A Method Details

A.1 NeRF

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.

Compositing 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.

A.2 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.

A.3 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.

A.4 Optimization Details

As described in Section 4.1, 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.

A.4.1 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.

B Experimental Details

B.1 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 [45] and posed images produced by rendering with Blender[46]. 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.

B.2 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 [47]. 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 [48], [49].

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.

B.3 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 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.

B.4 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 [50, 51] 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

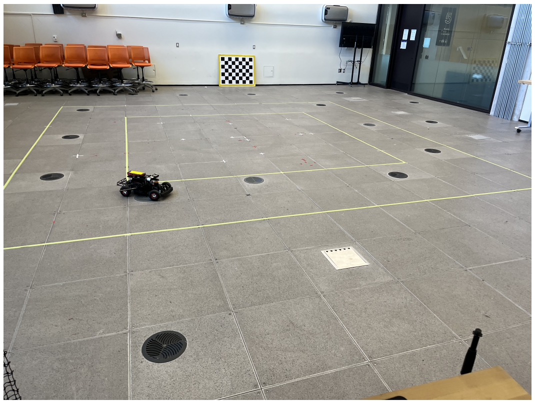


Figure B.1: Picture of driving area for the real world scenario experiments.

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 B.1.

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.

B.4.1 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.

B.4.2 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.

B.4.3 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.

C Additional Experimental Results

C.1 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. 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

[2] https://racecarj.com/

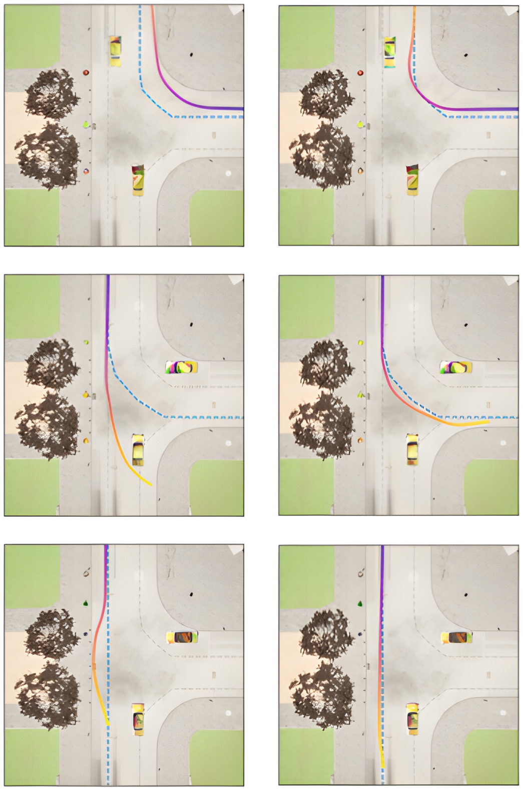


Figure C.1: The performance of the driving policy before (left) and after (right) retraining on the discovered adversarial scenarios.

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

Table 2: 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 Table1 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.

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 C.1 and report on the total deviation compared to before fine-tuning in Table 2. 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.

C.2 CARLA Visualizations

We show first person visualizations of our discoverered adversarial attacks inserted back into the CARLA deployment simulator in Figure C.2. We note the smoothness of the texture discovered by our method. Purely perceptual single-frame attacks typically exhibit a much higher frequency texture.

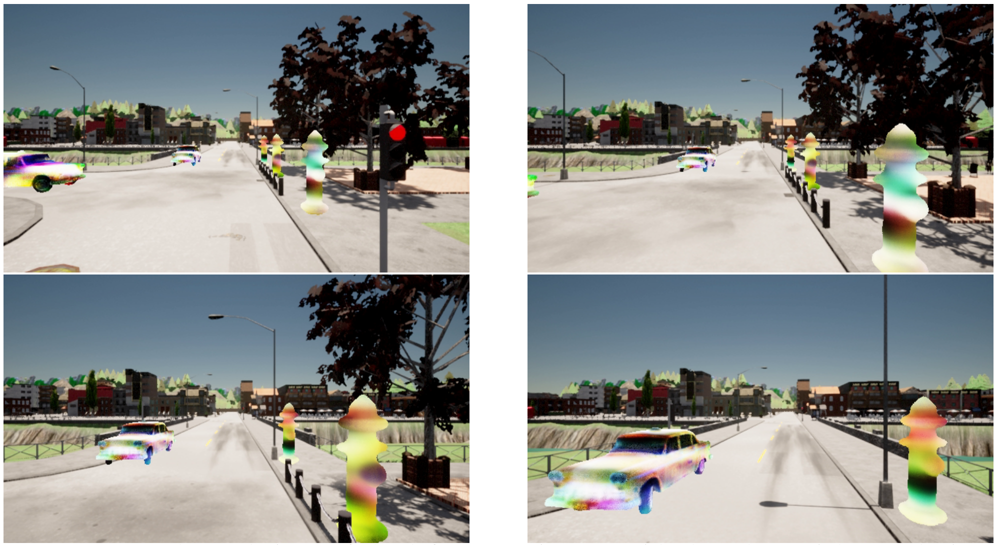


Figure C.2: Sample renderings of the left turn trajectory with the adversarial perturbations in CARLA from the ego vehicle's point of view. Four different snapshots from the evolution of the trajectory are shown.

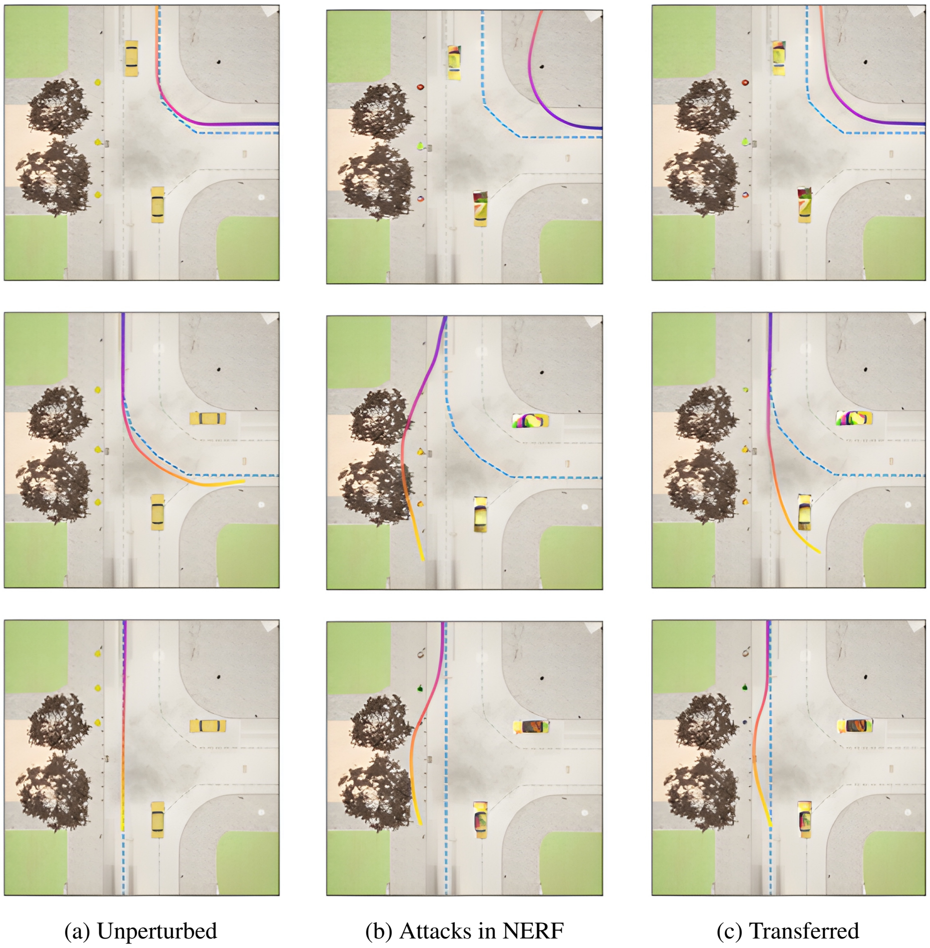
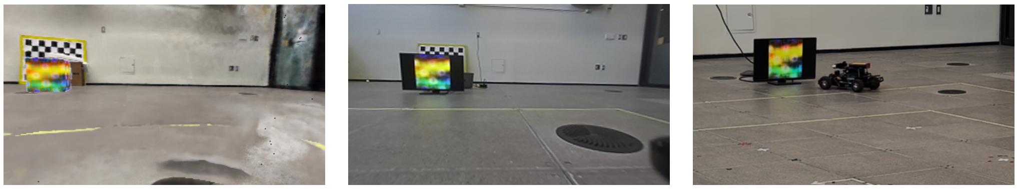


Figure C.3: Overhead views of three distinct trajectories driven by the policy. (a) shows the policy driving behavior in CARLA when no adversarial perturbation is introduced. (b) shows the policy driving behavior in the surrogate simulator with the discovered adversarial perturbation. (c) shows the same perturbation transferred to the deployment scene.



(a) Surrogate Simulator (b) First person view (c) Third person view

Figure C.4: Real-world adversarial monitor attack visualizations.

We show additional overhead trajectory views of adversarially attacked trajectories from one CARLA scene in Figure C.3.

C.3 Real-world Visualizations

We show aligned visualizations of the same adversarial real-world monitor attack in Figure C.4.