parameters. We then defined three different runs r in Duckietown (see Fig. 5), listed in order of increased difficulty:

- $r_{straight}$: A straight line. To stay on the lane, it is enough it either detect the white or the yellow line. Note: the duckiebot accumulates enough drift over the whole run that just going straight is not enough to stay on the lane.
- r_s: An S-curve. It consists of a first soft left turn that requires detecting the white line (because the yellow will have left the field of view when the turn is entered) and a second hard right turn that requires the detection of the yellow line.
- r_{curvy}: A longer, curvy run. This run combines several turns in both directions. Completing it requires error free control for approximately 30s. The control rate is approximately 14Hz, which means we need over 400 continuous and error free control commands to succeed in this run.

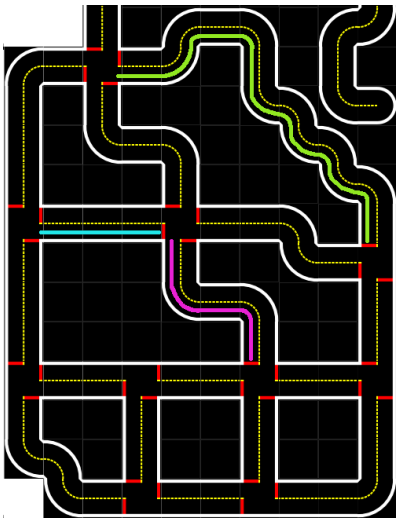

Fig. 5. The Duckietown map including the three runs chosen for our experiments. The bot drives on the right hand side. *blue:* $r_{straight}$. *pink:* r_s. *green:* r_{curvy}

The task for the bot is then to complete a run from beginning to the end while always staying in the lane. All runs were recorded and classified as a failure or success. Additionally, the time between start and end of a run, was recorded as the time-to-end t_e. We define the end here as the time when the bot either clearly failed (not recoverable) or succeeded.

To create controllable, artificial nuisances we adjust the lighting in Duckietown. The same six lighting conditions as in Sec. V-B, mentioned in Tab. I, were used. A single test T consists of a run r, a lighting condition L and a transformer G. The output of a test is the duration of the test t_e and the binary variable of success s.

$$T(r, L, G) = (t_e, s) \tag{7}$$

A test suite $S_T(G)$ consists then of the following tests:

- 20 times $T(r_{straight}, L, G) \quad \forall L \in \{L\}$
- 10 times $T(r_s, L, G) \quad \forall L \in \{L\}$
- 10 times $T(r_{curvy}, L, G) \quad \forall L \in \{L\}$

We performed a test suite for all transformers G.

VI. Results & Discussion

The following section shows the results of the neural network training. For a few selected networks, we will present a numerical evaluation as well as tests conducted on the physical robot.

A. Network Selection

Neural style transfer Fig. 6 displays the results for a selected set of style transfer networks. Our training framework allows training the network with an arbitrary amount of style images. We show the result for a single style, for 20 styles and for a large dataset of 2048 styles. When training with a single style, strong features as red lane markings appear on the generated images as artifacts. Using 20 styles

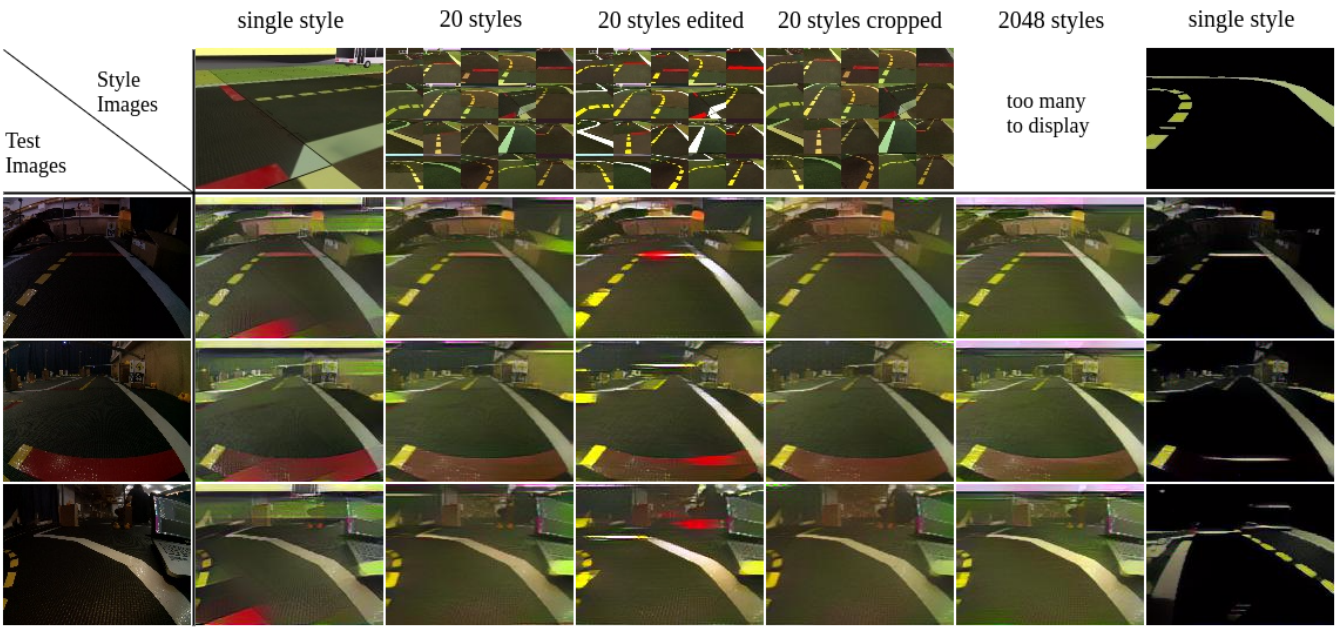

Fig. 6. Style transfer for different models. The top row shows the style image(s) used to train the network. The three lower rows show to the left an original image and the respective transformed images after the style transfer was applied.

already regularizes the style network sufficiently, such that these artifacts disappear. We could not find any advantage with using more than 20 styles.

Upon closer inspection, one can see that the generated images have a green tint. The sections of the style images that appear most green are generally meadows or the sky. These features appear mostly in the top part of the images. As a result, we were able to minimize the green tint through cropping the upper part of the images.

Regularization through increasing the style images produces an unwanted effect: the colors of the generated image are less rich. This might be a problem later since the colors of the lane markings need to be very distinct for the Duckiebot controller. To encourage the network to generate richer colors, we edited the style images by manually redrawing the lane markings with very intense colors. However, we found that this resulted in the network overfitting and producing generated images with overly intense colors which altered the actionable information.

Another interesting approach was the removal of features in the style images. We show here the results of a network that was trained on a single style image that contained only white and yellow lane markings. This network can emphasize the lane markings very effectively.

Two networks were then used for further evaluation. Firstly, we used the network trained on 20 styles with cropped backgrounds as we found it produced the best results in generating simulation-like images whilst keeping the actionable information intact. Secondly, the network trained on only lane markings was used as it creates a large contrast between markings and road. We hypothesized that this would help the Duckiebot controller to extract the lane markings.

Fig. 7. Transformation through a network trained with the pix2pix framework. *Top:* Real images taken in Duckietown with a smartphone camera. *Bottom:* Their transformation through the pix2pix network forward pass.

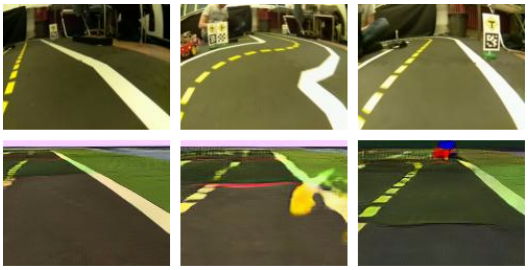

Fig. 8. Transformation through a network trained with the UNIT framework. *Top:* Real images taken in a Duckietown with a Duckiebot camera. *Bottom:* Their transformation through the UNIT network forward pass.

Paired domain transfer Fig. 7 shows the of the pix2pix framework trained as described in Sec. IV-B. Despite the small training set size, the generalization to unseen data seems to work well.

Unpaired domain transfer The absence of paired data

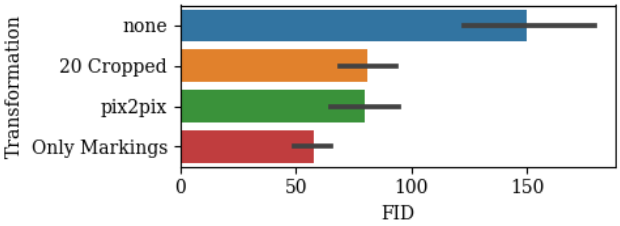

Fig. 9. The averaged FID scores between datasets taken under different lighting conditions. See eq. 6 for how the FID is computed for a given transformation. A small FID suggests higher similarity between generated images.

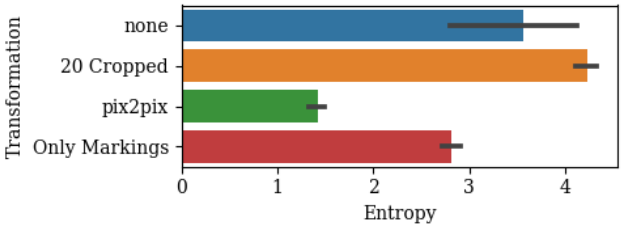

Fig. 10. Entropy of images before and after they have been transformed by the given transformation. A low entropy mean suggest low information and low standard deviation suggests low variability of information between the images.

makes training significantly more difficult. Although we invested a similar or higher amount of time in parameter and dataset tuning compared to the other methods, we were not able to produce satisfying transformations. Fig. 8 shows results for one of the better performing models. Although in some cases, the transformation works rather well, the chance of altering the actionable information is too high for it to be practically deployed on a physical robot. It is very likely that with the right configuration or with a different framework, better results could have been achieved. However, we found that the training procedure as well as the resulting models are not yet stable enough for usage in robotics. Nevertheless, unpaired domain transfer remains a very promising area of research and should be explored further in future work.

Selection We choose to further evaluate the above two above mentioned style transfer models as well as the paired domain transfer model. We will refer to the style transfer models as *20 Cropped* and *Only Markings*. The domain transfer models will be referred to as *pix2pix*. A single transformation is denoted by G and all three together by $\{G\}$.

B. Data variability

We report the Fréchet Inception Distance (Fig. 9) as well as the entropy (Fig. 10) for all three selected transformations. The successful reduction of the FID for all three transformations means that they were able to successfully increase the similarity between the different lighting conditions. Since the only notable differences between the datasets before the transformation is the light color and intensity, we can say that we achieved at least partial invariance against light nuisance.

For the *20 Cropped* model, the average entropy has a slightly higher value compared to before the transformation. This is because the low intensity images before the transformation contain low amounts of information. Both *pix2pix* and *Only Markings* where able to reduce the contained information, suggesting that they were both able to remove nuisances. The significantly lower standard deviation after the transformation means that the amount information contained in the output images is very consistent suggesting a low dependence on the amount of information in the input images.

C. Driving through Duckietown

Fig. 11 shows an example for every lighting condition and every transformer used in the experiments. Despite significant variability between the original images, we can see that the generated images for a single transformation look alike.

The experiments are split up in many runs. A run takes between 8 and 32 seconds. Since the run can fail right at the start or shortly before the end, simply measuring success or failure does not accurately reflect the complexity of the task. Therefore, we also measured the time to end t_e and introduce a success rate measure R_s:

$$R_s = \begin{cases} \frac{t_e}{q_{t_e}^{75}} & \text{if } r = f \\ 1 & \text{else} \end{cases} \tag{8}$$

where $q_{t_e}^{75}$ is the 75% quantile of t_e for successful events for a single run. Fig. 12 shows the success rates for all lighting conditions and transformations. There are three important points when interpreting the results:

1) The actual time taken for a successful run varies with $\pm$ 15%. This is due to variations in the speed of the robot and it creates a similar variation in the success rate for a single run.
2) Even a maximally unstable controller does not leave the lane in zero seconds. The fastest time to leave the lane is just under one second. So even a controller which drives straight off the road will have a success rate higher than zero. The value of the success rate that suggests that the controller was at least partially successful depends on the duration of a successful run, so it varies between the three runs.
3) The network which was used to send the images between the external computer and the Raspberry Pi introduced an additional latency in the order of 10 milliseconds. It was found that this delay had peaks at more than 1 second. We could not however find a pattern for the occurrence of these peaks and almost all measurements were affected by a similar amount of latency peaks. As a result, around 10 to 15 % of the r_s runs and 20 to 30 % of the r_{curvy} runs failed due to such latency peaks. This had a negligible influence on the $r_{straight}$ runs. Consequently, the success rate

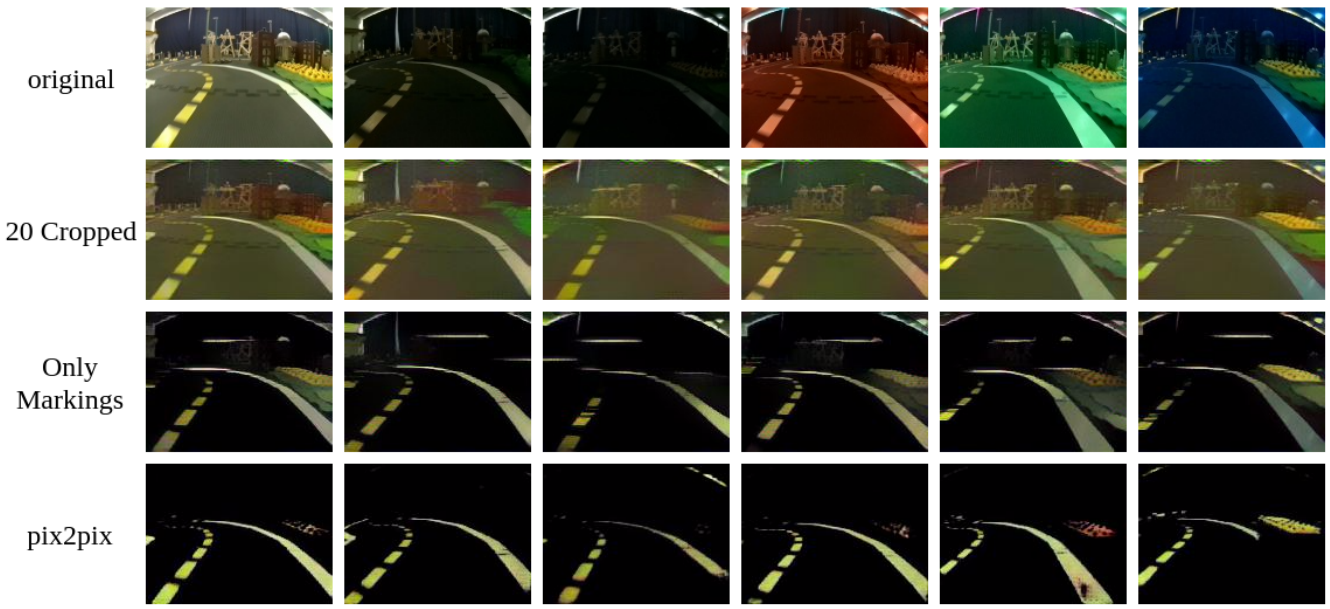

Fig. 11. Image canonization properties of selected transformation models. The top row shows images taken in Duckietown under different lighting conditions. The lower rows show the same images, but transformed by a network, as noted to the left.

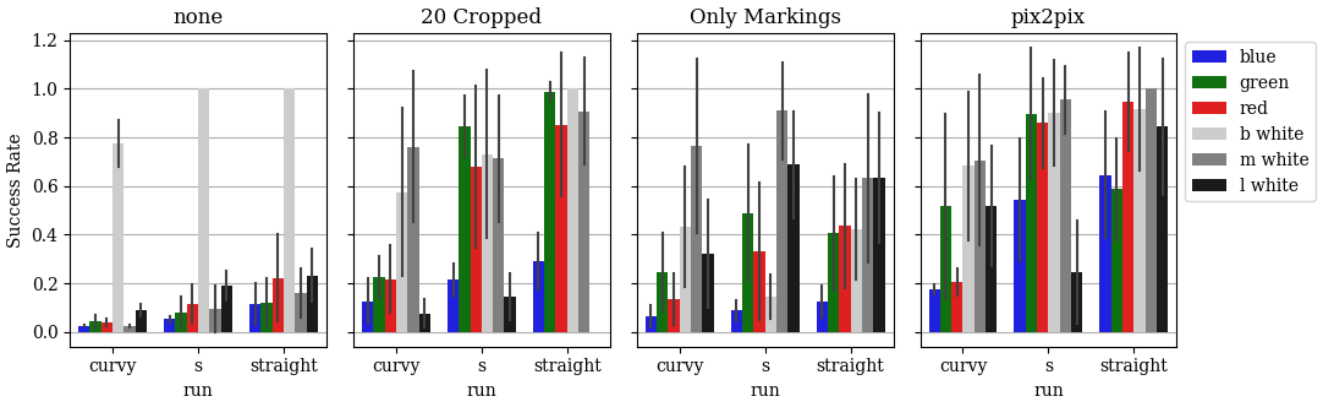

Fig. 12. Success rate for the lane following task of the Duckiebot. The different plots correspond to different transformations that were applied to the input image. Curvy, s and straight denote the different runs in Duckietown as shown in fig. 5. The colors correspond with the lighting condition in Duckietown.

shown suggests a worse performance than we can expect from the different transformations.

The existing controller performs well under bright, white light. In any other case it fails. *20 Cropped* and *pix2pix* were able to significantly improve the robustness in most cases. However, they achieved slightly worse results for the bright white light compared to no transformation. Overall, *pix2pix* performed best. *Only Markings* performed worse especially in cases of bright light.

For both *20 Cropped* and *pix2pix*, the most common failure cases are badly reproduced colors. The controller is very sensitive to colors and if a white lane is slightly yellow in color, it is often misclassified. Upon inspection of the logs, it became clear that *Only Markings* altered the actionable information because it would paint small white lanes at places where there are none. This occurred particularly in lighting conditions where brightness was high. Videos of failure cases for different lighting conditions can be seen in the supplementary material.

VII. Conclusion

We have shown that both neural style transfer and domain transfer can be used as image canonization methods. They can effectively create representations that are invariant to nuisances. Furthermore, the nuisance invariant representations can be used to improve the robustness of robotic agents. This enhanced robustness towards nuisances facilitates the transition of existing controllers into new environments.

Acknowledgements

First and foremost, I would like thank Julian Zilly for his advice and guidance throughout the whole work. Furthermore, Andrea Censi's input was highly appreciated and

helped shaping the thesis and give it structure. I would like to thank all citizens of Duckietown, in particular the major Gianmarco, for all the help concerning the Duckiebot and Duckietown. And last but not least, thanks to Emilio Frazzoli for making all of this possible.

REFERENCES

[1] S. Soatto, "Actionable information in vision," in *Machine learning for computer vision*. Springer, 2013, pp. 17–48.

[2] ——, "Steps toward a theory of visual information," *Technical Report UCLA-CSD100028*, 2010.

[3] A. Censi and R. M. Murray, "Uncertain semantics, representation nuisances, and necessary invariance properties of bootstrapping agents," in *Development and Learning (ICDL), 2011 IEEE International Conference on*, vol. 2. IEEE, 2011, pp. 1–8.

[4] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*, 2014, pp. 647–655.

[8] A. Loquercio, A. I. Maqueda, C. R. del Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[9] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2414–2423.

[10] Y. Taigman, A. Polyak, and L. Wolf, "Unsupervised cross-domain image generation," *arXiv preprint arXiv:1611.02200*, 2016.

[11] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4243–4250.

[12] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *arXiv preprint*, 2017.

[13] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *arXiv preprint*, 2017.

[14] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang *et al.*, "Duckietown: an open, inexpensive and flexible platform for autonomy education and research," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1497–1504.

[15] S. Soatto and A. Chiuso, "Visual representations: Defining properties and deep approximations," *arXiv preprint arXiv:1411.7676*, 2014.

[16] A. Achille and S. Soatto, "Emergence of invariance and disentangling in deep representations," *arXiv preprint arXiv:1706.01350*, 2017.

[17] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European Conference on Computer Vision*. Springer, 2016, pp. 694–711.

[18] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images." in *ICML*, 2016, pp. 1349–1357.

[19] V. Dumoulin, J. Shlens, and M. Kudlur, "A learned representation for artistic style," *Proc. of ICLR*, 2017.

[20] X. Huang and S. J. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization." in *ICCV*, 2017, pp. 1510–1519.

[21] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman, "Controlling perceptual factors in neural style transfer," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[22] Y. Li, N. Wang, J. Liu, and X. Hou, "Demystifying neural style transfer," *arXiv preprint arXiv:1701.01036*, 2017.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[24] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[25] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," *arXiv preprint arXiv:1511.05440*, 2015.

[26] M.-Y. Liu, T. Breuel, and J. Kautz, "Unsupervised image-to-image translation networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 700–708.

[27] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, "Multimodal unsupervised image-to-image translation," *arXiv preprint arXiv:1804.04732*, 2018.

[28] M.-Y. Liu and O. Tuzel, "Coupled generative adversarial networks," in *Advances in neural information processing systems*, 2016, pp. 469–477.

[29] Y. Aytar, L. Castrejon, C. Vondrick, H. Pirsiavash, and A. Torralba, "Cross-modal scene networks," *arXiv preprint arXiv:1610.09003*, 2016.

[30] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation," *arXiv preprint*, 2018.

[31] A. Liu, Y.-C. Liu, Y.-Y. Yeh, and Y.-C. F. Wang, "A unified feature disentangler for multi-domain image translation and manipulation," *arXiv preprint arXiv:1809.01361*, 2018.

[32] A. Bansal, S. Ma, D. Ramanan, and Y. Sheikh, "Recycle-gan: Unsupervised video retargeting," *arXiv preprint arXiv:1808.05174*, 2018.

[33] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, G. Liu, A. Tao, J. Kautz, and B. Catanzaro, "Video-to-video synthesis," *arXiv preprint arXiv:1808.06601*, 2018.

[34] C. Chan, S. Ginosar, T. Zhou, and A. A. Efros, "Everybody dance now," *arXiv preprint arXiv:1808.07371*, 2018.

[35] N. Ruiz, S. Schulter, and M. Chandraker, "Learning to simulate," *arXiv preprint arXiv:1810.02513*, 2018.

[36] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[37] OpenAI, :, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning Dexterous In-Hand Manipulation," *ArXiv e-prints*, Aug. 2018.

[38] A. Borji, "Pros and cons of gan evaluation measures," *arXiv preprint arXiv:1802.03446*, 2018.

[39] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[40] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems*, 2017, pp. 6626–6637.

[41] S. Benaim and L. Wolf, "One-sided unsupervised domain mapping," in *Advances in neural information processing systems*, 2017, pp. 752–762.

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

[43] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization. arxiv 2016," *arXiv preprint arXiv:1607.08022*, 2016.

[44] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis." in *CVPR*, vol. 1, no. 2, 2017, p. 3.

[45] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[46] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[47] R. Liu, J. Lehman, P. Molino, F. P. Such, E. Frank, A. Sergeev, and J. Yosinski, "An intriguing failing of convolutional neural networks and the coordconv solution," *arXiv preprint arXiv:1807.03247*, 2018.

APPENDIX

A. Data augmentation

Here we explain how the dataset for the pix2pix training was created. We assume to have a set of background images $\mathcal{I}_B$ and a set of road images $\mathcal{I}_R$. As background images we used photographs of our lab and the road images are simulation images that only contain a road and are otherwise black. We repeat the following steps to create an image pair (the equivalent procedure in code form can be found in our public repository):

- Sample background image I_A and road image I_R from $\mathcal{I}_B$ and $\mathcal{I}_R$, respectively.
- Resize background image to $size(I_R) + margins$, where $margins$ are random integers between 0 and 100.
- Scale the width of I_R such that the new width is at least its old width and at most the width of I_A, determined randomly.
- Create only markings image I_M by thresholding I_R such that only the lane markings are left[5].
- Allocate a black image I_B of the size of I_B
- Place I_M in I_B and I_R in I_A, both at the same position. They are vertically aligned at the bottom, the horizontal alignment is chosen randomly.
- I_A is augmented by the $augment\_image$ function described at the end of this document. We report the function in Python.

B. Neural Style Transfer Network

Architecture We use the architecture proposed by [17], as shown in Tab. II. For all experiments shown in this work, the batch normalization layers were exchanged with instance normalization [43]. A comparison of results for batch vs instance normalization can be seen in Fig. 17. A "$C \times H \times W$ conv" block denotes a convolutional layer with C filers of size $H \times W$ which is immediately followed by a normalization layer and a ReLU nonlinearity. Residual blocks are displayed in Fig. 13. All convolutional layers use reflection padding such that the output size is only dependent on the stride.

TABLE II
ARCHITECTURE OF NEURAL STYLE MODELS

Layer	Activation Size
Input	$3 \times 256 \times 256$
$32 \times 9 \times 9$ conv, stride 1	$32 \times 256 \times 256$
$64 \times 3 \times 3$ conv, stride 2	$64 \times 128 \times 128$
$128 \times 9 \times 9$ conv, stride 2	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 128 \times 128$
$32 \times 3 \times 3$ conv, stride 1/2	$32 \times 256 \times 256$
$3 \times 9 \times 9$ conv, stride 1	$3 \times 256 \times 256$

[5]Threshold values: every pixel that has either a red or gray value above 100 is kept. All others are set to zero. Gray values are computed with OpenCV ($gray = 0.299 * red + 0.587 * green + 0.114 * blue$).

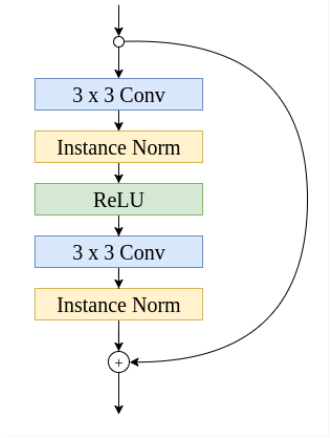

Fig. 13. Residual Block

Hyper parameters For all experiments showed in this workd, the content loss weight was $1e5$ and style loss weight $1e10$. Learning rate was 0.001.

C. pix2pix Network

The model is adapted from the original pix2pix paper [12], please refer to it for the specifications of the architecture. We used the implementation of `https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix`[6] and took the UNET architecture as decoder-encoder with 8 stages each (unet_256). We experimented with different architectures and parameters and we found this one with default parameter setting to work best.

D. UNIT Network

The model is taken from the original UNIT paper [26], please refer to it for specifications of the architecture. We used the implementation of `https://github.com/mingyuliutw/UNIT`[7]. We performed different experiments:

- Changing number of convolutional layers and residual layers: We tried to create low dimensional intermediate representations, as the main information (where is the lane), does not need a high dimensional representation. This showed to be ineffective. Additional losses/constraints would be needed to force the intermediate representation to hold specific information of road position and orientation. Especially new works about multi-domain image-to-image translation could provide inspiration (e.g. [31], [27].
- CoordConv layers [47] were used to make the convolutional layers more position aware. Although we see a promising feature in these new layers, we could not see an immediate advantage in our experiments.
- We introduced a content loss similar to style transfer content loss to force transformations to keep similar content between the two domains. However, this showed to be not effective.

[6]commit: f70e2ac595177ab2ce60aeec448bf4102fcde3c0
[7]commit: 25e99afe267df6eea2c97d23b05a42683d75e53c

```python
import random
import numpy as np
import cv2

def augment_image(img):
    img = gradient(img)
    for _ in range(4):
        if random.random() < 0.2:
            img = spot(img)
    img = add_noise(img)
    img = scale_channels(img)
    return img

def gradient(img):
    """ Scale an image with a gradient. Scale, vertical and horizontal gradient
    are chosen randomly"""
    choices = [True, False]
    height, width = img.shape[:2]
    if random.choice(choices):
        vec1 = np.linspace(0.3, 1.0, height)
        vec1 = vec1[:, None]
    else:
        vec1 = np.ones((size[0], 1))

    if random.choice(choices):
        vec2 = np.linspace(0.3, 1.0, width)
        vec2 = vec2[None, :]
    else:
        vec2 = np.ones((1, size[1]))

    grad = vec1.dot(vec2)
    grad *= scale
    grad = grad[:, :, np.newaxis]
    return img * grad

def spot(img, k_size=51, sigma=10, scale=100.0):
    """ Place a gaussian kernel at a random position in the image."""
    # Create kernel
    scale *= kernel_size * sigma
    kernel = cv2.getGaussianKernel(k_size, sigma)
    kernel = kernel.dot(kernel.transpose()) * scale
    kernel = kernel[:, :, np.newaxis].astype(np.uint8)

    # Place the kernel at random position
    height, width = img.shape[:2]
    h_start = int(random.random() * (height - k_size))
    w_start = int(random.random() * (width - k_size))
    img[h_start:h_start + k_size, w_start:w_start + k_size, :] += kernel
    return img

def add_noise(img):
    """ Add noise to the image"""
    img = img.astype(np.float)
    img += np.random.randint(-20, 20, img.shape)
    img = np.clip(img, 0, 255)  # Make sure it does not overflow
    return img.astype(np.uint8)

def scale_channels(img):
    """ Scales each color channel independently, changing overall color and brightness."""
    img = img.astype(np.float)
    img *= (np.random.random((1, 1, 3)) * 1.7 + 0.1)
    img = np.clip(img, 0, 255)  # Make sure it does not overflow
    return img.astype(np.uint8)
```

E. Neural Style Examples

In this section we show some examples for style images and the resulting transformations that did not fit into the paper. Fig. 17 shows a comparison between batch and instance normalization.

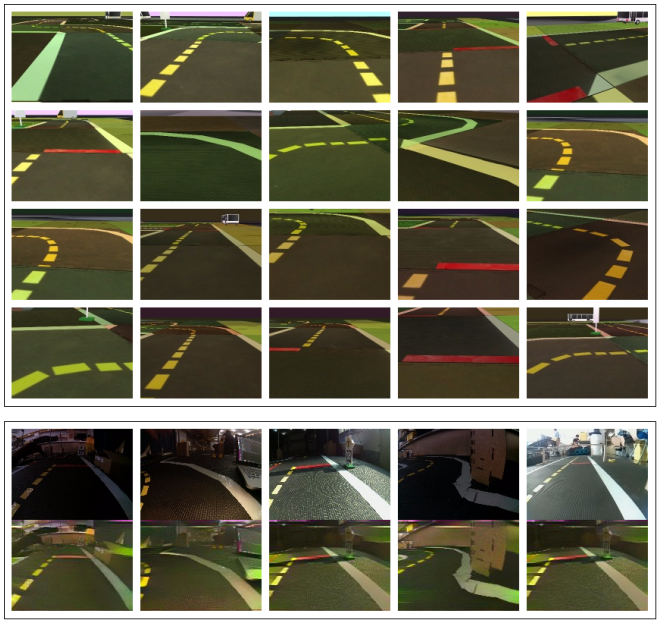

Fig. 14. *Top:* Style images *Bottom:* Resulting transformations (Upper row: original images, lower row: transformed images)

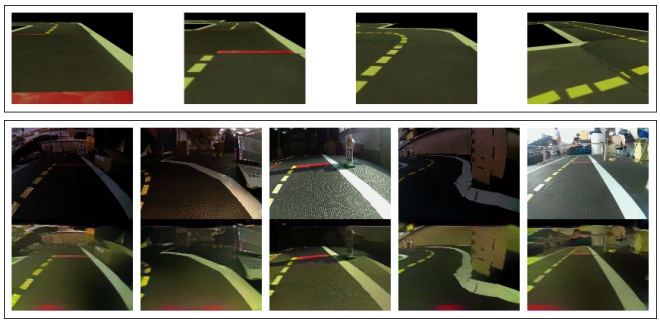

Fig. 15. *Top:* Style images *Bottom:* Resulting transformations (Upper row: original images, lower row: transformed images)

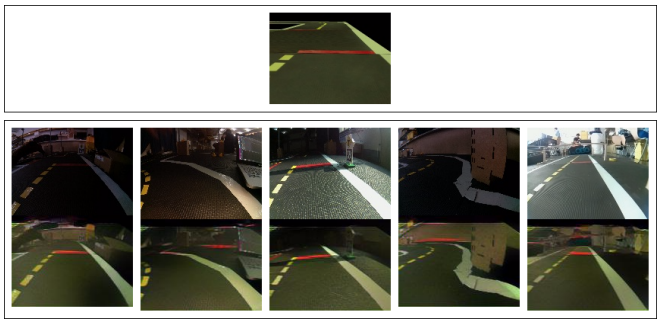

Fig. 16. *Top:* Style image *Bottom:* Resulting transformations (Upper row: original images, lower row: transformed images)

Fig. 17. Comparison between instance and batch normalization. *Top:* 4 style images. *Bottom:* resulting transformations. The upper row shows the original images, the middle row shows images generated from the model trained with instance normalization and the lower row images generated from a model trained with batch normalization.