Generating Transferable Adversarial Simulation Scenarios for Self-Driving via Neural Rendering

Yasasa Abeysirigoonawardena
University of Toronto
yasasa@cs.toronto.edu

Kevin Xie
University of Toronto
kevincxie@cs.toronto.edu

Chuhan Chen
Carnegie Mellon University
chuhanc@andrew.cmu.edu

Salar Hosseini Khorasgani
University of Toronto
salar.hosseinikhorasgani@mail.utoronto.ca

Ruiqi Wang
Stanford University
rqwang@stanford.edu

Florian Shkruti
University of Toronto
florian@cs.toronto.edu

***Abstract*—Self-driving software pipelines include components that are learned from a significant number of training examples, yet it remains challenging to evaluate the overall system's safety and generalization performance. Together with scaling up the real-world deployment of autonomous vehicles, it is of critical importance to automatically find simulation scenarios where the driving policies will fail. We propose a method that efficiently generates adversarial simulation scenarios for autonomous driving by solving an optimal control problem that aims to maximally perturb the policy from its nominal trajectory. Given an image-based driving policy, we show that we can inject new objects in a neural rendering representation of the deployment scene, and optimize their texture in order to generate adversarial sensor inputs to the policy. We demonstrate that adversarial scenarios discovered purely in the neural renderer (surrogate scene) can often be successfully transferred to the deployment scene, without further optimization. We demonstrate this transfer occurs both in simulated and real environments, provided the learned surrogate scene is sufficiently close to the deployment scene.**

I. Introduction

Safety certification of a self-driving stack would require driving hundreds of millions of miles on real roads, according to [1], to be able to estimate miles per intervention with statistical significance. This could correspond to decades of driving and data collection. Procedural generation of driving simulation scenarios has emerged as a complementary approach for designing unseen test environments for autonomous vehicles in a cost-effective way. Currently, generation of simulation scenarios requires significant human involvement, for example to specify the number of cars and pedestrians in the scene, their initial locations and approximate trajectories [2], as well as selection of assets to be added to the simulator. In addition to being challenging to scale, having a human in the loop can result in missing critical testing configurations.

In this paper, we cast adversarial scenario generation as a high-dimensional optimal control problem. Given a known image-based driving policy that we want to attack, as well as the dynamics of the autonomous vehicle, we aim to optimize a photorealistic simulation environment such that it produces sensor observations that are 3D-viewpoint-consistent, but adversarial with respect to the policy, causing it to deviate from its nominal trajectory. The objective of the optimal control problem is to maximize this deviation through plausible perturbations of objects in the photorealistic environment.

Our optimal control formulation requires differentiation through the sensor model in order to compute the derivative of the sensor output with respect to the underlying state perturbation. However, most existing photorealistic simulators for autonomous vehicles are not differentiable; they can only be treated as black boxes that allow forward evaluation, but not backpropagation. Instead of using an off-the-shelf photorealistic simulator and adding assets to match the scene, we train an editable neural rendering model that imitates the deployment scene, allowing us to insert new objects in the simulator and to optimize their texture through gradient-based optimization. This editable neural rendering model acts as a surrogate physics and rendering simulator, enabling us to differentiate through it efficiently in order to attack the driving policy's input sensor observations.

II. Related Work

Adversarial scenarios for autonomous driving. Perceptual adversarial attacks make modifications to prerecorded sensor data from real driving sessions to fool the perception system. Since this sensor data is fixed, they lack the ability to resimulate and typically only operate on the individual frame level. Previous works, [3, 4] attempt to attack a LiDAR object detection module by artificially inserting an adversarial mesh on top of car rooftops or objects in a prerecorded LiDAR sequence. They extend the scope of their attack further by incorporating textures to be able to attack image-based object detectors as well [5]. In both these works, the inserted object has a very low resolution and nondescript geometry. Recent self-driving simulators, such as DriveGAN [6], GeoSim [7] and UniSim [8] address these issues, with the latter enabling manipulable sensor-based simulators based on prerecorded datasets. These works, however, have not dealt with discovering attacks.

Another prominent line of works produce dynamic state-level adversarial attacks. These generally target the control/planning system only by perturbing trajectories of other

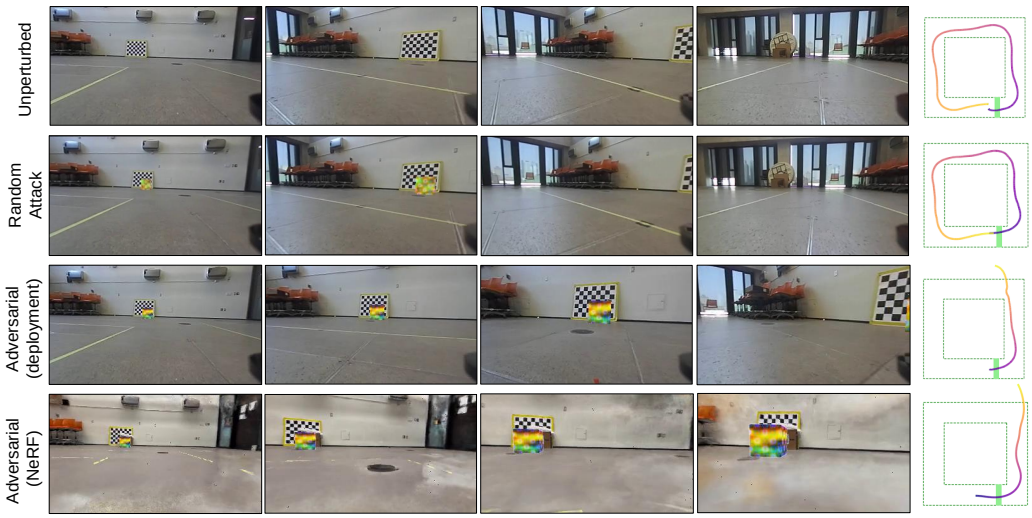


Fig. 1: First-person-view (FPV) of our adversarial attack transfer to an RC car with overhead trajectory view on the right. Row 1: Unperturbed policy execution; Row 2: Random search texture attack; Row 3: Our adversarial attack directly transferred to the real deployment scene, without additional optimization; Row 4: Our adversarial attack discovered in the surrogate NeRF simulator.

agents in the scene. Without considering the perception system, these methods use simplified traffic and state-based simulators that do not incorporate 3D rendering [9, 10, 11].

Closest to our work, a few methods have proposed to attack end-to-end self-driving policies that perform both perception and control. To this end, adversarial perturbations are made to existing self-driving simulators, primarily CARLA. In [12], the trajectories of other agents in a CARLA scene are modified to generate a collision event. Due to the non-differentiability of the simulator, a black-box Bayesian optimization is used to search for successful attacks. Gradient-based attacks on top of simulators have also been investigated. However, the requirement of differentiability has so far limited their scope to very simplified geometries that are composited post-hoc onto renderings from CARLA. In [13], flatly colored rectangles are composited on top of frames from the CARLA simulator and optimized to cause maximal deviation of an end-to-end image-based neural network steering controller. Similarly, work in [14] attempts to play a video sequence of adversarial images on a billboard in the scene using image composition. To our knowledge, no works in this setting have been able to demonstrate transfer of adversarial attacks to the real world.

III. Method

Our framework generates successful adversarial attacks of end-to-end image-based self-driving policies with only access to posed images from the deployment scene. An overview of the high-level steps in our framework is shown in Figure A.2.

We now briefly describe the setting and our adversarial attack method. More details are included in Appendix A. Let x_t denote the state of the car at time t, x^* denote a reference trajectory to track and CTE the cross-track error.

Our optimization problem is as follows:

$$\min_{\theta} J(\theta) = \sum_{t=0}^{T} C(x_t) \text{ such that } G(x_{t-1}, x_t, \theta) = 0 \quad (1)$$

Where we set the cost function $C(x_t)$ of our problem as the car's proximity to the reference x^*:

$$C(x_t) = -\text{CTE}(x_t, x^*) \quad (2)$$

In other words, we want to maximize deviation from the desired trajectory. We set the constraint function $G(x_t, x_{t+1}, \theta) = 0$ to be the following set of constraints:

$$u_t = \pi_\phi(o_t) \quad (3)$$

$$o_t = h_{\gamma,\theta}(x_t) \quad (4)$$

$$x_{t+1} = f_c(x_t, u_t) \quad (5)$$

Where π is the fixed driving policy[1], h is the neural rendering sensor model that outputs image observations o_t given the state of the car. The renderer depends on θ, the parameters of adversarial NeRF objects and γ, the fixed rendering parameters of the background scene NeRF. Finally, f_c denotes the dynamics of the ego vehicle that must be considered, since we want to find adversarial trajectories that are consistent across multiple frames.

A. Differentiable Renderer

Traditional simulators like CARLA do not admit computation of gradients. Thus, prior works rely on artificially compositing simplistic textured geometries on top of rendered

[1]We train our own policy and provide details in Appendix F.

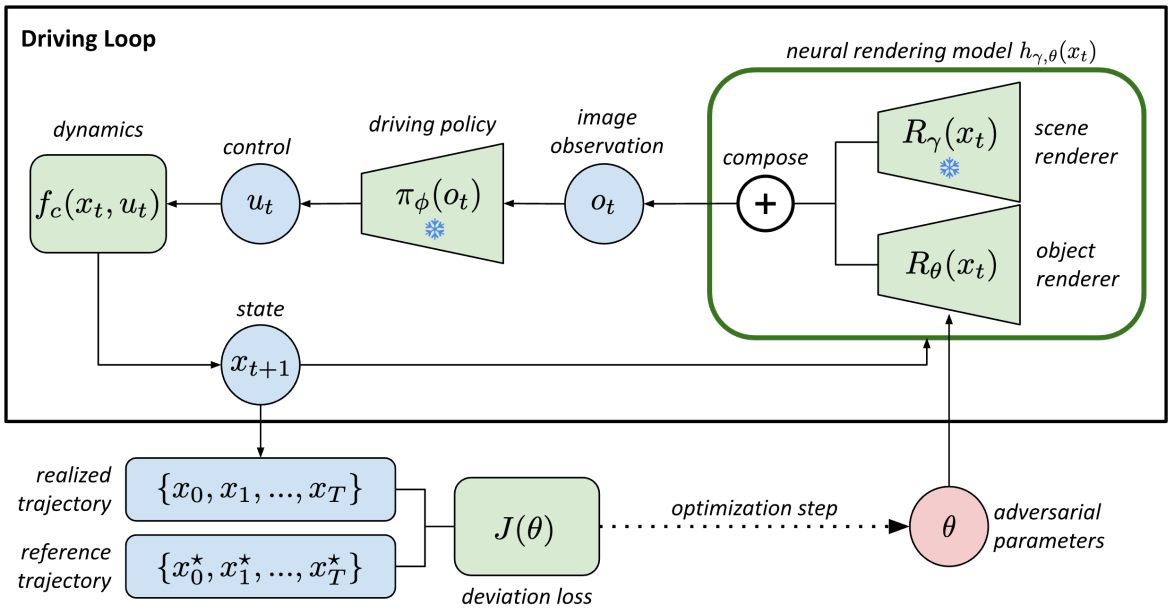


Fig. 2: A computation diagram of our algorithm for generating adversarial attacks. The inner driving loop consists of three components: the neural rendering model, the driving policy, and the car dynamics. We inject the adversarial perturbation to the surrogate scene by composing the outputs of one or more neural object renderers (the single object case is shown above for simplicity) with the output of the neural scene renderer. The parameters of the object renderer(s) are optimized to maximize the deviation of the realized trajectory from the reference trajectory, while keeping the parameters of the driving policy and scene renderer frozen.

images from CARLA and obtaining gradients with respect to the composited alteration [14]. We use NeRFs to learn surrogate models of the scene and sensor model instead. This surrogate model not only gives us an automated method to reconstruct scenes from pose-annotated images, but also provides efficient gradient computation giving us a differentiable form for the sensor h. For the purposes of optimization, we found traditional NeRF representations to be intractable in terms of compute and memory requirements (during gradient computation). Thus, we opt to use the multi-resolution hash grid representation, Instant-NGP [15].

Note that, similar to existing work, we detach the gradients of the image observation with respect to the camera coordinates (which are attached to the ego vehicle) [16]. We include more details regarding this in Appendix D.

B. Adversarial Object Insertion

We use insertion and texturing of multiple objects as our adversarial perturbations to the background scene. To do this, we first reconstruct regular objects, such as cars, as individual NeRFs from pose-annotated images. For our object NeRFs we simply store color values directly on the voxel grids of Instant-NGP, which are tri-linearly interpolated within each voxel. By choosing these color voxel grids as our adversarial parameters θ, we can perform independent adversarial texture attacks over multiple objects. The object NeRFs can be easily composed with our background scene NeRF. This is done via alpha compositing, which leverages opacity and depth values that can be easily computed.

C. Gradient-based Adversarial Attack

Obtaining gradients for the problem in Eqn. (1) should be possible with an autodifferentiation framework such as PyTorch [17]. We find that naively computing the gradient via backpropagation results in memory issues as we scale up trajectory lengths due to all the intermediary compute variables used to compute the integral in Eqn. A.1 being stored until the end of the trajectory. We achieve drastic memory savings by using the adjoint method [18] which only keeps track of the adjoint variables λ along the trajectory. In our case, the adjoint variables are three-dimensional, allowing us to only use as much memory as it takes to compute a single jacobian vector product of the composition of models given by (5), (3), (4).

To summarize, the computation of our gradient-based adversarial attack proceeds as follows: noitemsep,topsep=-1pt

1) We rollout our policy in our surrogate simulator to compute the loss and the trajectory $x_{1:T}$.
2) We perform a backward pass to compute adjoint variables for gradient computation.
3) Using the adjoint variables, we compute the gradient $\nabla_\theta J$ and update parameters θ.

IV. Experiments

To demonstrate the effectiveness of our framework, we aim to reconstruct a driving scenario from posed images, generate adversarial attacks and validate that those attacks transfer to the deployment scene. Through our experiments, we would like to answer the following key questions:

	CARLA Deployment	Surrogate Scene		CARLA Deployment	
Scenario	Unperturbed	Random	Gradient	Random	Gradient
Straight	1166	1132 ± 7	2347 ± 49	1193 ± 19	$1702. \pm 160$
Right	1315	2084 ± 10	4105 ± 847	1476 ± 12	$2101. \pm 75$
Left	1448	1460 ± 8	4125 ± 124	1158 ± 163	$2240. \pm 574$
	Physical Deployment	Surrogate Scene		Physical Deployment	
Setup	Unperturbed	Random	Gradient	Random	Gradient
Green Screen	48	34 ± 4	157 ± 1	46 ± 3	248 ± 72
Monitor				47 ± 3	76 ± 48

TABLE I: Comparison of the total cross-track error for all the scenario tested. Results are shown for the following cases: (1) no attack in the deployment scene (unperturbed), (2) an adversarial attack (random or gradient) in the surrogate NeRF scene, (3) an attack in the deployment scene. We separate results from the CARLA and physical deployments, we show that gradients in our surrogate simulator are useful for finding adversarial attacks and these attacks remain effective when transferred to the deployment environment.

(Q1) Can gradient based optimization find better adversarial examples than random search?
(Q2) Are NeRF models suitable surrogate renderers for gradient based adversarial optimization?
(Q3) Are adversarial attacks transferable from NeRF back to the deployment domain?

A. Evaluation Metrics

We evaluate our method on two distinct deployment environments, CARLA simulator and a real world RC car. In the CARLA simulator our objects were alpha composited on top of the base CARLA rendering. In the real world deployment scene, we test overlaying a texture on the camera input feed as well as placing a monitor containing the texture. More details of these setups are given in D1

We measure the effectiveness of an attack with our adversarial objective, the cross track error of the vehicle. We use the road center as the reference and so even an unperturbed driving policy has some non-zero deviation which we report under "Unperturbed" in Table I. To characterize the insensitivity of our method to random seeds, we run 5 separate attacks per scenario for both the gradient-based and random attacks with different random initializations of the adversarial parameters. We report the mean and standard deviation of our metric. Our proposed method of attack is via gradient-based optimization using the method outlined in Section III-C. The gradient-based attack uses 50 iterations of optimization using Adam, with a learning rate of 0.1. Due to the high dimensional parameterization, detailed in D1, bayesian optimization becomes computationally intractable. Therefore, as a baseline for our method, we perform a random search parameter attack on the NeRF surrogate model that samples parameters from a Gaussian distribution with mean zero and a standard deviation of 5. We chose this standard deviation to match the distribution over parameters we found in our gradient attacks. We use the same number of function evaluations, selecting the best achieved attack among the 50 random samples for the CARLA experiment. For real-world experiments, we didn't find much variation between random attacks in the surrogate simulator, showing the difficulty of random search in high dimensional parameter spaces.

B. Experimental Results

Example gradient attack trajectories are shown in Figure A.1 We include more visualization of results for deployments of adversarial attacks, both in CARLA simulation in the real world, in Appendix H3. In Table I we compare the total cross track errors caused by our adversarial attack against the expert lane following controller. We observe in all 3 CARLA scenarios (averaged over 5 seeds each) that our adversarial attacks using gradient optimization consistently produce significant deviation from the lane center. When transferring these attacks back into the deployment scene, we see that although the magnitude of the deviation is reduced, we still retain a significant increase over the unperturbed or random search setting. The difference is likely due to visual imperfections in our surrogate NeRF simulator compared to the deployment scene. The random search perturbations are far less effective, remaining near the baseline unperturbed trajectory for 2 out of the 3 cases.

For the real world experiment, we observed a similar result. Random attacks consistently fail to elicit deviation from the driving policy both in the surrogate and deployment scenes. Over 5 random seeds, not a single random attack was able to cause the vehicle to exit the track. Gradient attacks on the other hand are reliably able to find strong attacks with little variance in the surrogate scene. When transferring our attacks to the real world, we find the attacks to retain their strength in the green screen setup. The strength of the attack is relatively diminished when using the monitor to project the attack but is nonetheless consistently higher than the random attack and causes the vehicle to understeer and exit the track on occasion. We suspect this is due to the display properties of the monitor which can alter the appearance of the adversarial perturbation.

V. Conclusion

We presented a method for generating 3D-consistent object-based adversarial perturbations in autonomous driving scenarios. Unlike previous approaches that rely on making edits on top of fixed pre-recorded data or black-box simulators,

we develop a differentiable simulator directly with a neural radiance field representation of geometry and texture of a scene that admits gradients through the rendering of camera and depth observations. Through alpha-compositing, we can introduce new objects also represented as neural radiance fields into the scene and optimize color perturbations of the objects. While our particular implementation is only a first step towards demonstrating NERF based adversarial attack generation, we believe that our framework represents a promising new direction for automatic evaluation of autonomous vehicles. We expect our method to benefit greatly from continued improvements being made to neural rendering and their wider adoption for AV/robotic simulation.

REFERENCES

[1] Kalra Nidhi and Susan M. Paddock. Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability? https://www.rand.org/pubs/research_reports/RR1478.html, 2016. [Online; accessed 19-July-2018].

[2] Alexis C Madrigal. Inside waymo's secret world for training self-driving cars. *The Atlantic*, Aug 2017. URL https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/.

[3] James Tu, Mengye Ren, Siva Manivasagam, Ming Liang, Bin Yang, Richard Du, Frank Cheng, and Raquel Urtasun. Physically Realizable Adversarial Examples for LiDAR Object Detection, April 2020. URL https://arxiv.org/abs/2004.00543.

[4] Yulong Cao, Chaowei Xiao, Dawei Yang, Jing Fang, Ruigang Yang, Mingyan Liu, and Bo Li. Adversarial objects against lidar-based autonomous driving systems. *CoRR*, abs/1907.05418, 2019. URL http://arxiv.org/abs/1907.05418.

[5] James Tu, Huichen Li, Xinchen Yan, Mengye Ren, Yun Chen, Ming Liang, Eilyan Bitar, Ersin Yumer, and Raquel Urtasun. Exploring adversarial robustness of multi-sensor perception systems in self driving, January 2022. URL https://arxiv.org/abs/2101.06784.

[6] Seung Wook Kim, Jonah Philion, Antonio Torralba, and Sanja Fidler. Drivegan: Towards a controllable high-quality neural simulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5820–5829, June 2021.

[7] Yun Chen, Frieda Rong, Shivam Duggal, Shenlong Wang, Xinchen Yan, Sivabalan Manivasagam, Shangjie Xue, Ersin Yumer, and Raquel Urtasun. Geosim: Realistic video simulation via geometry-aware composition for self-driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7230–7240, June 2021.

[8] Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1389–1399, June 2023.

[9] Jingkang Wang, Ava Pun, James Tu, Sivabalan Manivasagam, Abbas Sadat, Sergio Casas, Mengye Ren, and Raquel Urtasun. AdvSim: Generating Safety-Critical Scenarios for Self-Driving Vehicles, January 2022. URL https://arxiv.org/abs/2101.06549.

[10] Davis Rempe, Jonah Philion, Leonidas J Guibas, Sanja Fidler, and Or Litany. Generating useful accident-prone driving scenarios via a learned traffic prior. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17305–17315, 2022.

[11] Maximilian Igl, Daewoo Kim, Alex Kuefler, Paul Mougin, Punit Shah, Kyriacos Shiarlis, Dragomir Anguelov, Mark Palatucci, Brandyn White, and Shimon Whiteson. Symphony: Learning realistic and diverse agents for autonomous driving simulation, 2022. URL https://arxiv.org/abs/2205.03195.

[12] Yasasa Abeysirigoonawardena, Florian Shkurti, and Gregory Dudek. Generating adversarial driving scenarios in high-fidelity simulators. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8271–8277. IEEE, 2019.

[13] Jinghan Yang, Adith Boloor, Ayan Chakrabarti, Xuan Zhang, and Yevgeniy Vorobeychik. Finding Physical Adversarial Examples for Autonomous Driving with Fast and Differentiable Image Compositing, June 2021. URL https://arxiv.org/abs/2010.08844.

[14] Naman Patel, Prashanth Krishnamurthy, Siddharth Garg, and Farshad Khorrami. Overriding Autonomous Driving Systems Using Adaptive Adversarial Billboards. *IEEE Transactions on Intelligent Transportation Systems*, 23 (8):11386–11396, August 2022.

[15] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):1–15, July 2022.

[16] Niklas Hanselmann, Katrin Renz, Kashyap Chitta, Apratim Bhattacharyya, and Andreas Geiger. King: Generating safety-critical driving scenarios for robust imitation via kinematics gradients, 2022. URL https://arxiv.org/abs/2204.13683.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[18] Krishna Murthy Jatavallabhula, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss,

Breandan Considine, Jerome Parent-Levesque, Kevin Xie, Kenny Erleben, Liam Paull, Florian Shkurti, Derek Nowrouzezahrai, and Sanja Fidler. gradsim: Differentiable simulation for system identification and visuomotor control. *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=c_E8kFWfhp0.

[19] A. Tewari, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, Z. Xu, T. Simon, M. Nießner, E. Tretschk, L. Liu, B. Mildenhall, P. Srinivasan, R. Pandey, S. Orts-Escolano, S. Fanello, M. Guo, G. Wetzstein, J.-Y. Zhu, C. Theobalt, M. Agrawala, D. B Goldman, and M. Zollhöfer. Advances in neural rendering. In *ACM SIGGRAPH 2021 Courses*, SIGGRAPH '21, 2021.

[20] Adam R. Kosiorek, Heiko Strathmann, Daniel Zoran, Pol Moreno, Rosalia Schneider, Soňa Mokrá, and Danilo J. Rezende. Nerf-vae: A geometry aware 3d scene generative model, 2021. URL https://arxiv.org/abs/2104.00587.

[21] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[22] Bangbang Yang, Yinda Zhang, Yinghao Xu, Yijin Li, Han Zhou, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. Learning object-compositional neural radiance field for editable scene rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13779–13788, 2021.

[23] Sagie Benaim, Frederik Warburg, Peter Ebert Christensen, and Serge Belongie. Volumetric disentanglement for 3d scene manipulation, 2022. URL https://arxiv.org/abs/2206.02776.

[24] Ashkan Mirzaei, Tristan Aumentado-Armstrong, Konstantinos G. Derpanis, Jonathan Kelly, Marcus A. Brubaker, Igor Gilitschenski, and Alex Levinshtein. Spinnerf: Multiview segmentation and perceptual inpainting with neural radiance fields, 2022. URL https://arxiv.org/abs/2211.12254.

[25] Pratul P Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T Barron. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7495–7504, 2021.

[26] Weicai Ye, Shuo Chen, Chong Bao, Hujun Bao, Marc Pollefeys, Zhaopeng Cui, and Guofeng Zhang. Intrinsicnerf: Learning intrinsic neural radiance fields for editable novel view synthesis, 2022. URL https://arxiv.org/abs/2210.00647.

[27] Yinghao Xu, Menglei Chai, Zifan Shi, Sida Peng, Skorokhodov Ivan, Siarohin Aliaksandr, Ceyuan Yang, Yujun Shen, Hsin-Ying Lee, Bolei Zhou, and Tulyakov Sergy. Discoscene: Spatially disentangled generative radiance field for controllable 3d-aware scene synthesis, 2022. URL https://arxiv.org/abs/2212.11984.

[28] Abhijit Kundu, Kyle Genova, Xiaoqi Yin, Alireza Fathi, Caroline Pantofaru, Leonidas J Guibas, Andrea Tagliasacchi, Frank Dellaert, and Thomas Funkhouser. Panoptic neural fields: A semantic object-aware neural scene representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12871–12881, 2022.

[29] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022.

[30] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33: 15651–15663, 2020.

[31] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. *European Conference on Computer Vision (ECCV)*, 2022.

[32] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects, 2022. URL https://arxiv.org/abs/2212.08051.

[33] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL http://www.blender.org.

[34] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars, 2016. URL https://arxiv.org/abs/1604.07316.

[35] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[36] Felipe Codevilla, Matthias Müller, Alexey Dosovitskiy, Antonio M. López, and Vladlen Koltun. End-to-end driving via conditional imitation learning. *CoRR*, abs/1710.02410, 2017. URL http://arxiv.org/abs/1710.02410.

[37] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[38] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

APPENDIX

A. NeRF

Neural 3D representations, such as neural radiance fields (NeRF), have seen significant activity in recent years due to their ability to reconstruct real world objects and scenes to very high detail using only posed images. A survey of recent progress in neural rendering can be found in [19].

Differentiable rendering. NeRFs represent scenes as emissive volumetric media [15]. Unlike surface rendering, volumetric rendering does not suffer from explicit hard discontinuities, which are difficult to handle for traditional surface rendering methods[19]. We exploit the differentiable volume rendering of NeRFs, to robustly compute efficient gradients for arbitrary geometries.

Volume Rendering A neural radiance field consists of two fields, $\sigma_\phi(x), L_\psi(x,\omega)$ that encode the density σ at every location x and the outgoing radiance L at that location in the direction ω. In NeRFs, both of these functions are represented by parameterized differentiable functions, such as neural networks. Given a radiance field, we are able to march rays through an image plane and reconstruct a camera image from a given camera pose and intrinsic matrix using the rendering function:

$$I(x,\omega) = \int_0^T \sigma(t) \exp\left(\int_0^t \sigma(\hat{t})\mathrm{d}\hat{t}\right) L(t,-\omega)\mathrm{d}t \tag{A.1}$$

Where $L(t,\cdot)$ and $\sigma(t)$ are shorthands for $L(t\omega + x, \cdot)$ and $\sigma(t\omega + x)$, and $I(x,\omega)$ is the intensity at a location x given in world space in the direction ω.

Composition and editing. Recent works have extended the static single scene setting of NeRF to composition of NeRFs, scene disentanglement, as well as editing and relighting. Specifically, [20] encodes scenes with latent codes from which new scenes can be generated. [21], [22] and [23] introduce compositional radiance fields to represent different objects and realize scene decomposition. [24] utilizes 2D segmentation information to perform 3D scene inpainting. [25] and [26] decompose color into different illumination components. [27] [28] learn priors from big datasets of images to disentangle existing scenes.

For our adversarial attacks to contain 3D semantics, it is crucial to insert the perturbation in a 3D aware manner. For this we utilize another feature of neural radiance fields, which is to output opacity values. Specifically, in Eqn. (A.1) we can extract the transmittance component, which acts as a measure of the pixel transparency α:

$$\alpha(x,\omega) = \exp\left(\int_0^t \sigma(\hat{t})\mathrm{d}\hat{t}\right) \tag{A.2}$$

Furthermore, we can replace the radiance term with distance in (A.1) to extract the expected termination depth of a ray z:

$$z(x,\omega) = \int_0^T t\sigma(t)\alpha(t)\mathrm{d}t \tag{A.3}$$

We consider the case of two radiance fields, the object radiance field σ_o, L_o and the background radiance field σ_s, L_s. We use a transformation matrix to correspond ray coordinates between the scene and the object radiance field.

By applying equations (A.1), (A.2), (A.3) to a single ray that corresponds to both the base scene and the object radiance field, we obtain the values c_o, α_o, z_o, c_s, α_s, z_s respectively, where α_* is the opacity and z_* is the depth along the ray. We denote the foreground and background values at a pixel as

$$f = \arg\min_{o,s}(z_s, z_o) \tag{A.4}$$

$$b = \arg\max_{o,s}(z_s, z_o) \tag{A.5}$$

The final blended color is then given by:

$$c = \frac{\alpha_f c_f + (1-\alpha_f)\alpha_b c_b}{\alpha_f + \alpha_b(1-\alpha_f)} \tag{A.6}$$

In the case of multiple object NeRFs, we simply repeat the alpha blending for each object to composite them all into the same scene.

Accelerated rendering. The original NeRF method has high computational cost of training and rendering. Structured grid NeRFs reduce computation cost by storing direct density and color variables [29] or latent features[30, 31] on explicit 3D grids. Instant Neural Graphics Primitives (iNGP) uses multi-scale dense grids and sparse hash grids of features that are decoded to color and density by a MLP [15]. We use iNGP like models to represent the scene and objects in our work.

B. Vehicle Dynamics

The dynamics in equation (5) can take multiple forms, for the CARLA experiments, we choose the simplest kinematic model of a car, a Dubin's vehicle:

$$\dot{x} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ u \end{bmatrix} \tag{A.7}$$

For the purposes of the CARLA deployment environment, we find that it is sufficient to consider the kinematic model with fixed velocity, and only angular control. Thus, our imitation learning policy in Eqn. (3) only outputs steering commands. We note that our approach is applicable to any dynamics model, as long as it is differentiable.

For the real world experiments, we opted for a fixed velocity Ackerman steering model:

$$\dot{x} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v}{l}\tan(\theta) \end{bmatrix} \tag{A.8}$$

where l is the robot wheelbase.

C. Implicit Differentiation

To carry out the adjoint method for obtaining gradients of the trajectory optimization problem stated in Equation (1), we need to perform two passes over the trajectory.

Explicitly, the method performs a forward simulation to compute the variables x_t and then subsequently a backward pass to compute adjoint variables λ_t by solving the equations:

$$\frac{\partial G(x_{t-1}, x_t)}{\partial x_t}^\top \lambda_t = -\frac{\partial C(x_t)}{\partial x_t}^\top - \frac{\partial G(x_t, x_{t+1})}{\partial x_t}^\top \lambda_{t+1} \tag{A.9}$$

with the boundary condition:

$$\frac{\partial G(x_{T-1}, x_T)}{\partial x_T}^\top \lambda_T = -\frac{\partial C(x_T)}{\partial x_T}^\top \tag{A.10}$$

Finally, the gradient of the loss can be calculated as:

$$\nabla_\theta J = \lambda_1^\top \frac{\partial G(x_0, x_1, \theta)}{\partial x_0} \frac{\partial x_0}{\partial \theta} + \sum_{t=1}^{T} \lambda_t^\top \frac{\partial G(x_{t-1}, x_t, \theta)}{\partial \theta} \tag{A.11}$$

Throughout both passes we do not need to store large intermediate variables and only need to accumulate the gradient at each step.

D. Optimization Details

As described in Section III-A, following prior work, we do not propagate gradients of camera parameters through the sensor model function. Specifically, we set,

$$o_t = h_{\gamma,\theta}(\text{stop\_gradient}(x_t)) \tag{A.12}$$

Thus gradients of the observation will only be taken with respect to the adversarial object parameters θ and not the state of the car. The gradient with respect to x_t corresponds to exploiting higher order effects of how the observation would change if the car was looking in a slightly different direction due to previous steps of the attacks, and leads to a very non-smooth loss objective that is not useful for finding practical attacks.

For experiments in the real world, we found the attacks were sometimes very sensitive to the robot's pose. To alleviate this issue, we chose to optimize multiple randomly sampled initial poses simultaneously. The samples were normally distributed around the nominal car starting location, with a standard deviation of 0.1.

1) Optimization parameters: In all our experiments, our optimization parameters θ correspond to values on the NGP voxel grid. Since we have removed the decoder, the grid values directly correspond to the color for a given position in the volume. Due to this, the parametrization even for small models can get quite large, in the order of a 5 million for the hydrant.

E. NeRF Models

When training the surrogate NeRF models of the background scene and objects, we use the default Instant-NGP hyperparameters and optimize over 50 epochs using the Adam optimizer.

The source 3D assets for our objects were obtained from the Objaverse dataset [32] and posed images produced by rendering with Blender[33]. For our object models, we choose to use Instant-NGP without a decoder, instead directly encoding the colour values in the feature grid. Furthermore, we remove view dependence for better multi-view consistency. Finally, we use lower resolutions for the object feature grids as compared to the scene feature grids. The object feature grids contain resolutions up to 128^3 and 64^3 features for the car and hydrant, respectively. Since our adversarial objective does not have any smoothness constraint, we found it critical to use lower resolution grids and remove the positionally encoded feature decoders to avoid aliasing effects.

Fig. A.1: Base car on the left; random texture in the middle; adversarial texture on the right.

F. Driving Policy.

We train our own policy on which the attack will be performed. Our policy is an end-to-end RGB image neural network policy and the architecture is taken from [34]. We make a slight addition to goal condition the policy by adding a goal input to the linear component and increasing the capacity of the linear layers. The policy is trained via imitation learning, specifically DAgger [35], [36].

Expert actions are given by a lane following controller tuned for the simulator that gets access to the ground truth state, unlike the policy. The expert queried from various states random distances from the center of the road to recover from. Furthermore, random noise augmentation is used on the images during training to make the policy more robust to noisy observations.

G. CARLA

We fit the background scene model using a dataset of 1800 images and their corresponding camera poses, which provide a dense covering of the CARLA scene.

When transferring our attacks back to the deployment scene, opacity values are usually not available. In order to evaluate our attacks, we assume that objects are opaque ($\alpha = 1$), and thus our method of blending in Equation A.3 can be calculated using just the depth and color values. We observe from experiments on the CARLA simulator that this type of composition is sufficient for the evaluation in the deployment environment.

Driving Policy For our driving policy the initial training dataset of images is collected from the intersection in CARLA. We further fine-tuned the policy with some additional data collected from our surrogate simulator to ensure that our policy is not trivially failing due to slight visual differences. We use a total dataset of 120000 images in CARLA and 60000 images in the surrogate simulator in order to train the policy. We validated our policy on a hold out validation set consisting of 12000 images captured purely from the

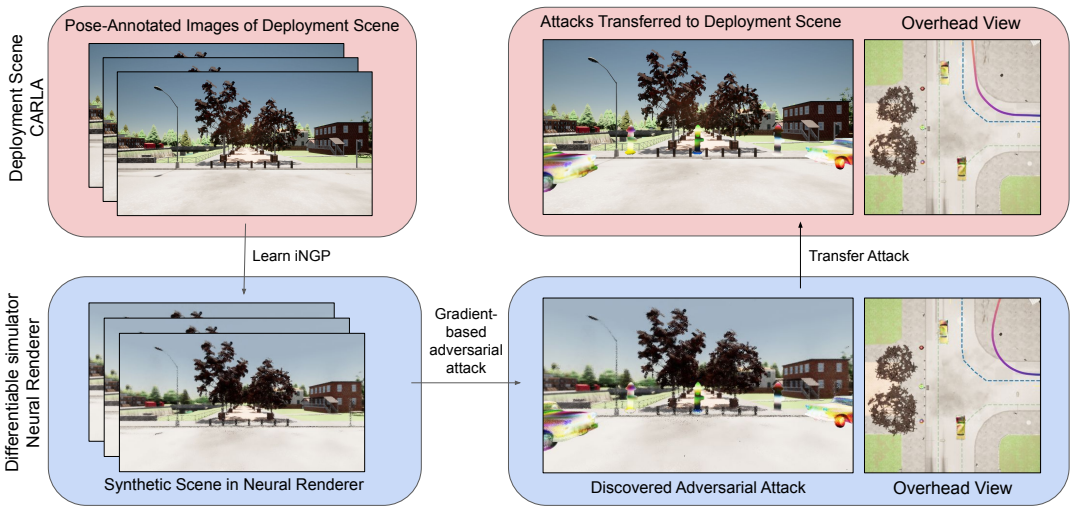


Fig. A.2: Our method can be summarized in the four steps shown. (a) In the top left, we obtain posed images from the deployment scene which can be a simulator or the real world. (b) In the bottom left, we reconstruct a surrogate scene by fitting a NeRF to the posed images as a differentiable simulator and observe only minor perceptual gap. (c) Having the surrogate scene, we can insert objects, which are also represented as NeRFs, and attack their color fields to generate textural attacks. (d) The discovered adversarial objects are introduced back into the deployment scene.

surrogate simulator. All data were collected by running the expert on the 3 reference trajectories. The policy was trained using behaviour cloning, where we gave examples of recovery from deviation by collecting data from random start locations around the nominal trajectory.

H. Real World

We fit the background to a room in the real world using a dataset of 2161 images captured from an iPhone camera at 4K resolution. We collect data covering the room by walking around, then attach the iPhone to the robot to collect further data from the driving view points. The captured videos are processed using COLMAP [37, 38] for both camera intrinsic and the poses.

Driving Policy. We train a driving policy to track a square track in the room marked by green tape, this policy was trained using an expert PID controller with global positioning supplied by the VICON system providing 9584 images. We further augment this again with 12000 images from driving data in the NeRF scene. An overview of our working area is given in Figure A.3.

For all real world attacks we optimize the color of a cube in the surrogate NeRF scene, placed at one of the corners such that the camera will encounter this cube as the car takes the turn.

1) Robot: We carry out experiments using the RACECAR/J[2] platform. The robot is equipped with a ZED stereo camera, of which we only utilize the RGB data from the left sensor, which has been configured to a resolution of 366x188 at 10 frames per second. We operate the robot inside a VICON system that positions the robot at a rate of 50Hz streaming through a remotely connected computer that runs policy as well as the image processing for some of the attacks.

[2]https://racecarj.com/

Fig. A.3: Picture of driving area for the real world scenario experiments.

2) Green Screen Attack: For the green screen attack, we utilized a VICON system to accurately position both our robot and the green screen target. Using the green screen target position, as well as the camera parameters, we project one face of the cube on the input image to the policy. We opt to overlay the cube in such a manner to keep the policy driving in real time and to ensure that there is no penalty on control frequency. The image compositions is done at the remote computer where the controls are computed, which are then sent wirelessly to the robot to execute.

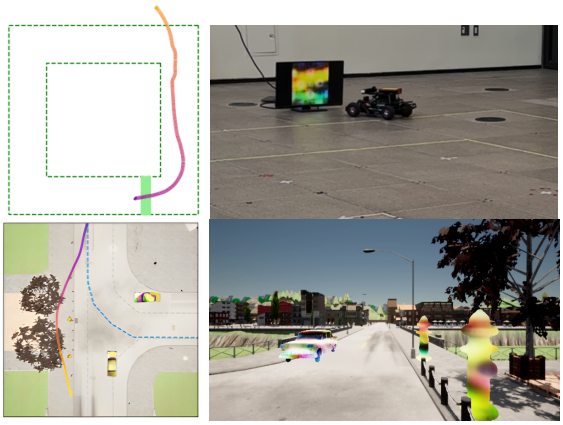


Fig. A.1: Selected overhead views and snapshots from adversarial deployment trajectories in the real world (top row: monitor displays adversarial texture discovered in NeRF), and in CARLA (bottom row: adversarial objects inserted in the simulator).

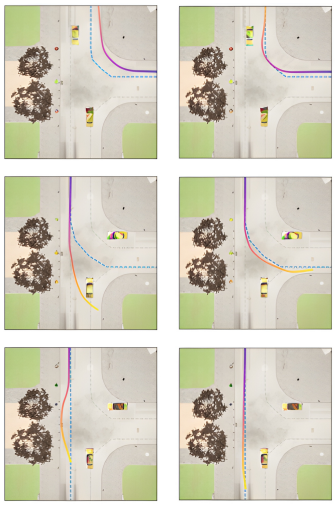


Fig. A.2: The performance of the driving policy before (left) and after (right) retraining on the discovered adversarial scenarios.

3) Monitor Attack: To replace the green screen with a physical object, we place a monitor and display the same attack as above on the monitor. We place the monitor in a location such that it is visually consistent with the NeRF and green screen attacks. For the monitor attack, we utilize a 27-inch monitor with a 16:9 aspect ratio. Since the adversarial objects optimized in earlier examples are cubes we only use the center of the monitor to display the attack.

I. Incorporating Discovered Adversarial Scenarios in the Training Set

Our primary focus in this paper was to discover adversarial attacks for the evaluation of pretrained self-driving policy.

Fig. A.3: Sample renderings of the left turn trajectory with the adversarial perturbations in CARLA from the ego vehicle's point of view. Four different snapshots from the evolution of the trajectory are shown.

Here we perform some preliminary investigations on fine-tuning our self-driving policies, on the old data and the adversarial attacks we found. Specifically, we take the attacks discovered by the gradient-based optimization and use them to collect additional imitation learning data. The collection is performed in the CARLA simulator using the depth compositing approach to insert the adversarial objects, as was done for the evaluation in the main paper. Apart from the object compositing, the data is collected in the same way as the original CARLA data used to train the base policy. We collect 24000 total frames over three trajectories with two different starting points. After fine-tuning our policy on the combination of the original dataset and the new adversarially augmented dataset, we evaluate the fine-tuned agent in the same scenario. We visualize the trajectories of the fine-tuned policy in Figure A.2 and report on the total deviation compared to before fine-tuning in Table II. We find that the policy is no longer susceptible to the adversarial attacks, even though the initial starting position for evaluation was unseen during training.

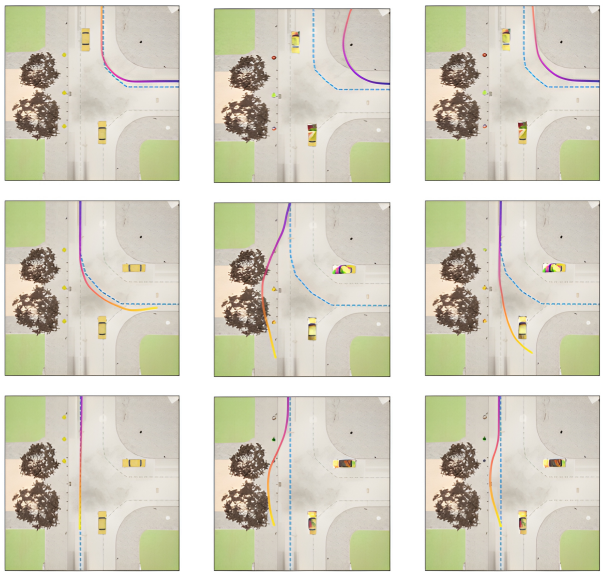
J. CARLA Visualizations

We show first person visualizations of our discoverered adversarial attacks inserted back into the CARLA deployment simulator in Figure A.3. We note the smoothness of the texture discovered by our method. Purely perceptual single-frame attacks typically exhibit a much higher frequency texture.

We show additional overhead trajectory views of adversarially attacked trajectories from one CARLA scene in Figure A.4.

	CARLA	Attack Transfer in CARLA		CARLA Attack After Retraining
Scenario	Unperturbed	Random	Gradient	Gradient
Straight	1166	1193 ± 19	$1702. \pm 160$	1250
Right	1315	1476 ± 12	$2101. \pm 75$	1307
Left	1448	1158 ± 163	$2240. \pm 574$	1419

TABLE II: Comparison of the total cross-track error for the retraining experiment over the 3 different trajectories. Results are extending the results from the main paper TableI shown for the following cases: (1) no attack in CARLA (unperturbed), (2) an attack in the CARLA scene, (3) an attack in the CARLA scene after the driving policy is retrained using adversarial data.



(a) Unperturbed (b) Attacks in NERF (c) Transferred

Fig. A.4: Overhead views of three distinct trajectories driven by the policy. (a) shows the policy driving behavior in CARLA when no adversarial perturbation is introduced. (b) shows the policy driving behavior in the surrogate simulator with the discovered adversarial perturbation. (c) shows the same perturbation transferred to the deployment scene.