Supplement file of VR-Goggles for Robots: Real-to-sim Domain Adaptation for Visual Control

Jingwei Zhang*[1] Lei Tai*[2] Peng Yun[2] Yufeng Xiong[1] Ming Liu[2] Joschka Boedecker[1] Wolfram Burgard[1]

A. Detils for artistic style transfer and temporal error map

For the sequence artistic style transfer, implementation wise, we use the pretrained *VGG-19* as the loss network, *relu*2_2 as the content layer, *relu*1_2, *relu*2_2, *relu*3_2 and *relu*4_2 as the style layers. We set the weight for each loss as: 1e5 for content, 2 for style, 1e-7 for spatial regularization, 10 for optical flow, and 100 for shift. The downsampling factor K for the transformer network [1] is 4. Shifts are uniformly sampled from $[1, K-1]$ for every training frame.

We follow the same procedure of computing the temporal error as in [2] to evaluate the consistency between consecutive frames:

$$E_{temporal} = \sqrt{\frac{1}{(T-1)\times M}\sum_{t=1}^{T-1}\sum_{m=1}^{M}\mathbf{c}_m\left(\hat{s}_m^t - f\left(\hat{s}_m^{t+1}\right)\right)^2},$$

where N is the number of all pixels (width $\times$ hight $\times$ channel) and T represents the length of the sequence. f wraps the stylized frame $t+1$ back to t based on the ground truth optical flow provided in the *Sintel* datset. Fig. 3 in the main paper shows a sample temporal error map for two consecutive frames.

We train all the three methods for $40,000$ steps on the collected Videvo dataset with an Nvidia GTX 1080 Ti. The batch size is 1. The training time for of *FF*, *FF+flow* and *Ours* are 3.27, 3.33 and 5.18 hours respectively. Note that it takes approximately 3 hours before training to compute the optical flow which is required for the training of the *FF+flow* method, and this method also requires sequential training data while *Ours* does not.

B. Training details of Carla benchmark evalution

We use the goal-directed navigation benchmark [3] in *Carla* simulator. We deploy all experiments in Town 1 and test under the testing weather condition. There are four tasks in the benchmark including *Straight*, *One turn*, *Navigation* and *Navigation with dynamic obstacles*. Each of the tasks is carried out over 25 episodes. Each episode has a preset starting and goal location. The agent needs to reach the goal within a certain time budget to achieve a successful episode. The time budget is defined by the time needed to reach the goal when travelling along the optimal path within 10 km/h. The optimal path to finish all 25×4 episodes in the benchmark is around 53 km.

All of the training of the policies and the adaptation models are implemented with Pytorch and deployed with a single Nvidia 1080Ti GPU. Images provided in the *Carla* dataset are of size 88×200. We train our visual control policy following the branched structure of condition imitation learning [4]. The speed of the car is also taken as the input. Four differnt branches corresponds to four differnt high level commands (*straight*, *left*, *right*, *follow line*) provided by the global path planner. Each of the branches will output three values indicating the steering angle, the acceleration and the brake respectively. The embedding of the image is also used to predict the speed of the car as an auxiliary task. The same data augmentation methods as in [4] are incorporated including contrast variations, brightness and tone variations, adding Gaussian blur, Gaussian noise and salt-and-pepper noise, and masking out a random set of rectangles in the image. The batch size for training is 1000. Our implementation code for this part is public[1].

	daytime	daytime after rain	clear sunset	daytime hard rain
Training	853	840	715	816
Evaluation	99	99	84	93

TABLE I: Sequence data distribution under the four different weathers of the original *Carla* dataset. Each of the sequence consists of 200 frames.

The original Carla dataset consists of 3289 sequences for training and 374 sequences for evaluating where each sequence includes 200 frames. We separate them based on the four weather conditions as shown in Table I.

We train all of the policies for 90 epochs with Adam and an initial learning rate of 0.0002. The learning rate is reduced by half every ten epochs. The training for each of the three *Single-Domain* policies takes around 5 hours under its corresponding training weather condition. The training of the *Multi-Domain* policy takes almost 16 hours since it uses all of three training weather conditions as described in Section IV-B. of the main paper. The evaluation dataset is used to choose the best model after

*indicates equal contribution.

[1]Department of Computer Science, University of Freiburg. {zhang, xiongy, jboedeck, burgard}@cs.uni-freiburg.de

[2]Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology. {ltai, pyun, eelium}@ust.hk

[1]https://github.com/onlytailei/carla_cil_pytorch

(a) Input (b) 5 epochs (c) 50 epochs (d) 100 epochs (e) 150 epochs (f) 200 epochs

Fig. 1: The effects of *VR-Goggles* transfer model training at different training stages. After 50 epochs, the output of the transfer model is quite stable.

each epoch. We randomly choose 30 sequences from the evaluation data under each of the three training weather conditions as the evaluation data for *Multi-Domain*. Our policy shows comparable quality compared with the original implementation [3] in training conditions as reported in Table II of our main paper.

Three transfer models are trained to translate images from the testing weather condition (*daytime hard rain*) to each of the three training conditions for both *CycleGan* and *VR-Goggles*. We follow the default settings of official CycleGAN code[2]. We do not conduct any resizing or cropping during training. The learning rate is 0.0002 for the first 100 and decayed to 0 for the next 100 epochs. The weight of the shift loss is 1000. The domain adaptation takes almost the same time for both *CycleGAN* and *VR-Goggles*, which is almost 15 hours. The last models for all of the setups are used for the benchmark testing. In fact, the visual quality of the generated image is quite stable after 50 epochs as shown in Figure 1, and we don't really need to train the model for 15 hours for applications. However, to compare with the original CycleGAn fairly, we follow their settings for the account of training epoch [5]. For the experiments in the real world (Section IV.C. of the main paper), we train *CycleGAN* and *VR-Goggles* for 50 epochs.

An interesting point in Table II of the main paper is that testing result under the *Multi-Domain* policy is better than in training condition for the *Straight* task. It is also the case in the results presented in the original *Carla* benchmark tests [3].

C. Comparing Policy Transfer Methods: Simulated Indoor Navigation

To illustrate the difference between the policy transfer pipelines of our proposed *real-to-sim* approach and several representative *sim-to-real* approaches we conduct a direct comparison in a simulated indoor navigation experiment.

We build an indoor office environment in *Gazebo* [7]. With two different sets of textures, we render two environments: ***Sim-Env*** and ***"Real"-Env***, shown on the left of Fig. 3. We train agents to learn navigation policies to accomplish the task of navigating to chairs based purely on its front-facing color camera readings; the agent obtained a reward of -0.005 for a step cost, -0.05 for collision, and 1 for reaching the target. We illustrate the procedures of the following approaches to transfer policies learned in *Sim-Env* to *"Real"-Env*, and show the average steps obtained by the agents in evaluation during training in Fig. 3:

(1) ***From-Scratch*** [8]: Canonical A3C trained from scratch on *Sim-Env*. We execute 8 training processes and 1 evaluation process, each with their own copy of *Sim-Env*. We note here that this is all the policy training required for our *real-to-sim* approach, as no fine-tuning or retraining of the policy is needed for deploying it in real-world scenes; also, since our approach decouples the policy training and the domain adaptation, the adaptation networks can be trained in parallel with the policy training. This is also the policy training procedure for several sim-to-real approaches [9], [10], except that an additional adaptation step has to be added for each of the training frames; also, for each visually different real-world scene, this line of approaches needs to go through another complete policy training procedure. (2) ***Multi-Domain*** (inspired by [11], [12]): Our A3C variant of the *domain randomization* approach. The key concept of *domain randomization* [11], [12] is randomizing the textures, viewing angles, etc. of objects during the training in simulation, such that when deploying the trained model in real-world scenarios, the modality of the real-world objects could just be naturally dealt with as another variation. Here we adopt the same basic idea but implement it under the constraints of our simulator (changing the texture for each frame is not as straightforward in *Gazebo*). In detail, each of the 8 A3C workers renders its environment with its own set of textures. All 8 sets of textures for the various objects in the environment are shown in Fig. 2. The evaluation during training uses *Sim-Env* as its environment. We note here that this is all the policy training required for the *Multi-Domain* approach, since once converged, the agent is expected to be directly deployable in real-world environments. Unfortunately, for our considered task and

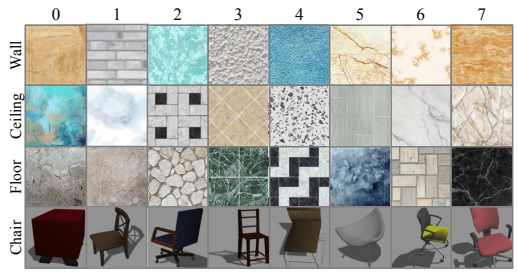

Fig. 2: 8 different sets of textures of the 8 workers for training *Multi-Domain* (Sec. -C).

[2]https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

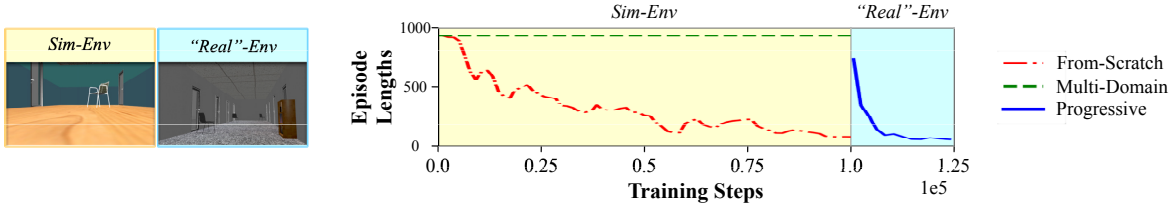

Fig. 3: Average episode lengths in evaluation during training, comparing policy transfer procedures of *From-Scratch*, *Multi-Domain* and *Progressive*. We show median-filtered curves in line with [6]. An adapting phase is necessary for *Progressive* in each new environment; while for our *real-to-sim* approach, *From-Scratch* is all the policy training needed.

setup, our A3C variant of the approach does not manage to learn useful policies, as is shown by the green dashed line in Fig. 3 (we trained it for 8 times the iterations shown but still it fails to converge). We suspect that our variant of randomizing textures might have imposed challenges for our RL task where informative reward signals are sparse. We suspect that more workers with more sets of textures could lead to improved performance. However, with relatively limited computing resources and with simulators where there is no relatively efficient solutions for randomizing the textures of objects at each frame, the *Multi-Domain* method does not learn useful policies. (3) ***Progressive*** [6]: *Progressive Nets* for transferring the policy of *From-Scratch* on *Sim-Env* to *"Real"-Env*. As described in [6], a second column is added after the first column is trained (the *From-Scratch* policy on *Sim-Env*). We note that for this approach, although the adaptation of the policy in new environments is significantly accelerated compared to training from scratch, the overall policy training needed contains both the initial training (the yellow block), and an adapting phase (the cyan blocks) in case we want to deploy the agent in new environments.

Further implementation details for *Progressive*: We follow the same parameter initialization strategy as in [6] for the output layers and the connection layers to guarantee that the initial policy output of the agent is identical to the first column. Observing that the simulation and real-world environments presented in [6] are relatively more visually similar than *Sim-Env* and *Real-Env*, we additionally conduct experiments where we do not incorporate the above parameter initialization strategy, as we suspect that a random initialization might be more beneficial for transfer scenarios where two environments are more visually different. We found out that both the identical and the random initialization works, with the former converging faster. So we only report the experiments with the identical initialization, shown in the blue line in Fig. 3.

D. Comparing Domain Adaptation Methods: Additional Materials

We additionally validate the *shift loss* in the field of *domain adaptation* in *outdoor* urban street scenarios (where we collect synthetic domain images $s \sim p_{\text{sim}}$ from the *CARLA* simulator [3], and realistic domain images $r \sim p_{\text{real}}$ from the *RobotCar* dataset [13]). We compare the following three setups: ***CyCADA*** [14]: *CycleGAN* with semantic constraints, trained on single frames; ***CyCADA+flow***: *CyCADA* with temporal constraints ([2]), trained on sequential frames; ***Ours***: *CyCADA* with *shift loss*, trained on single frames; we refer to this as the *VR-Goggles*.

We pretrain the segmentation network f_S using *Deeplab* [15]. It is worth mentioning that the original *CyCADA* paper did not use the semantic constraint in their experiments due to memory issues. We are able to incorporate semantic loss calculation, by cropping the input in each iteration.

In Fig. 4, we show a comparison of the subsequent frames generated by the three approaches. Our method again achieves the highest consistency and eliminates more artifacts due to the smoothness of the learned model.

Fig. 4: Comparison of the translated images for sequential input frames for the different approaches. $1st$ *row*: two subsequent input frames from the realistic domain, with several representative images from the simulated domain shown in between; $2nd \sim 4th$ *row*: outputs from *CyCADA*, *CyCADA+flow* and *Ours*. Our method is able to output consistent subsequent frames and eliminate artifacts. We adjust the brightness of some zoom-ins for visualization purposes.

An additional implementation detail for all our *domain adaptation* experiments: As a naive random crop would

highly likely lead to semantic permutations, we crop inputs of the two domains in the same training iteration from the same random position, and our empirical results show that this greatly stabilizes the adaptation.

REFERENCES

[1] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *ECCV*, 2016, pp. 694–711.

[2] H. Huang, H. Wang, W. Luo, L. Ma, W. Jiang, X. Zhu, Z. Li, and W. Liu, "Real-time neural style transfer for videos," in *CVPR*, 2017, pp. 783–791.

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference of Robot Learning*, 2017, pp. 1–16.

[4] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *International Conference on Robotics and Automation*, 2018, pp. 1–9.

[5] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *CVPR*, 2017, pp. 2223–2232.

[6] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Conference on Robot Learning*, 2017, pp. 262–270.

[7] N. Koenig, B. A, and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IROS*, vol. 3. IEEE, 2004, pp. 2149–2154.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016, pp. 1928–1937.

[9] 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," *arXiv preprint arXiv:1709.07857*, 2017.

[10] G. J. Stein and N. Roy, "Genesis-rt: Generating synthetic images for training secondary real-world tasks," in *ICRA*. IEEE, 2018, pp. 7151–7158.

[11] F. Sadeghi and S. Levine, "CAD2RL: real single-image flight without a single real image," in *Robotics: Science and Systems*, 2017.

[12] 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 *IROS*. IEEE, 2017, pp. 23–30.

[13] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 Year, 1000km: The Oxford RobotCar Dataset," *The International Journal of Robotics Research (IJRR)*, vol. 36, no. 1, pp. 3–15, 2017.

[14] J. Hoffman, E. Tzeng, T. Park, J. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," pp. 1994–2003, 2018.

[15] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs." *IEEE transactions on pattern analysis and machine intelligence*, 2017.