Supplementary Material

DTA: Physical Camouflage Attack using Differentiable Transformation Network

A. Overview

In this supplementary material, we describe our detailed algorithm, experiment setup, evaluation details, and evaluation results that can not be included in the main paper due to limited space. Furthermore, we also provide additional experiments and more evaluation samples in both simulated environments and real world.

B. Algorithm Details

In this section, we provide detailed algorithm and figures about the training process of the Differentiable Transformation Network (DTN), our proposed texture rendering method. Algorithm 1 and Figure 1 describe the detailed training process of the DTN.

Algorithm 1: Training process of the DTN.

Input: Transformation set $T = \{t^{(1)}, \ldots, t^{(M)}\}$, Flat color texture set $C = \{c^{(1)}, \ldots, c^{(N)}\}$, Rendering function R, Segmentation function S
Output: DTN f_w
(1) Export X_{orig} and X_{seg} from the photo-realistic rendering engine
for $m = 1$ to M **do**
  $X_{seg}^{(m)} \leftarrow S(t^{(m)})$
  for $n = 1$ to N **do**
    $X_{orig}^{(m)(n)} \leftarrow R(t^{(m)}, c^{(n)})$
  end for
end for
(2) Dataset masking and pairing
for $m = 1$ to M **do**
  for $n = 1$ to N **do**
    $X_{ref} \leftarrow X_{ref} \cup (X_{orig}^{(m)(1)} \times X_{seg}^{(m)})$
    $H_{exp} \leftarrow H_{exp} \cup (c^{(n)} \times X_{seg}^{(m)})$
    $X_{ren} \leftarrow X_{ren} \cup (X_{orig}^{(m)(n)} \times X_{seg}^{(m)})$
  end for
end for
(3) DTN Training
Initialize f_w with random weights w
for number of training iterations **do**
  Sample minibatch of each b samples $x_{ref} \in X_{ref}$, $\eta_{exp} \in H_{exp}$, $x_{ren} \in X_{ren}$
  Update f_w with gradient descent based on the binary cross-entropy loss between $f_w(x_{ref}, \eta_{exp})$ and x_{ren}
end for

C. Experiment Details

C.1. Dataset Details

For DTN Training We generate the datasets used for training and evaluating DTN model using CARLA Simulator. We use Toyota Camry as our target object, i.e., where the adversarial pattern will be generated and evaluated. We select a map (Town03 in CARLA) and randomly choose 75 spawn locations to generate both training and validation datasets, and choose 50 spawn locations for generating testing datasets. For every location, we spawn the target car and capture the photo realistic

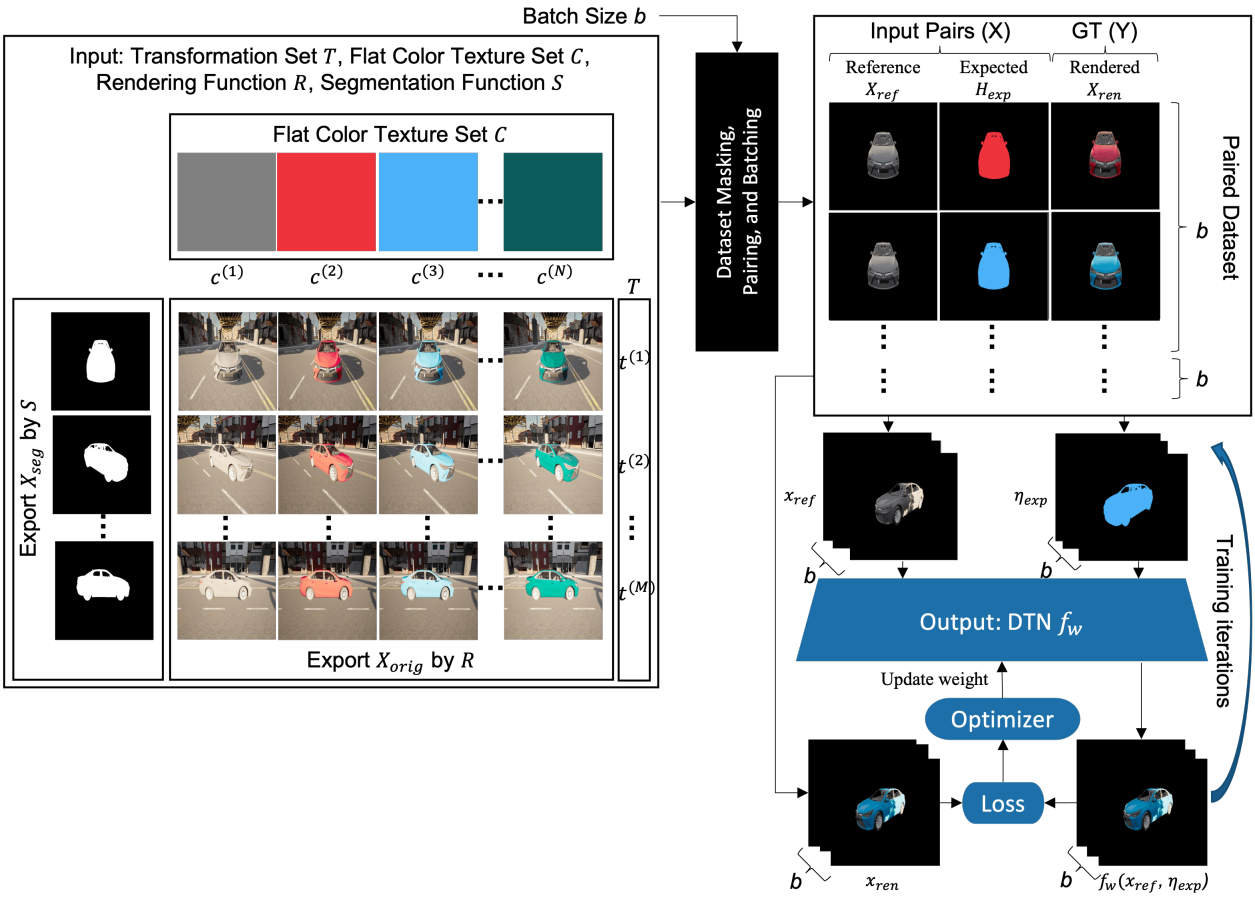


Figure 1. Training process of DTN.

images using cameras with 5 m distance, 15-degree pitch, and for every 45-degree rotation. For each spawn location and camera pose, we sequentially change the car texture with 50 random flat color textures and get a segmentation image. The segmentation image is used for extracting either target object or background as needed by DTN. Datasets used for training and testing use different spawn locations and different random color textures. The datasets are then grouped by the same transformation and same texture as shown in Figure 1.

For Adversarial Camouflage Generation We select 250 spawn locations in the same map and generate datasets of the target object with the flat color texture used as a reference image during DTN training. The camera pose used for adversarial pattern generation is the same camera pose used during DTN training.

For Adversarial Pattern Evaluation We use more camera poses and different towns to evaluate whether the generated attack pattern is robust enough to generalize the attack performance on untrained transformation distribution. We selected 200 spawn locations from four simulated towns (Town01, Town04, Town05, and Town06 in CARLA) with minimum occlusions. For each location, we spawn the textured car and capture images with camera settings: distance (5 m, 10 m, 15 m), pitch angles (0°, 15°, 30°), and a 30-degree rotation interval as described in Figure 2.

C.2. Implementation Details of Compared Methods

We provide detailed parameter setup of compared methods: CAMOU [9], ER [8], UPC [3] and DAS [7] in our experiment.

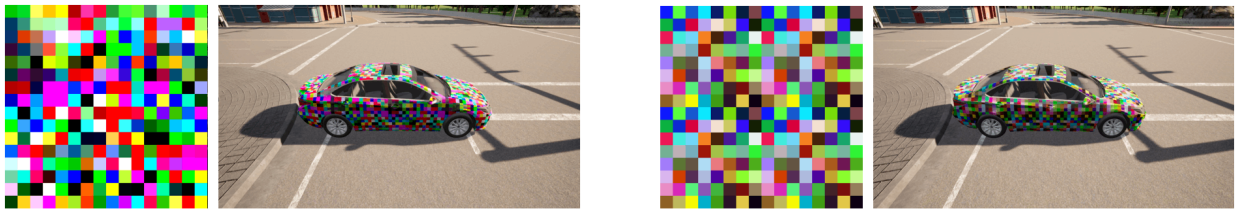
Repeated Pattern-based Physical Camouflage CAMOU and ER produce the repeated patterns covering all target object surfaces as the physical camouflage output. We select the same 16×16 attack resolution following the best CAMOU result to conduct a fair comparison. We closely follow the approach to replicate the original papers, but we rebuild the environment and target models based on our evaluation setup (Generating attack pattern on Toyota Camry and targeting both EfficientDetD0 and YOLOv4). For CAMOU, we use the same clone network architecture and other parameters as the original paper for

(a) Various camera angle

(b) Various camera pitch

Figure 2. Camera pose distribution for adversarial camouflage evaluation.

generating the camouflage pattern. For ER, we also use the same parameters as the original paper, except that we change $p = 3$ and $r = 1$ parameters so that it outputs a same 16×16 texture.



(a) CAMOU [9]

(b) Enlarge and Repeat (ER) [8]

Figure 3. Repeated pattern-based physical camouflage from original CAMOU [9] and ER [8] when re-implemented in our environment. For each Figure (a) and (b), left side shows the generated camouflage pattern, while the right side is the result of a car rendered by repeated texture using world-aligned texture.

Patch-based Physical Camouflage UPC and DAS produce patches that are attached to specific target object surface as the physical camouflage. Since they have different settings to recreate in our environment, we only evaluate them on the transferability experiment. We run their official code and keep the original attack setup on cars, and then extract their attack patches to our environment. We use decals in Unreal for painting their patches to our car surfaces. To conduct a fair comparison, we paint the patches into car hood, doors, rooftop, and back so that they can be seen from various angles. Figure 4 shows the sample of how we re-implement the generated patches to our transferability test environment.

(a) Universal Physical Camouflage (UPC) [3]

(b) Dual Attention Suppression (DAS) Attack [7]

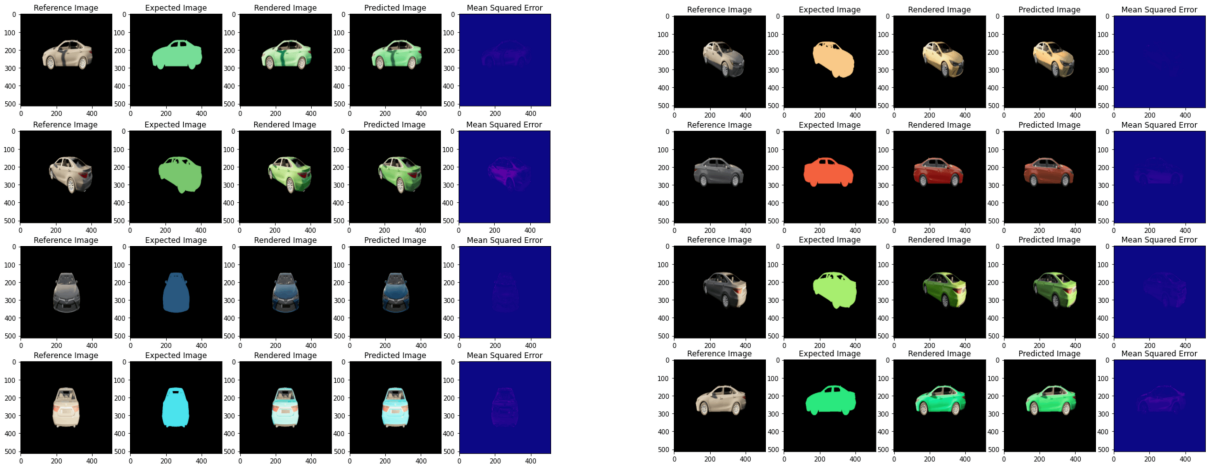
Figure 4. Transferability: implementation of original camouflage of UPC [3] and DAS [7] in our environment. For each (a) and (b), the left and right side shows the sample of the original paper and in our test environment, respectively.

D. Evaluation Details

D.1. DTN Evaluation

Prediction results We visualize the sample prediction results of DTA on unseen data after the training is complete. Figure 5 shows sample prediction result of trained DTA on validation and test datasets. As shown, the model can learn how to render the expected image given the inputs. Apart from learning the texture color changes caused by object material and lighting,

DTA can also extract parts that do not need to be changed from reference images, such as tires, car headlights, interiors, license plate, etc. This provides convenience to the user without the need to separate the masking from the part where the texture will be applied.



(a) Prediction on Validation set

(b) Prediction on Test set

Figure 5. DTA sample predictions on unseen data

Visualizing Transformation Features We visualize the output of each encoded transformation features TF and how it is used to transform the expected texture η_{exp}. On Fig. 6, it can be observed that the first subtractor and adder TF encode the color transformation including the part where the texture is not applied (e.g., wheels, interior, etc). TF multiplier encodes how the texture is minimized or emphasized. Since the image is visualized in RGB format, white or gray color indicate that the proportion between each color channel is similar. Additionally, TF multiplier also removes the nonapplied texture part. Finally, the final TF adder gives the final touch for transforming the expected texture to the rendered version, which includes adding the nonapplied texture part from the reference image.

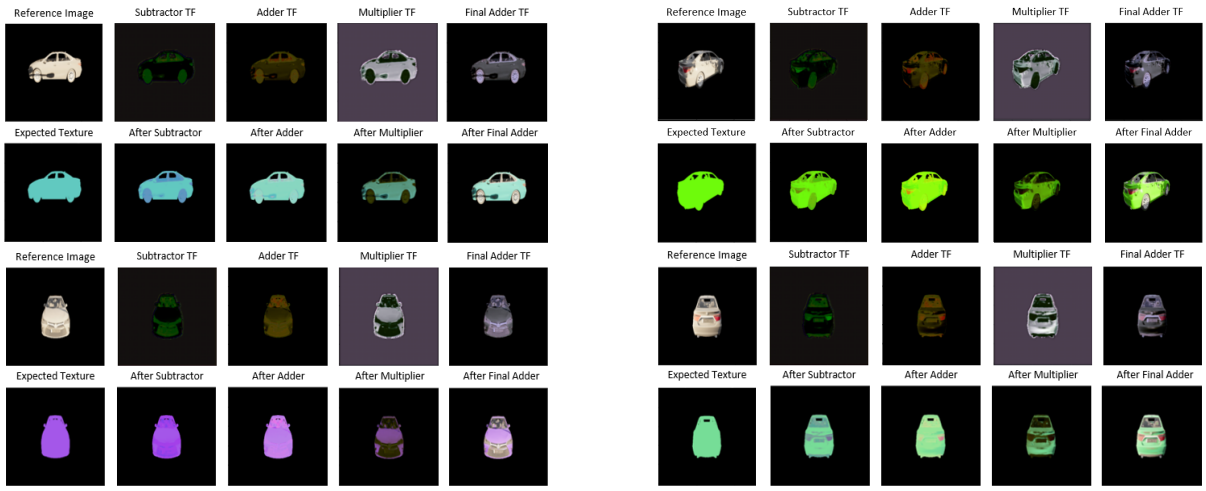


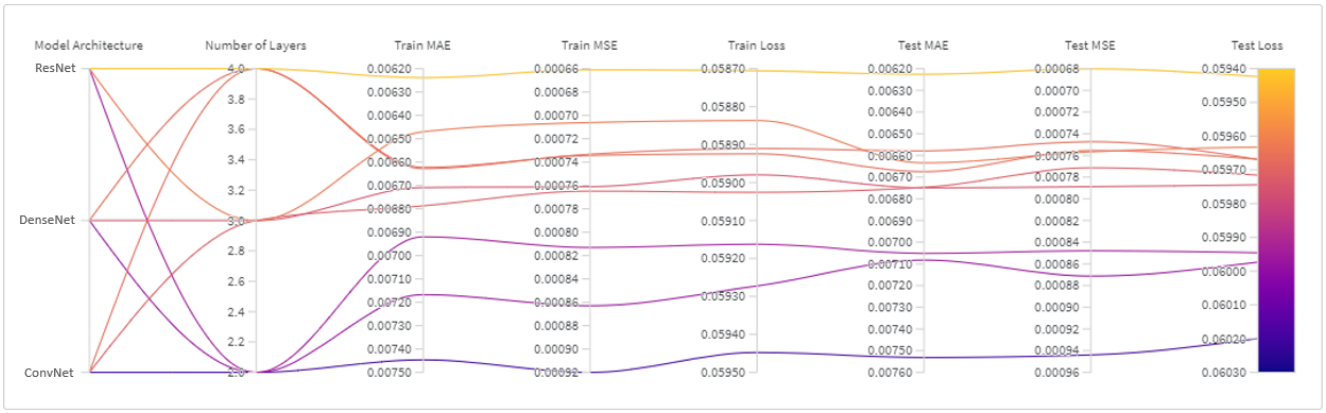
Figure 6. Visualization of each transformation process.

Different Architectures We further evaluate and visualize the performance of DTN with different architectures [ResNet, DenseNet, Plain CNN] and $k = [2, 3, 4]$ layers. We run the evaluation five times and use the same unique random seeds for each run on different model. The evaluation results can be seen in Figure 7.

Figure 7a shows parallel coordinate chart for evaluating the effect of different architectures (first column) and number of layers (second column) with respect to train and test evaluation (i.e., Mean Absolute Error, Mean Squared Error, and Loss, respectively). The values are the mean over multiple runs. All evaluation metrics columns are sorted where brighter colors gradient (yellow) are better.

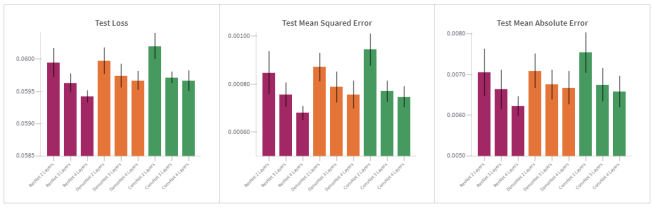
Figure 7b and 7c show the detailed training and test evaluation bar plot for every architecture (ResNet [Red], DenseNet [Orange], and Plain CNN [Green]) with different number of layers k. The bar contains the mean and standard deviation information over multiple runs.

Figure 7d and 7e describe grouped loss, including train/validation loss for each epochs and final test loss. The difference is whether the loss is grouped by architecture or number of layers. This shows the effect of each parameter on the evaluation metric more clearly.

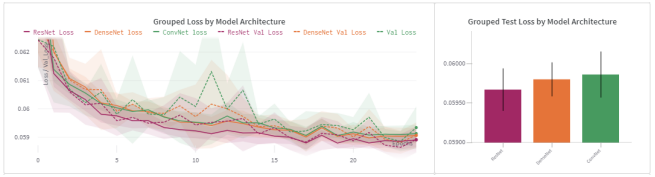


(a) Parallel coordinates charts for evaluating effect of different model architecture and number of layers with respect to evaluation metrics.

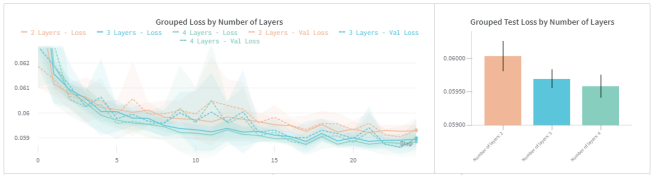
(b) Training Loss, Mean Squared Error, and Mean Absolute Error.



(c) Test Loss, Mean Squared Error, and Mean Absolute Error.



(d) Grouped Loss by Model Architecture.

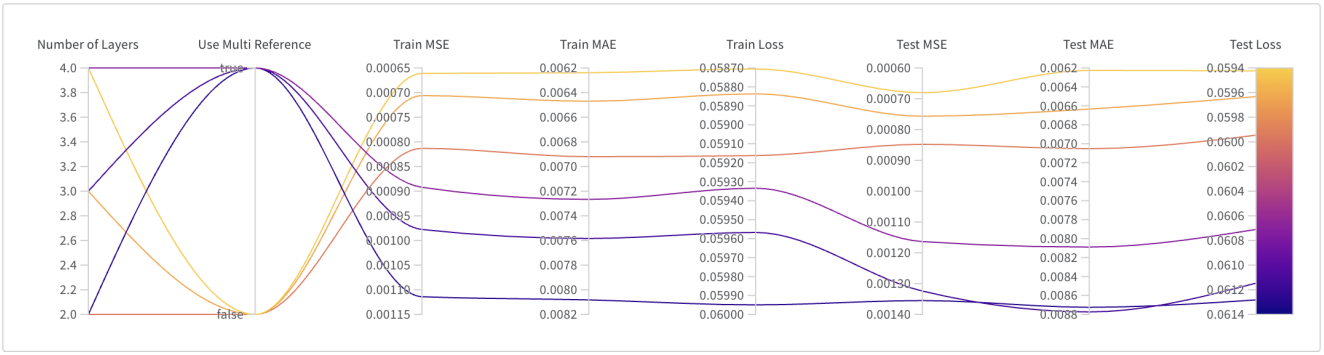


(e) Grouped Loss by Number of Layers k.

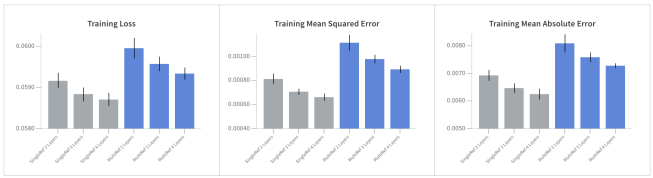
Figure 7. DTN evaluation with different architectures [ResNet (red), DenseNet (orange), ConvNet (green)] and number of layers $k = [2, 3, 4]$.

Different Reference Types. We select the best-performing model and re-evaluate DTN by varying the training configuration such as whether to use single fixed or multiple flat color texture for the reference input.

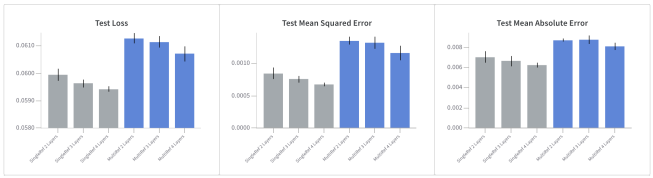
Figure 8 shows the evaluation of DTN with different reference types [SingleRef or MultiRef] and $k = [2, 3, 4]$ layers. Figure 9 shows sample prediction result of trained model between single/fixed-reference vs multi-reference. Overall figures show that training DTN with single-reference texture is better than using multi-reference texture. Learning the transformation from a single fixed color texture reference is considered as a simpler task. Even though we can increase the number of layers for lowering the loss, our final goal of training DTN is for generating the adversarial pattern. In this case, using a single reference color image as the reference dataset is considered the best practice as it has a lower reconstruction loss.



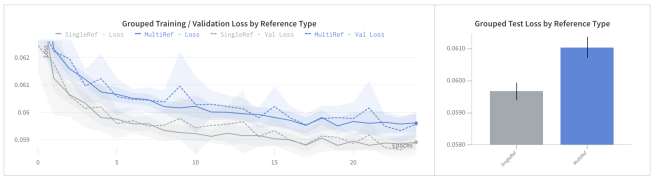
(a) Parallel coordinates charts for evaluating effect of different reference type and number of layers respected to evaluation metrics.



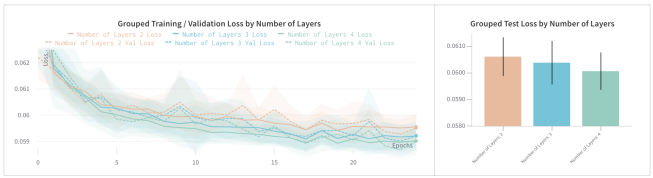
(b) Training Loss, Mean Squared Error, and Mean Absolute Error.



(c) Test Loss, Mean Squared Error, and Mean Absolute Error.

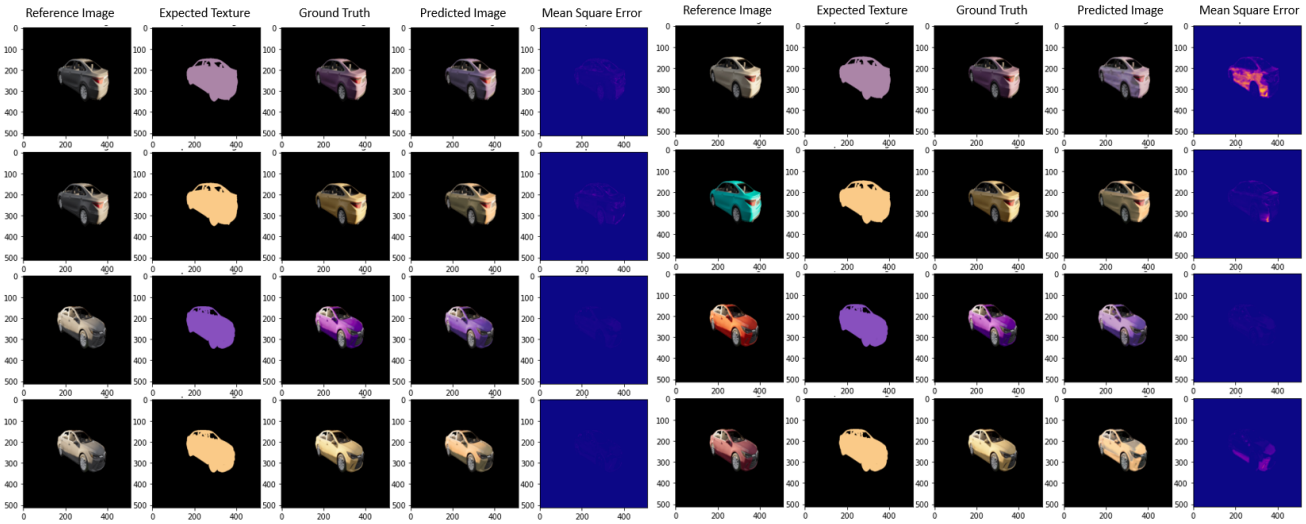


(d) Grouped Loss by Reference Types.



(e) Grouped Loss by Number of Layers k.

Figure 8. DTN evaluation with different reference types [SingleRef (gray), MultiRef (blue)] and number of layers $k = [2, 3, 4]$.



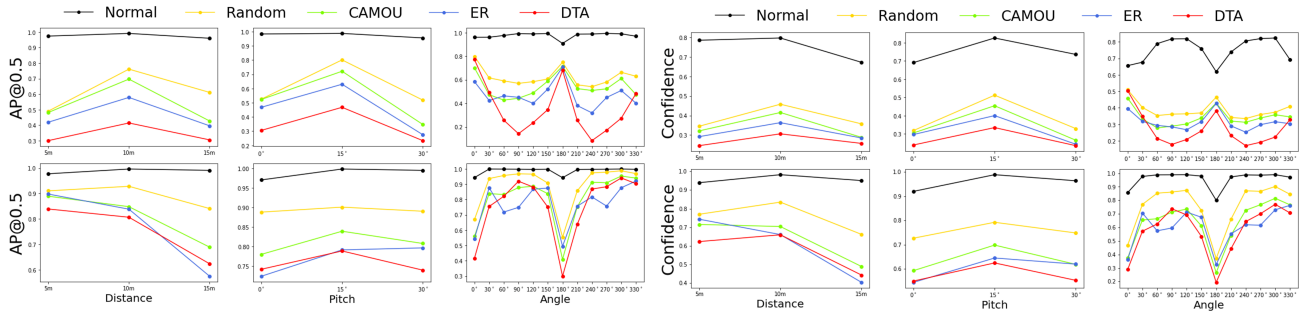
(a) Prediction on Single/Fixed Reference Trained Model

(b) Prediction on Multi Reference Trained Model

Figure 9. DTN sample predictions for single/fixed-reference vs multi-reference model. Both models have the same parameters except for the type of reference used.

D.2. Adversarial Camouflage Evaluation

Targeted Model Evaluation We perform comparative experiments to evaluate our adversarial camouflages with previous methods by attacking the COCO pre-trained EfficientdetD0 [6] and YOLOv4 [1]. Figure 10 describes adversarial camouflages comparison on different camera poses. From the figure, it can be inferred that our attack pattern is more robust with various transformations compared to other textures. Figure 11 shows sample images of each adversarial camouflage on different camera poses. In the figures, red and green border represents miss-detection and correct detection, respectively, while yellow border signifies partially-correct detection (i.e., other labels are also detected).



(a) Target Average Precision@0.5

(b) Target Average Confidence Score

Figure 10. Adversarial camouflage comparison on CARLA Simulator with different camera poses. First row: EfficientDetD0, second row YOLOv4

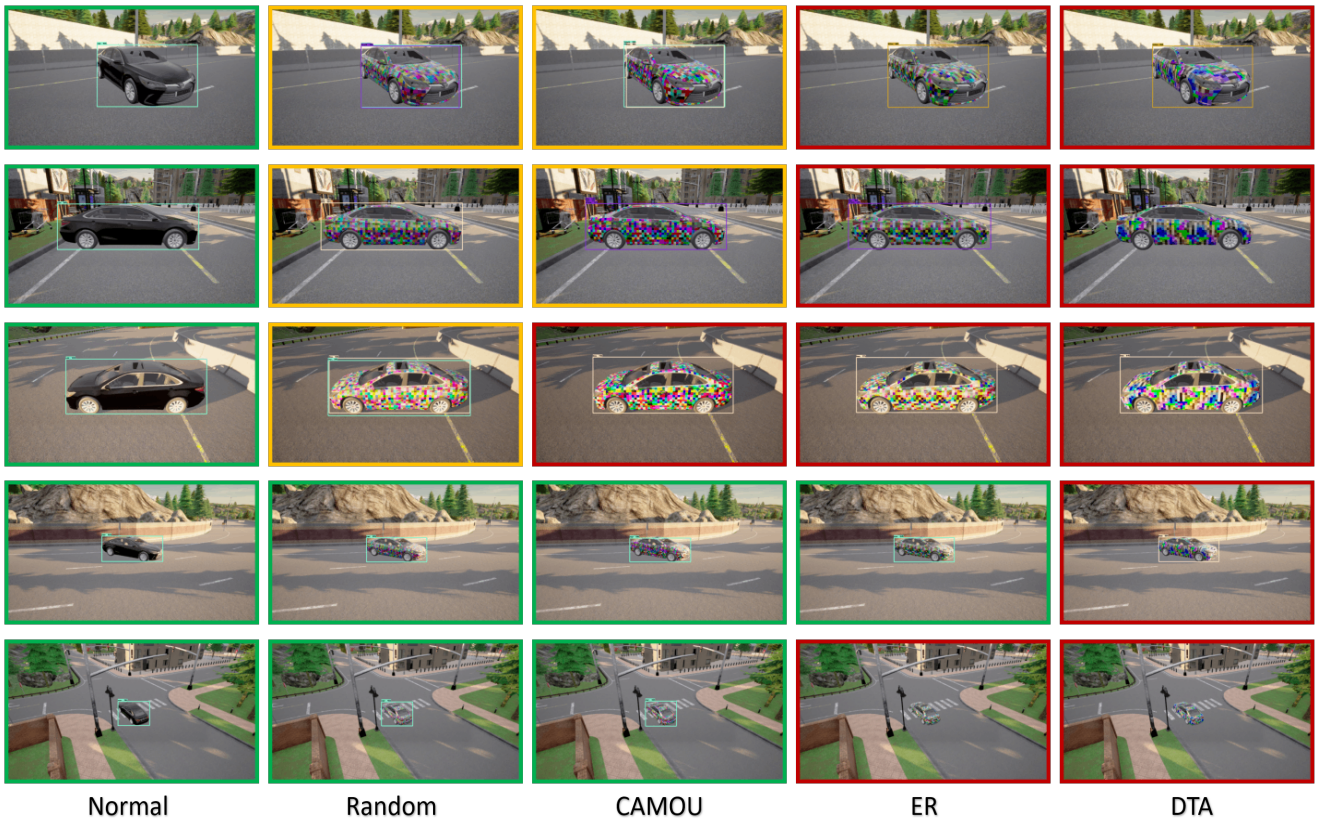


Figure 11. Sample images of adversarial camouflage on CARLA Simulator with different camera poses. Border: Red = miss-detection; Yellow = partially correct (other label also detected); Green = correct (detected as car).

Transferability Evaluation We further perform comparative experiments to evaluate our adversarial camouflages with previous methods in transferability setting which target object, transformations, and target model are different during the camouflage generation phase. The target models in this evaluation are COCO pre-trained SSD [4], Faster R-CNN [5], and Mask-RCNN [2]. Figure 12 presents attack transferability evaluation results by transformation(distance, pitch and angle). From the figure, we can infer that our attack pattern is still more robust in the transferability setting compared to others. Furthermore, Figure 13 shows sample images of each adversarial camouflage on different camera poses and target model.

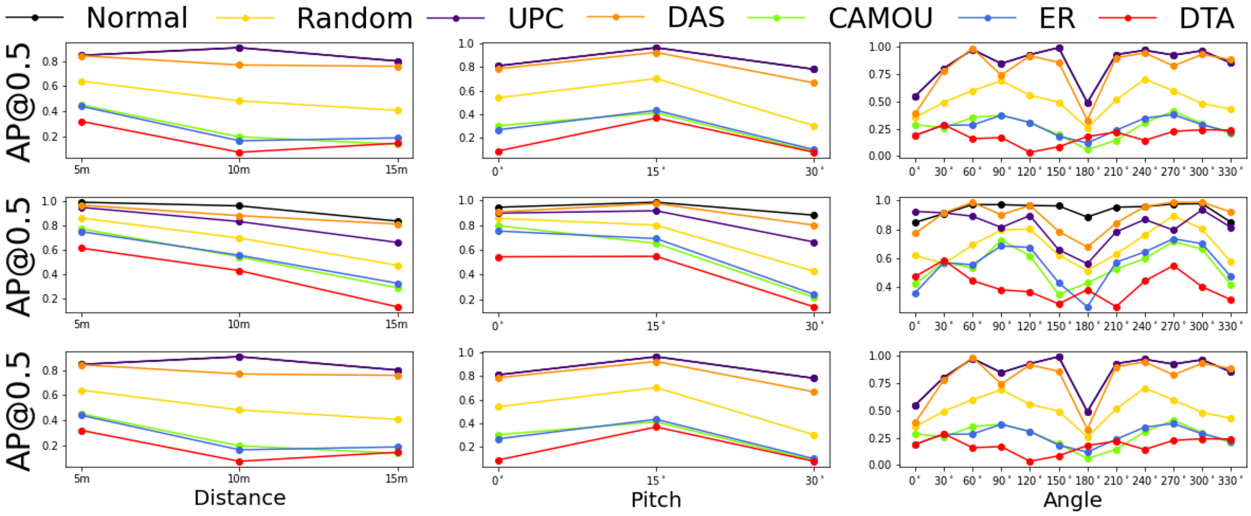
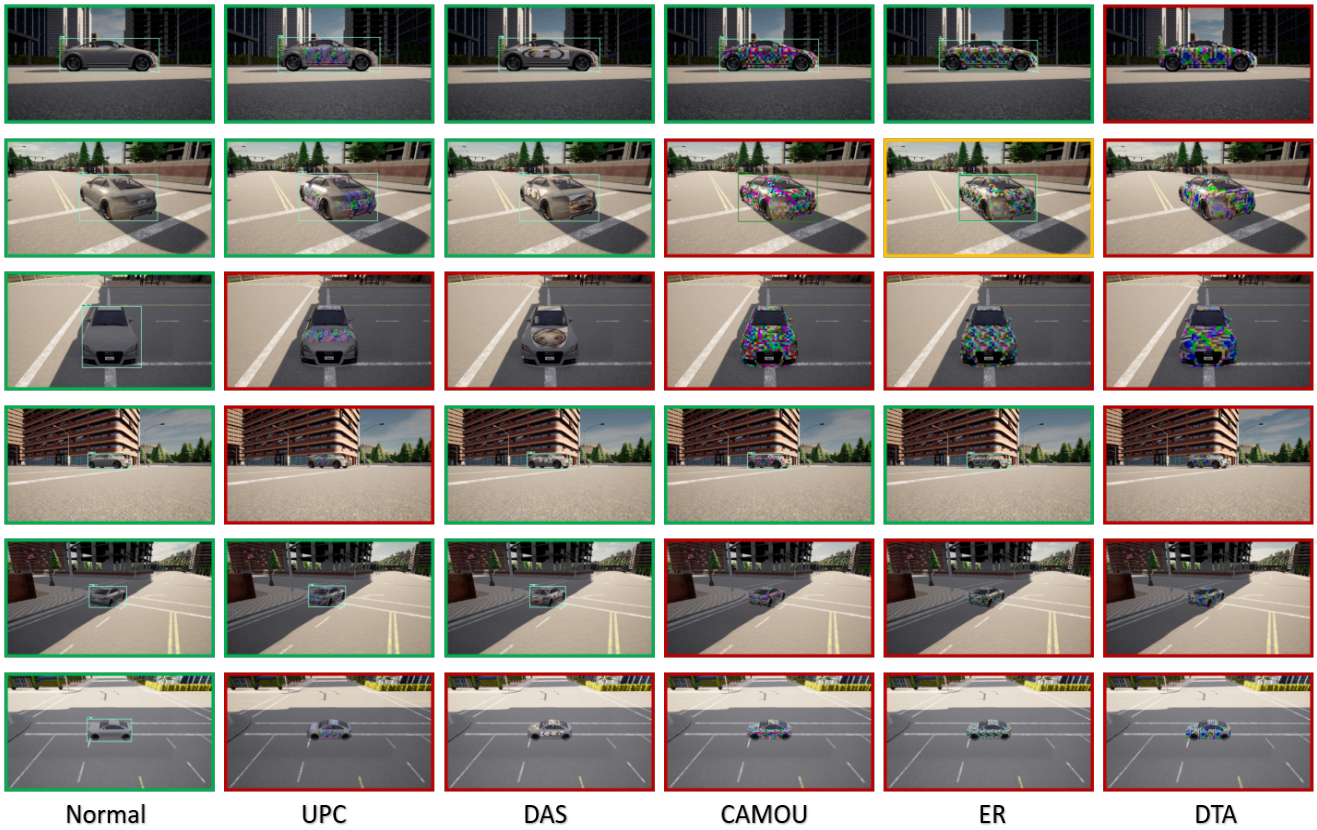
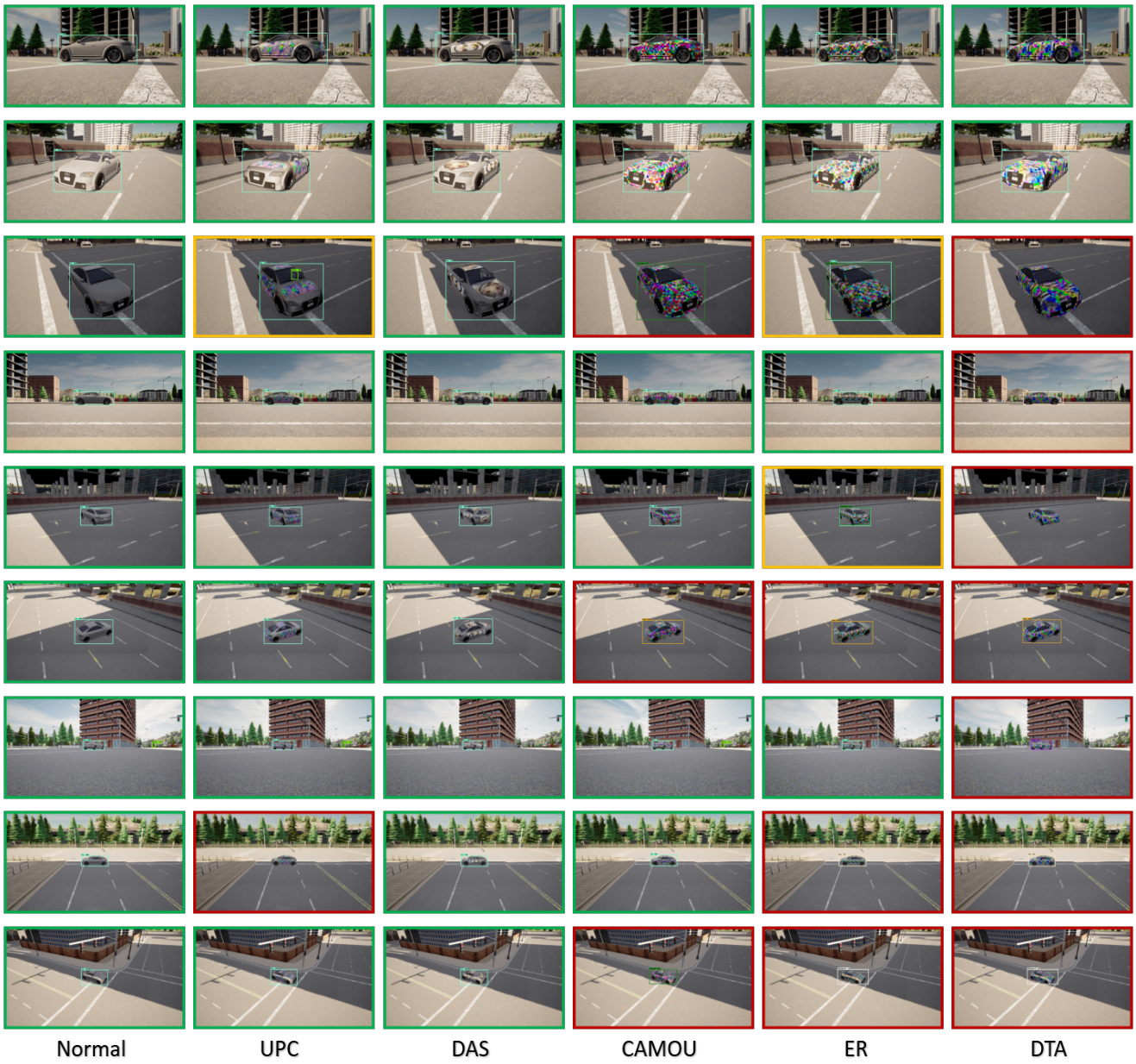


Figure 12. Attack transferability results by transformations. First row: SSD, second row: Faster R-CNN, third row: Mask R-CNN



(a) SSD model predictions

(b) Faster R-CNN model predictions



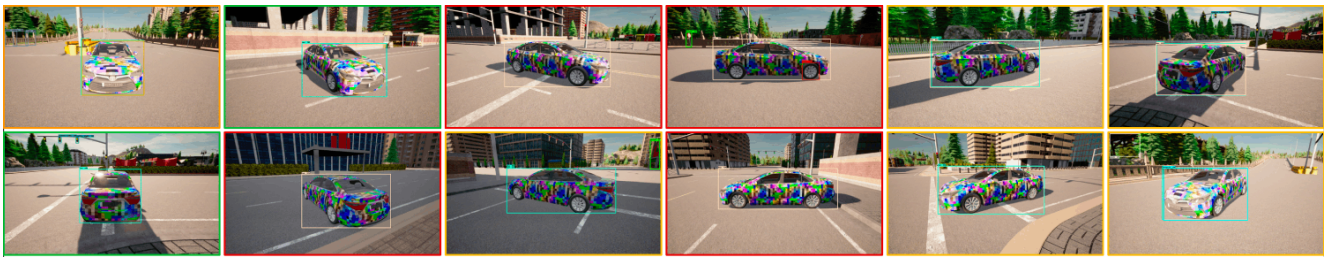
(c) Mask R-CNN model predictions

Figure 13. Transferability: Sample images of adversarial camouflage on CARLA Simulator with different camera poses. Border: Red = miss-detection; Yellow = partially correct (other label also detected); Green = correct (detected as car).

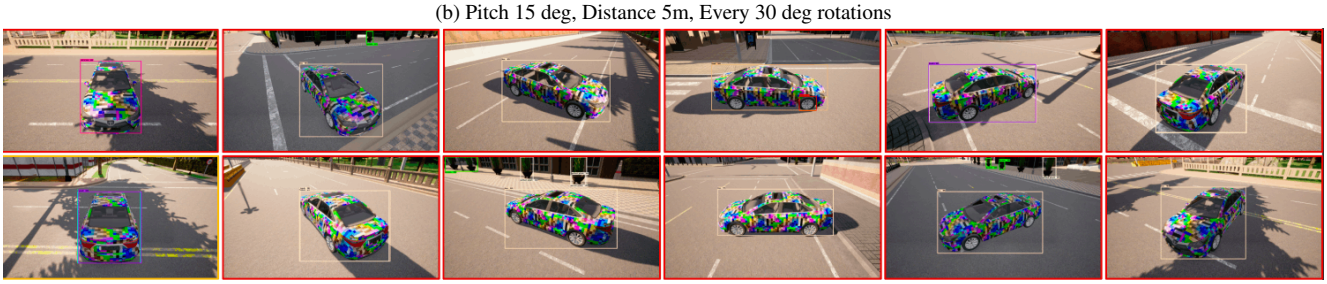
Detail Visualization of our Camouflage on Simulated Environment We provide more detailed samples for showing how our camouflage performance under a variety of camera poses and locations in a simulated environment. Figure 14 shows the evaluation of our camouflage pattern on CARLA Simulator, by varying value of pitch, camera distance, and for every thirty degree rotations.

Detail Visualization in the Real World We provide more detailed samples for showing how our camouflage performance in the real world. Figure 15 shows our evaluations on a lifelike miniature of Tesla Model 3. In particular, Figure 15a shows the detection for normal situation (i.e., normal version of Tesla Model 3), whereas Figure 15b shows the detection when adversarial attack is in operation.

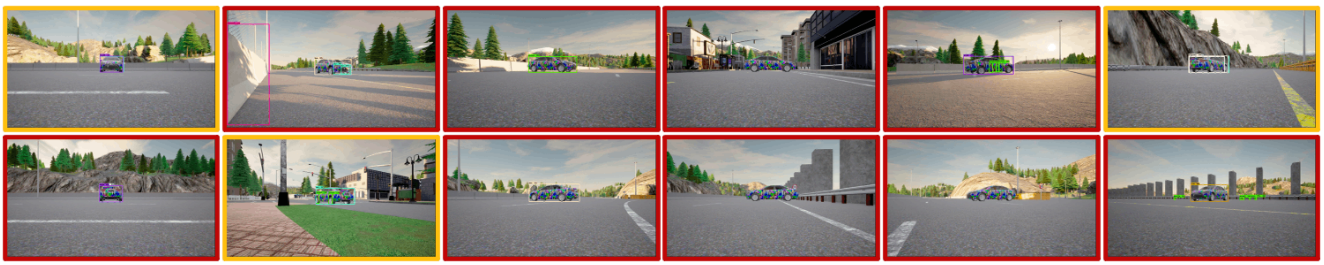
(a) Pitch 0 deg, Distance 5m, Every 30 deg rotations



(b) Pitch 15 deg, Distance 5m, Every 30 deg rotations

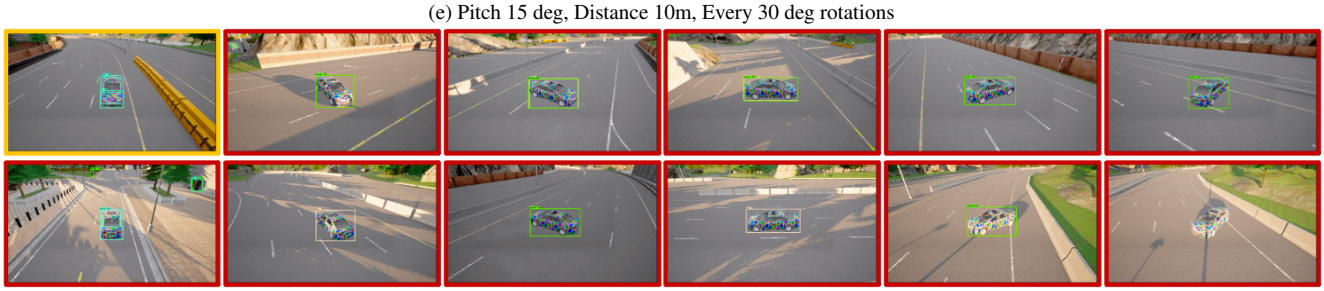


(c) Pitch 30 deg, Distance 5m, Every 30 deg rotations



(d) Pitch 0 deg, Distance 10m, Every 30 deg rotations

(e) Pitch 15 deg, Distance 10m, Every 30 deg rotations



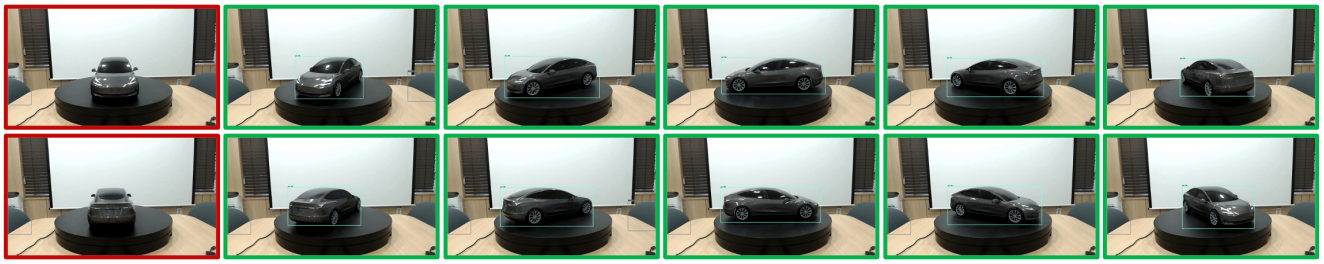
(f) Pitch 30 deg, Distance 10m, Every 30 deg rotations

(g) Pitch 0 deg, Distance 15m, Every 30 deg rotations

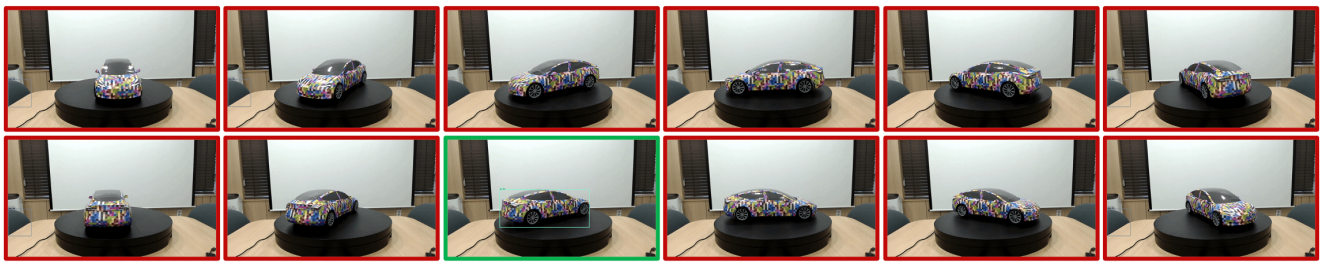
(h) Pitch 15 deg, Distance 15m, Every 30 deg rotations

(i) Pitch 30 deg, Distance 15m, Every 30 deg rotations

Figure 14. Our Adversarial Camouflage Evaluation on CARLA Simulator. Border: Red = miss-detection; Yellow = partially correct (other label also detected); Green = correct.



(a) Normal Tesla Model 3 Evaluation



(b) Attacked Tesla Model 3 Evaluation

Figure 15. Our Adversarial Camouflage Evaluation on Real World. Border: Red = miss-detection; Yellow = partially correct (other label also detected); Green = correct.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. 7

[2] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 8

[3] Lifeng Huang, Chengying Gao, Yuyin Zhou, Cihang Xie, Alan L. Yuille, Changqing Zou, and Ning Liu. Universal physical camouflage attacks on object detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 720–729, June 2020. 2, 3

[4] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 8

[5] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015. 8

[6] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, 2020. 7

[7] Jiakai Wang, Aishan Liu, Zixin Yin, Shunchang Liu, Shiyu Tang, and Xianglong Liu. Dual attention suppression attack: Generate adversarial camouflage in physical world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8565–8574, 2021. 2, 3

[8] Tong Wu, Xuefei Ning, Wenshuo Li, Ranran Huang, Huazhong Yang, and Yu Wang. Physical adversarial attack on vehicle detector in the carla simulator. *CoRR*, abs/2007.16118, 2020. 2, 3

[9] Yang Zhang, Hassan Foroosh, Philip David, and Boqing Gong. Camou: Learning physical vehicle camouflages to adversarially attack detectors in the wild. In *International Conference on Learning Representations*, 2018. 2, 3