SADA: Semantic Adversarial Diagnostic Attacks for Autonomous Applications

Article *in* Proceedings of the AAAI Conference on Artificial Intelligence · April 2020

DOI: 10.1609/aaai.v34i07.6722

3 authors:

Abdullah Hamdi
King Abdullah University of Science and Technology
10 PUBLICATIONS **22** CITATIONS
SEE PROFILE

Matthias Mueller
Intel
30 PUBLICATIONS **3,156** CITATIONS
SEE PROFILE

Bernard Ghanem
King Abdullah University of Science and Technology
235 PUBLICATIONS **10,217** CITATIONS
SEE PROFILE

SADA: Semantic Adversarial Diagnostic Attacks for Autonomous Applications

Abdullah Hamdi, Matthias Müller, Bernard Ghanem
King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
{abdullah.hamdi, matthias.mueller.2, Bernard.Ghanem}@kaust.edu.sa

Abstract

One major factor impeding more widespread adoption of deep neural networks (DNNs) is their lack of robustness, which is essential for safety-critical applications such as autonomous driving. This has motivated much recent work on adversarial attacks for DNNs, which mostly focus on pixel-level perturbations void of semantic meaning. In contrast, we present a general framework for adversarial attacks on trained agents, which covers semantic perturbations to the environment of the agent performing the task as well as pixel-level attacks. To do this, we re-frame the adversarial attack problem as learning a distribution of parameters that always fools the agent. In the semantic case, our proposed adversary (denoted as BBGAN) is trained to sample parameters that describe the environment with which the black-box agent interacts, such that the agent performs its dedicated task poorly in this environment. We apply BBGAN on three different tasks, primarily targeting aspects of autonomous navigation: object detection, self-driving, and autonomous UAV racing. On these tasks, BBGAN can generate failure cases that consistently fool a trained agent.

Introduction

As a result of recent advances in machine learning and computer vision, deep neural networks (DNNs) are now interleaved with many aspects of our daily lives. DNNs suggest news articles to read and movies to watch, automatically edit our photos and videos, and translate between hundreds of languages. They are also bound to disrupt transportation with autonomous driving slowly becoming a reality. While there are already impressive demos and some successful deployments, safety concerns for boundary conditions persist. While current models work very well on average, they struggle with robustness in certain cases. Recent work in the adversarial attack literature shows how sensitive DNNs are to input noise. These attacks usually utilize the information about the network structure to perform gradient updates in order to derive targeted perturbations (coined white-box attacks). These perturbations are injected into the input image at the pixel-level, so as to either confuse the network or enforce a specific behavior (Szegedy et al. 2013; Goodfellow, Shlens, and Szegedy 2015; Carlini and Wagner 2017; Kurakin, Goodfellow, and Bengio 2016).

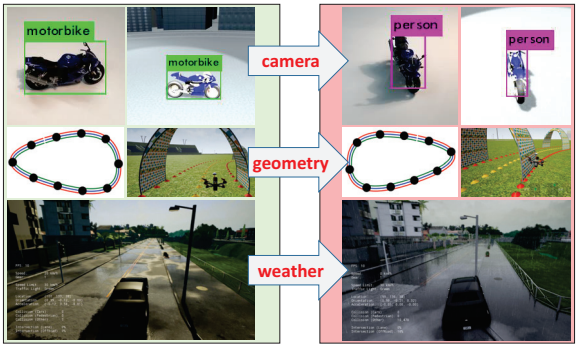

Figure 1: **Semantic Adversarial Diagnostic Attacks**. Neural networks can perform very well on average for a host of tasks; however, they do perform poorly or downright fail when encountering some environments. To diagnose why they fail and how they can be improved, we seek to learn the underlying distribution of semantic parameters, which generate environments that pose difficulty to these networks when applied to three safety critical tasks: object detection, self-driving cars, and autonomous UAV racing.

In practice, such pixel attacks are less likely to naturally occur than semantic attacks which include changes in camera viewpoint, lighting conditions, street layouts, etc. The literature on semantic attacks is much sparser, since they are much more subtle and difficult to analyze (Alcorn et al. 2019; Zeng et al. 2019). Yet, this type of attack is critical to understand/diagnose failure cases that might occur in the real-world. While it is very difficult to investigate semantic attacks on real data, we can leverage simulation as a proxy that can unearth useful insights transferable to the real-world. Figure 1 shows an example of an object misclassified by the YOLOV3 detector (Redmon and Farhadi 2018) applied to a rendered image from a virtual environment, an autonomous UAV racing (Müller et al. 2017) failure case in a recently developed general purpose simulator (Sim4CV (Müller et al. 2018b)), and an autonomous driving failure case in a popular driving simulator (CARLA (Dosovitskiy

et al. 2017)). These failures arise from adversarial attacks on the semantic parameters of the environment.

In this work, we consider environments that are adequately photo-realistic and parameterized by a compact set of variables that have direct semantic meaning (*e.g.* camera viewpoint, lighting/weather conditions, road layout, etc.). Since the generation process of these environments from their parameters is quite complicated and in general non-differentiable, we treat it as a *black-box* function that can be queried but not back-propagated through. We seek to learn an adversary that can produce fooling parameters to construct an environment where the agent (which is also a black-box) fails in its task. Unlike most adversarial attacks that generate sparse instances of failure, our proposed adversary provides a more comprehensive view on how an agent can fail; we learn the distribution of fooling parameters for a particular agent and task and then sample from it. Since Generative Adversarial Networks (GANs (Goodfellow et al. 2014; Arjovsky, Chintala, and Bottou 2017)) have emerged as a promising family of unsupervised learning techniques that can model high-dimensional distributions, we model our adversary as a GAN, denoted as black-box GAN (BBGAN).

Contributions. (1) We formalize adversarial attacks in a more general setup to include both semantic and conventional pixel attacks. (2) We propose BBGAN in order to learn the underlying distribution of semantic adversarial attacks and show promising results on three different safety-critical applications used in autonomous navigation.

Related Work

Pixel-level Adversarial Attacks

Szegedy *et al.* formulate attacking neural networks as an optimization problem (Szegedy et al. 2013). Their method produces a minimal perturbation of the image pixels that fools a trained classifier (incorrect predictions). Several works followed the same approach but with different formulations, such as Fast Gradient Sign Method (FGSM) (Goodfellow, Shlens, and Szegedy 2015) and Projected Gradient Descent (Kurakin, Goodfellow, and Bengio 2016). A comprehensive study of different ways to fool networks with minimal pixel perturbation can be found in the paper (Carlini and Wagner 2017). Most efforts use the gradients of the function to optimize the input, which might be difficult to obtain in some cases (Zeng et al. 2019). However, all of these methods are limited to pixel perturbations to fool trained agents, while we consider more general cases of attacks, *e.g.* changes in camera viewpoint to fool a detector or change in weather conditions to fool a self-driving car. Furthermore, we are interested in the distribution of the semantic parameters that fool the agent, more so than individual fooling examples.

Semantic Attacks beyond Pixels

Beyond pixel perturbations, several recent works perform attacks on the object/camera pose to fool a classifier (Alcorn et al. 2019; Zeng et al. 2019; Hamdi and Ghanem 2019). Other works proposed attacks on 3D point clouds using Point-Net (Xiang, Qi, and Li 2019), and on 3D meshes using differentiable functions that describe the scene (Xiao et al. 2019). Inspired by these excellent works, we extend semantic attacks by using readily available virtual environments with plausible 3D setups to systematically test trained agents. In fact, our formulation includes attacks not only on static agents like object detectors, but also agents that interact with dynamic environments, such as self-driving agents. To the best of our knowledge, this is the first work to introduce adversarial attacks in CARLA(Dosovitskiy et al. 2017), a standard autonomous navigation benchmark.

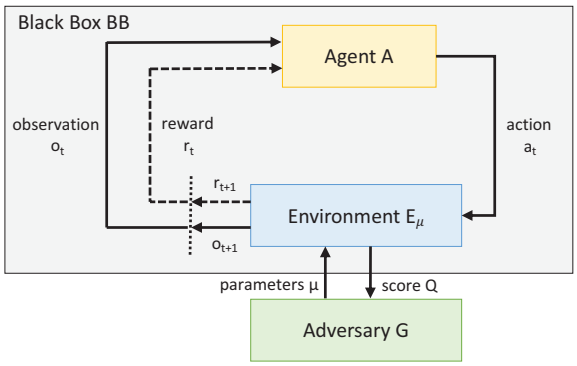

Figure 2: Generic Adversarial Attacks on Agents. $\mathbf{E}_{\boldsymbol{\mu}}$ is a parametric environment with which an agent $\mathbf{A}$ interacts. The agent receives an observation $\mathbf{o_t}$ from the environment and produces an action $\mathbf{a_t}$. The environment scores the agent and updates its state until the episode finishes. A final score $Q(\mathbf{A}, \mathbf{E}_{\boldsymbol{\mu}})$ is given to the adversary $\mathbf{G}$, which in turn updates itself to propose more adversarial parameters $\boldsymbol{\mu}$ for the next episode.

Adversarial Attacks and Reinforcement Learning

Our generic formulation of adversarial attacks is naturally inspired by RL, in which agents can choose from multiple actions and receive partial rewards as they proceed in their task (Sutton and Barto 1998). In RL, the agent is subject to training in order to achieve a goal in the environment; the environment can be dynamic to train a more robust agent (Morimoto and Doya 2001; Pinto et al. 2017; Held et al. 2017). However, in adversarial attacks, the agent is usually fixed and the adversary is the subject of the optimization in order to fool the agent. We formulate adversarial attacks in a general setup where the environment rewards an agent for some task. An adversary outside the environment is tasked to fool the agent by modifying the environment and receiving a score after each episode.

Methodology

Generalizing Adversarial Attacks

Extending attacks to general agents. In this work, we generalize the adversarial attack setup beyond pixel perturbations. Our more general setup (refer to Figure 2) includes semantic attacks, *e.g.* perturbing the camera pose or lighting conditions of the environment that generates observations (*e.g.* pixels in 2D images). An environment $\mathbf{E}_{\boldsymbol{\mu}}$ is parametrized by $\boldsymbol{\mu} \in [\boldsymbol{\mu}_{\min}, \boldsymbol{\mu}_{\max}]^d$. It has an internal state $\mathbf{s}_t$ and produces observations $\mathbf{o}_t \in \mathbb{R}^n$ at each time step

$t \in \{1, \ldots, T\}$. The environment interacts with a trained agent $\mathbf{A}$, which gets $\mathbf{o}_t$ from $\mathbf{E}_{\boldsymbol{\mu}}$ and produces actions $\mathbf{a}_t$. At each time step t and after the agent performs $\mathbf{a}_t$, the internal state of the environment is updated: $\mathbf{s}_{t+1} = \mathbf{E}_{\boldsymbol{\mu}}(\mathbf{s}_t, \mathbf{a}_t)$. The environment rewards the agent with $r_t = R(\mathbf{s}_t, \mathbf{a}_t)$, for some reward function R. We define the episode score $Q(\mathbf{A}, \mathbf{E}_{\boldsymbol{\mu}}) = \sum_{t=1}^{T} r_t$ of all intermediate rewards. The goal of $\mathbf{A}$ is to complete a task by maximizing Q. The adversary $\mathbf{G}$ attacks the agent $\mathbf{A}$ by modifying the environment $\mathbf{E}_{\boldsymbol{\mu}}$ through its parameters $\boldsymbol{\mu}$ without access to $\mathbf{A}$ and $\mathbf{E}_{\boldsymbol{\mu}}$.

Distribution of Adversarial Attacks. We define $\mathbf{P}_{\boldsymbol{\mu'}}$ to be the *fooling distribution* of semantic parameters $\boldsymbol{\mu'}$ representing the environments $\mathbf{E}_{\boldsymbol{\mu'}}$, which fool the agent $\mathbf{A}$.

$$\boldsymbol{\mu'} \sim \mathbf{P}_{\boldsymbol{\mu'}} \Leftrightarrow Q(\mathbf{A}, \mathbf{E}_{\boldsymbol{\mu'}}) \leq \epsilon; \;\; \boldsymbol{\mu'} \in [\boldsymbol{\mu}_{\min}, \boldsymbol{\mu}_{\max}]^d \quad (1)$$

Here, ϵ is a task-specific threshold to determine success and failure of the agent $\mathbf{A}$. The distribution $\mathbf{P}_{\boldsymbol{\mu'}}$ covers all samples that result in failure of $\mathbf{A}$. Its PDF is unstructured and depends on the complexity of the agent. We seek an adversary $\mathbf{G}$ that learns $\mathbf{P}_{\boldsymbol{\mu'}}$, so it can be used to comprehensively analyze the weaknesses of $\mathbf{A}$. Unlike the common practice of finding adversarial examples (*e.g.* individual images), we address the attacks distribution-wise in a compact semantic parameter space. We denote our analysis technique as Semantic Adversarial Diagnostic Attack (SADA). *Semantic* because of the nature of the environment parameters and *diagnostic* because a fooling distribution is sought. We show later how this distribution can be used to reveal agents' failure modes. We propose to optimize the following objective for the adversary $\mathbf{G}$ to achieve this challenging goal:

$$\begin{aligned} &\arg\min_{\mathbf{G}} \;\; \mathbb{E}_{\boldsymbol{\mu} \sim \mathbf{G}}[Q(\mathbf{A}, \mathbf{E}_{\boldsymbol{\mu}})] \\ &\text{s.t. } \{\boldsymbol{\mu} : \boldsymbol{\mu} \sim \mathbf{G}\} = \{\boldsymbol{\mu'} : \boldsymbol{\mu'} \sim \mathbf{P}_{\boldsymbol{\mu'}}\} \end{aligned} \quad (2)$$

Algorithm 1 describes a general setup for $\mathbf{G}$ to learn to generate fooling parameters. It also includes a mechanism for evaluating $\mathbf{G}$ in the black-box environment $\mathbf{E}_{\boldsymbol{\mu}}$ for L iterations after training it to attack the agent $\mathbf{A}$. An attack is considered a fooling attack, if parameter $\boldsymbol{\mu}$ sampled from $\mathbf{G}$ achieves an episode score $Q(\mathbf{A}, \mathbf{E}_{\boldsymbol{\mu}}) \leq \epsilon$. Consequently, the Attack Fooling Rate (AFR) is defined as the rate at which samples from $\mathbf{G}$ are fooling attacks. In addition to AFR, the algorithm returns the set $S_{\boldsymbol{\mu'}}$ of adversarial examples that can be used to diagnose the agent. The equality constraint in Eq (2) is very strict to include *all* fooling parameters $\boldsymbol{\mu'}$ of the fooling distribution. It acts as a perceptuality metric in our generalized attack to prevent unrealistic attacks. Next, we relax this equality constraint to leverage recent advances in GANs for learning an estimate of the distribution $\mathbf{P}_{\boldsymbol{\mu'}}$.

Algorithm 1: Generic Adversarial Attacks on Agents

Returns: Attack fooling Rate (AFR)
Requires: Agent $\mathbf{A}$, Adversary $\mathbf{G}$, Environment $\mathbf{E}_{\boldsymbol{\mu}}$, number of episodes T, training iterations L, test size M, fooling threshold ϵ
Training G: **for** $i \leftarrow 1$ **to** L **do**
  Sample $\boldsymbol{\mu}_i \sim \mathbf{G}$ and initialize $E_{\boldsymbol{\mu}_i}$ with initial state $\mathbf{s}_1$
  for $t \leftarrow 1$ **to** T **do**
    $E_{\boldsymbol{\mu}_i}$ produces observation $\mathbf{o_t}$ from $\mathbf{s_t}$
    $\mathbf{A}$ performs $\mathbf{a}_t(\mathbf{o_t})$ and receives $r_t \leftarrow R(\mathbf{s}_t, \mathbf{a}_t)$
    State updates: $\mathbf{s}_{t+1} \leftarrow E_{\boldsymbol{\mu}_i}(\mathbf{s}_t, \mathbf{a}_t)$
  end
  $\mathbf{G}$ receives the episode score $Q_i(\mathbf{A}, \mathbf{E}_{\boldsymbol{\mu}}) \leftarrow \sum_{t=1}^{T} r_t$
  Update $\mathbf{G}$ to solve for Eq (2)
end
Testing G: Initialize fooling counter $f \leftarrow 0$
for $j \leftarrow 1$ **to** M **do**
  sample $\boldsymbol{\mu}_j \sim \mathbf{G}$ and initialize $E_{\boldsymbol{\mu}_j}$ with initial state $\mathbf{s}_1$
  for $t \leftarrow 1$ **to** T **do**
    $\mathbf{a}_t(\mathbf{o_t})$; $r_t \leftarrow R(\mathbf{s}_t, \mathbf{a}_t)$; $\mathbf{s}_{t+1} \leftarrow E_{\boldsymbol{\mu}_j}(\mathbf{s}_t, \mathbf{a}_t)$
  end
  $Q_j(\mathbf{A}, \mathbf{E}_{\boldsymbol{\mu}_j}) \leftarrow \sum_{t=1}^{T} r_t$
  if $Q_j(\mathbf{A}, \mathbf{E}_{\boldsymbol{\mu}_j}) \leq \epsilon$ **then**
    $f \leftarrow f + 1$
  end
end
Returns: AFR = f/M

Black-Box Generative Adversarial Network

Generative Adversarial Networks (GANs) are a promising family of unsupervised techniques that can model complex domains, *e.g.* natural images (Goodfellow et al. 2014; Arjovsky, Chintala, and Bottou 2017; Gurumurthy, Kiran Sarvadevabhatla, and Venkatesh Babu 2017). GANs consist of a discriminator $\mathbf{D_x}$ and a generator $\mathbf{G_x}$ that are adversarially trained to optimize the loss $L_{GAN}(\mathbf{G_x}, \mathbf{D_x}, \mathbf{P}_X)$, where $\mathbf{P}_X$ is the distribution of images in domain X and $\mathbf{z} \in \mathbb{R}^c$ is a latent random Gaussian vector.

$$\min_{\mathbf{G_x}} \max_{\mathbf{D_x}} \;\; L_{\text{GAN}}(\mathbf{G_x}, \mathbf{D_x}, \mathbf{P}_X) = \quad (3)$$
$$\mathbb{E}_{\mathbf{x} \sim p_x(\mathbf{x})}[\log \mathbf{D_x}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - \mathbf{D_x}(\mathbf{G_x}(\mathbf{z})))]$$

$\mathbf{D_x}$ tries to determine if a given sample (*e.g.* image $\mathbf{x}$) is real (exists in the training dataset) or fake (generated by $\mathbf{G_x}$). On the other hand, $\mathbf{G_x}$ tries to generate samples that fool $\mathbf{D_x}$ (*e.g.* misclassification). Both networks are proven to converge when $\mathbf{G_x}$ can reliably produce the underlying distribution of the real samples (Goodfellow et al. 2014).

We propose to learn the fooling distribution $\mathbf{P}_{\boldsymbol{\mu'}}$ using a GAN setup, which we denote as black-box GAN (BBGAN). We follow a similar GAN objective but replace the image domain $\mathbf{x}$ by the semantic environment parameter $\boldsymbol{\mu}$. However, since we do not have direct access to $\mathbf{P}_{\boldsymbol{\mu'}}$, we propose a module called the *inducer*, which is tasked to produce the *induced set* $S_{\boldsymbol{\mu'}}$ that belongs to $\mathbf{P}_{\boldsymbol{\mu'}}$. In essence, the *inducer* tries to choose a parameter set which represents the fooling distribution to be learnt by the BBGAN as well as possible. In practice, the inducer selects the best fooling parameters (based on the Q scores of the agent under these parameters) from a set of randomly sampled parameters in order to construct this *induced set*. Thus, this setup relaxes Eq (2) to:

$$\begin{aligned} &\arg\min_{\mathbf{G}} \;\; \mathbb{E}_{\boldsymbol{\mu} \sim \mathbf{G}}[Q(\mathbf{A}, \mathbf{E}_{\boldsymbol{\mu}})] \\ &\text{s.t. } \{\boldsymbol{\mu} : \boldsymbol{\mu} \sim \mathbf{G}\} \subset \{\boldsymbol{\mu'} : \boldsymbol{\mu'} \sim \mathbf{P}_{\boldsymbol{\mu'}}\} \end{aligned} \quad (4)$$

So, the final BBGAN loss becomes:

$$\min_{\mathbf{G}_{\boldsymbol{\mu}}} \max_{\mathbf{D}_{\boldsymbol{\mu}}} \;\; L_{\text{BBGAN}}(\mathbf{G}_{\boldsymbol{\mu}}, \mathbf{D}_{\boldsymbol{\mu}}, S_{\boldsymbol{\mu'}}) = \quad (5)$$
$$\mathbb{E}_{\boldsymbol{\mu} \sim S_{\boldsymbol{\mu'}}}[\log \mathbf{D}_{\boldsymbol{\mu}}(\boldsymbol{\mu})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - \mathbf{D}(\mathbf{G}(z)))]$$

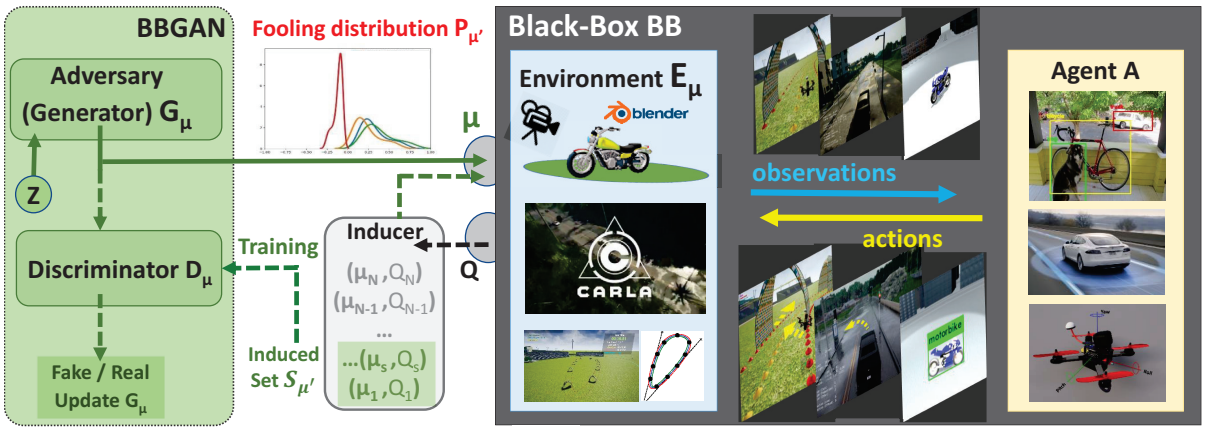

Figure 3: BBGAN: Learning Fooling Distribution of Semantic Environment Parameters. We learn an adversary $\mathbf{G}$, which samples semantic parameters $\boldsymbol{\mu}$ that parametrize the environment $\mathbf{E}_{\boldsymbol{\mu}}$, such that an agent $\mathbf{A}$ fails in a given task in $\mathbf{E}_{\boldsymbol{\mu}}$. The inducer produces the induced set $S_{\boldsymbol{\mu}'}$ from a uniformly sampled set Ω by filtering the lowest scoring $\boldsymbol{\mu}$ (according to Q value), and passing $S_{\boldsymbol{\mu}'}$ for BBGAN training. Note that $Q_1 \leq Q_s ..., \leq Q_N$, where $s = |S_{\boldsymbol{\mu}'}|$, $N = |\Omega|$. The inducer and the discriminator are only used during training (dashed lines), after which the adversary learns the fooling distribution $\mathbf{P}_{\boldsymbol{\mu}'}$. Three safety-critical applications are used to demonstrate this in three virtual environments: object detection (in Blender (Blender Online Community 2018)), self-driving cars (in CARLA (Dosovitskiy et al. 2017)), and autonomous racing UAVs (in Sim4CV (Müller et al. 2018b)).

Here, $\mathbf{G}_{\boldsymbol{\mu}}$ is the generator acting as the adversary, and $\mathbf{z} \in \mathbb{R}^m$ is a random variable sampled from a normal distribution. A simple inducer can be just a filter that takes a uniformly sampled set $\Omega = \{\boldsymbol{\mu}_i \sim \text{Uni}([\boldsymbol{\mu}_{\min}, \boldsymbol{\mu}_{\max}])\}_{i=1}^{i=N}$ and suggests the lowest Q-scoring $\boldsymbol{\mu}_i$ that satisfies the condition $Q(\boldsymbol{\mu}_i) \leq \epsilon$. The selected samples constitute the induced set $S_{\boldsymbol{\mu}'}$. The BBGAN treats the induced set as a training set, so the samples in $S_{\boldsymbol{\mu}'}$ act as virtual samples from the fooling distribution $\mathbf{P}_{\boldsymbol{\mu}'}$ that we want to learn. As the induced set size $S_{\boldsymbol{\mu}'}$ increases, the BBGAN learns more of $\mathbf{P}_{\boldsymbol{\mu}'}$. As $|S_{\boldsymbol{\mu}'}| \to \infty$, any sample from $S_{\boldsymbol{\mu}'}$ is a sample of $\mathbf{P}_{\boldsymbol{\mu}'}$ and the BBGAN in Eq (5) satisfies the strict condition in Eq (2). Consequently, sampling from $\mathbf{G}_{\boldsymbol{\mu}}$ would consistently fool agent $\mathbf{A}$. We show an empirical proof for this in the **supplement** and show how we consistently fool three different agents by samples from $\mathbf{G}_{\boldsymbol{\mu}}$ in the experiments. The number of samples needed for $S_{\boldsymbol{\mu}'}$ to be representative of $\mathbf{P}_{\boldsymbol{\mu}'}$ depends on the dimensionality d of $\boldsymbol{\mu}$. Because of the black-box and stochastic nature of $\mathbf{E}_{\boldsymbol{\mu}}$ and $\mathbf{A}$ (similar to other RL environments), we follow the random sampling scheme common in RL (Mania, Guy, and Recht 2018) instead of deterministic gradient estimation. In the experiments, we compare our method against baselines that use different approaches to solve Eq (2).

Special Cases of Adversarial Attacks

One can show that the generic adversarial attack framework detailed above includes well-known types of attacks as special cases, summarized in Table 1. In fact, the general setup allows for static agents (*e.g.* classifiers and detectors) as well as dynamic agents (*e.g.* an autonomous agent acting in a dynamic environment). It also covers pixel-wise image perturbations, as well as, semantic attacks that try to fool the agent in a more realistic scenario. The generic attack also allows for a more flexible way to define the attack success based on an application-specific threshold ϵ and the agent score Q. In the **supplement**, we provide more details on the inclusiveness of our generic setup and how it covers original pixel-level adversarial attack objectives.

Applications

Object Detection

Object detection is one of the core perception tasks commonly used in autonomous navigation. Based on its suitability for autonomous applications, we choose the very fast, state-of-the-art YOLOv3 object detector (Redmon and Farhadi 2018) as the agent in our SADA framework. We use the open-source software Blender to construct a scene based on freely available 3D scenes and CAD models. We pick an urban scene with an open area to allow for different rendering setups. The scene includes one object of interest as well as a camera and main light source directed toward the center of the object. The light is a fixed strength spotlight located at a fixed distance from the object. The material of each object is semi-metallic, which is common for the classes under consideration. The 3D collection consists of 100 shapes of 12 object classes (aeroplane, bench, bicycle, boat, bottle, bus, car, chair, dining table, motorbike, train, truck) from Pascal-3D (Xiang, Mottaghi, and Savarese 2014) and ShapeNet (Chang et al. 2015). At each iteration, one shape from the intended class is randomly picked and placed in the middle of the scene. The rendered image is then passed to YOLOV3 for detection. For the environment parameters, we use eight parameters that have shown to affect detection performance and frequently occur in real setups (refer to Figure 4).

Attack Variables	Pixel Adversarial Attack on Image Classifiers	Semantic Adversarial Attack on Object Detectors	Semantic Adversarial Attack on Autonomous Agents
Agent $\mathbf{A}$	K-class classifier $\mathbf{C} : [0,1]^n \rightarrow [l_1, l_2, ..., l_K]$ l_j : the softmax value for class j	K-class object detector $\mathbf{F} : [0,1]^n \rightarrow (\mathbb{R}^{N\times K}, \mathbb{R}^{N\times 4})$ N : number of detected objects	self-driving policy agent $\mathbf{A}$ *e.g.* network to regress controls
Parameters $\boldsymbol{\mu}$	the pixels noise added on attacked image $\mathbf{x}_i$	parameters describing the scene *e.g.* camera pose, object , light	parameters involved in the simulation *e.g.* road shape , weather , camera
Environment $\mathbf{E}_{\boldsymbol{\mu}}$	dataset Φ containing all images and their true class label $\Phi = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\lvert\Phi\rvert}$	dataset Φ containing all images and their true class label	simulation environment partially described by $\boldsymbol{\mu}$ that $\mathbf{A}$ navigates in for a target
Observation $\mathbf{o}_t$	attacked image after added noise $= \mathbf{x}_i + \boldsymbol{\mu}$, where $\mathbf{x}$, $\boldsymbol{\mu} \in \mathbb{R}^n$	the rendered image using the scene parameters $\boldsymbol{\mu}$	sequence of rendered images $\mathbf{A}$ observes during the simulation episode
Agent actions $\mathbf{a}_t(\mathbf{o}_t)$	predicted softmax vector of attacked image	predicted confidence of the true class label	steering command to move the car/UAV in the next step
Score $Q(\mathbf{A}, \mathbf{E}_{\boldsymbol{\mu}})$	the difference between true and predicted softmax	predicted confidence of the true class label	the average sum of rewards over five different episodes

Table 1: **Cases of Generic Adversarial Attacks**: variable substitutions that lead to known attacks.

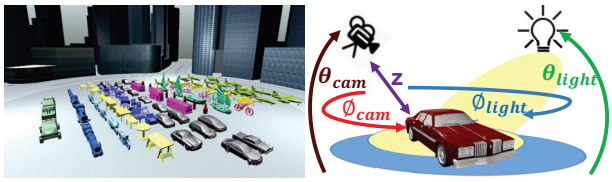

Figure 4: Object Detection Attack Setup: (*Left*): the 100 shapes from Pascal3D and ShapeNet of 12 object classes, used to uncover the failure cases of the YOLOV3 detector. (*Right*): the semantic parameters $\boldsymbol{\mu}$ defining the environment. (z): camera distance to the object, ($\phi_{\text{cam}}, \theta_{\text{cam}}, \phi_{\text{light}}, \phi_{\text{light}}$): camera azimuth, pitch and light source azimuth, and pitch angles respectively.

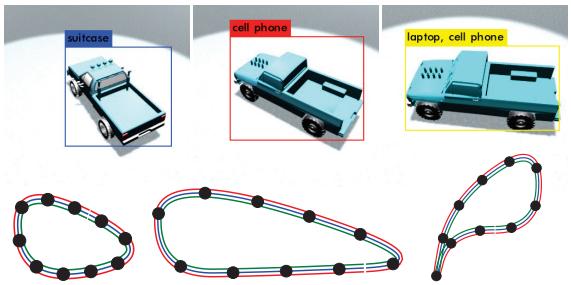

Figure 5: Qualitative Examples: (*Top*): BBGAN generated samples that fool YOLOV3 detector on the Truck class. (*Bottom*): BBGAN generated tracks that fool the UAV navigation agent.

Self-Driving

There is a lot of recent work in autonomous driving especially in the fields of robotics and computer vision (Franke 2017; Codevilla et al. 2018). In general, complete driving systems are very complex and difficult to analyze or simulate. By learning the underlying distribution of failure cases, our work provides a safe way to analyze the robustness of such a complete system. While our analysis is done in simulation only, we would like to highlight that sim-to-real transfer is a very active research field nowadays (Sadeghi and Levine 2017; Tobin et al. 2017). We use an autonomous driving agent (based on CIL (Codevilla et al. 2018)), which was trained on the environment $\mathbf{E}_{\mu}$ with default parameters. The driving-policy was trained end-to-end to predict car controls given an input image and is conditioned on high-level commands (*e.g. turn right at the next intersection*). The environment used is CARLA driving simulator (Dosovitskiy et al. 2017), the most realistic open-source urban driving simulator currently available. We consider the three common tasks of driving in a straight line, completing one turn, and navigating between two random points. The score is measured as the average success of five pairs of start and end positions. Since experiments are time-consuming, we restrict ourselves to three parameters, two of which pertain to the mounted camera viewpoint and the third controls the appearance of the environment by changing the weather setting (*e.g.* 'clear noon', 'clear sunset', 'cloudy after rain', etc.). As such, we construct an environment by randomly perturbing the position and rotation of the default camera along the z-axis and around the pitch axis respectively, and by picking one of the weather conditions. Intuitively, this helps measure the robustness of the driving policy to the camera position (*e.g.* deploying the same policy in a different vehicle) and to environmental conditions.

UAV Racing

In recent years, UAV (unmanned aerial vehicle) racing has emerged as a new sport where pilots compete in navigating small UAVs through race courses at high speeds. Since this is a very interesting research problem, it has also been picked up by the robotics and vision communities (Kaufmann et al. 2018). We use a fixed agent to autonomously fly through each course and measure its success as percentage of gates passed (Müller et al. 2018a). If the next gate was not reached within 10 seconds, we reset the agent at the last gate. We also record the time needed to complete the course. The agent uses a perception network that produces waypoints from image input and a PID controller to produce low-level controls. We use the general-purpose simulator for computer

	Object Detection				Autonomous Driving			UAV Track Generation		
	Bicycle	Motorbike	Truck	12-class avg	Straight	One Curve	Navigation	3 anchors	4 anchors	5 anchors
Full Set	14.6%	32.5%	56.8%	37.1 %	10.6%	19.5%	46.3%	17.0%	23.5%	15.8%
Random	13.3%	38.8%	73.8%	45.7%	8.0%	18.0%	48.0%	22.0%	30.0%	16.0%
Multi-Class SVM	20.0%	45.6%	70.8%	45.8%	96.0%	**100%**	**100%**	24.0%	30.0%	14.0%
GP Regression	17.6%	43.6%	83.6%	45.26%	**100%**	**100%**	**100%**	74.0%	94.0%	44.0%
Gaussian	19.6%	40.4%	72.4%	47.0%	54.0%	30.0%	64.0%	49.3%	56.0%	28.7%
GMM10%	26.0%	48.4%	75.2%	49.0%	90.0%	72.0%	98.0%	57.0%	63.0%	33.0%
GMM50%	16.4%	46.8%	72.0%	47.8%	92.0%	68.0%	**100%**	54.0%	60.0%	40.0%
Bayesian	48.0%	52.0%	75.6%	56.1%	-	-	-	-	-	-
BBGAN (vanilla)	44.0%	45.2%	90.8%	74.5%	**100%**	98.0%	98.0%	42.0%	94.0%	86.0%
BBGAN (boost)	**65.8%**	**82.0%**	**100%**	**80.5%**	**100%**	**100%**	**100%**	**86.0%**	**98.0%**	**92.0%**

Table 2: Attack Fooling Rate (AFR) Comparison: AFR of adversarial samples generated on three safety-critical applications: YOLOV3 object detection, self-driving, and UAV racing. For detection, we report the average AFR performance across all 12 classes and highlight 3 specific ones. For autonomous driving, we compute the AFR for the three common tasks in CARLA. For UAV racing, we compute AFR for race tracks of varying complexity (3, 4, or 5 anchor points describe the track). We see that our BBGAN outperforms all the baselines, and with larger margins for higher dimensional tasks (*e.g.* detection). Due to the expensive computations and sequential nature of the Bayesian baseline, we omit it for the two autonomous navigation applications. Best results are in **bold**.

vision applications, Sim4CV (Müller et al. 2018b). Here, we change the geometry of the race course environment rather than its appearance. We define three different race track templates with 3-5 2D anchor points, respectively. These points describe a second order B-spline and are perturbed to generate various race tracks populated by uniformly spaced gates. Please refer to the **supplement** for more details.

Experiments

BBGAN

Training. To learn the fooling distribution $\mathbf{P}_{\boldsymbol{\mu}'}$, we train the BBGAN using a vanilla GAN model (Goodfellow et al. 2014). Both, the generator $\mathbf{G}$ and the discriminator $\mathbf{D}$ consist of a MLP with 2 layers. We train the GAN following convention, but since we do not have access to the true distribution that we want to learn (*i.e.* real samples), we *induce* the set by randomly sampling N parameter vector samples $\boldsymbol{\mu}$, and then picking the s worst among them (according to the score Q). For object detection, we use $N = 20000$ image renderings for each class (a total of 240K images). Due to the computational cost, our dataset for the autonomous navigation tasks comprises only $N = 1000$ samples. For instance, to compute one data point in autonomous driving, we need to run a complete episode that requires 15 minutes. The induced set size is always fixed to be $s = 100$.

Boosting. We use a boosting strategy to improve the performance of our BBGAN. Our boosting strategy simply utilizes the samples generated by the previous stage adversary $\mathbf{G}_{k-1}$ in inducing the training set for the current stage adversary $\mathbf{G}_k$. This is done by adding the generated samples to Ω before training $\mathbf{G}_k$. The intuition here is that the main computational burden in training the BBGAN is not the GAN training itself, but computing the agent episodes, each of which can take multiple hours for the case of self-driving. For more details, including the algorithm, a mathematical justification and more experimental results please refer to the **supplement**.

Testing, Evaluation, and Baselines

To highlight the merits of BBGAN, we seek to compare it against baseline methods, which also aim to estimate the fooling distribution $\mathbf{P}_{\boldsymbol{\mu}'}$. In this comparative study, each method produces M fooling/adversarial samples (250 for object detection and 100 for self-driving and UAV racing) based on its estimate of $\mathbf{P}_{\boldsymbol{\mu}'}$. Then, the *attack fooling rate* (AFR) for each method is computed as the percentage of the M adversarial samples that fooled the agent. To determine whether the agent is fooled, we use a fooling rate threshold $\epsilon = 0.3$ (Chen et al. 2018), $\epsilon = 0.6$, and $\epsilon = 0.7$ for object detection, self-driving, and UAV racing, respectively. In the following, we briefly explain the baselines. **Random.** We uniformly sample random parameters $\boldsymbol{\mu}$ within an admissible range that is application dependent.

Gaussian Mixture Model (GMM). We fit a full covariance GMM of varying Gaussian components to estimate the distribution of the samples in the induced set $S_{\boldsymbol{\mu}'}$. The variants are denoted as Gaussian (one component), GMM10% and GMM50% (number of components as percentage of the samples in the induced set).

Bayesian. We use the Expected Improvement (EI) Bayesian Optimization algorithm (Jones, Schonlau, and Welch 1998) to minimize the score Q for the agent. The optimizer runs for 10^4 steps and it tends to gradually sample more around the global minimum of the function. We use the last $N = 1000$ samples to generate the induced set $S_{\boldsymbol{\mu}'}$ and then learn a GMM with different Gaussian components. Finally, we sample M parameter vectors from the GMMs and report results for the best model.

Multi-Class SVM. We bin the score Q into 5 equally sized bins and train a multi-class SVM classifier on the complete set Ω to predict the correct bin. We then randomly sample parameter vectors $\boldsymbol{\mu}$, classify them, and sort them by the predicted score. We pick M samples with the lowest Q score.

Gaussian Process Regression. Similar to the SVM case, we train a Gaussian Process Regressor (Martin, Wang, and Englot 2018) with an exponential kernel to regress the scores

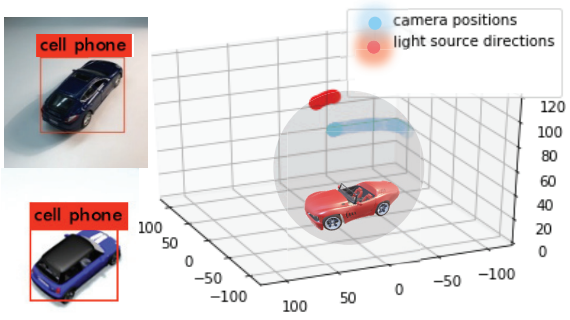

Figure 6: Visualization of the Fooling Distribution. (*right*): We plot the camera positions and light source directions of 250 sampled parameters in a 3D sphere around the object. (*left*): We show how real photos of a toy car, captured from the same angles as rendered images, confuse the YOLOV3 detector in the same way.

Q from the corresponding $\boldsymbol{\mu}$ parameters that generated the environment on the dataset Ω.

Results

Table 2 summarizes the AFR results for the aforementioned baselines and our BBGAN approach across all three applications. For object detection, we show 3 out of 12 classes and report the average across all classes. For autonomous driving, we report the results on all three driving tasks. For UAV racing, we report the results for three different track types, parameterized by an increasing number of 2D anchor points (3, 4 and 5) representing $\boldsymbol{\mu}$. Our results show that we consistently outperform the baselines, even the ones that were trained on the complete set Ω rather than the smaller induced set $S_{\boldsymbol{\mu}'}$, such as the multi-class SVM and the GP regressor. While some baselines perform well on the autonomous driving application where $\boldsymbol{\mu}$ consists of only 3 parameters, our approach outperforms them by a large margin on the tasks with higher dimensional $\boldsymbol{\mu}$ (*e.g.* object detection and UAV racing with 5-anchor tracks). Our boosting strategy is very effective and improves results even further with diminishing returns for setups where the vanilla BBGAN already achieves a very high or even the maximum success rate.

To detect and prevent mode collapse, a GAN phenomenon where the generator collapses to generate a single point, we do the following. (1) We visualize the Nearest Neighbor (NN) of the generated parameters in the training set as in Figure 7. (2) We visualize the distributions of the generated samples and ensure their variety as in Figure 6. (3) We measure the average standard deviation per parameter dimension to make sure it is not zero. (4) We visualize the images/tracks created by these parameters as in Figure 5.

Analysis

Diagnosis. The usefulness of SADA lies in that it is not only an attacking scheme using BBGAN, but also serves as a diagnosis tool to assess the systematic failures of agents. We perform diagnostic studies (refer to Figure 6) to identify cases of systematic failure for the YOLOv3 detector.

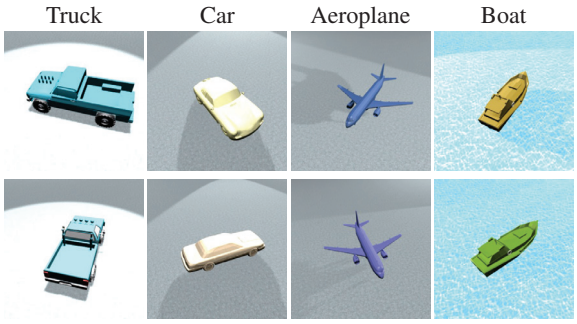

Figure 7: Nearest Neighbor in Training: (*top*): generated fooling samples by our BBGAN for 4 different classes. (*bottom*): the corresponding NN from the training set. We can see that our BBGAN generates novel fooling parameters that are not present in training.

Transferability. To demonstrate the transferability of the fooling parameter distribution to the real-world, we photograph toy models using a standard mobile phone camera and use office desk lights analogous the light source of the virtual environment. We orient the camera and the lighting source according to the fooling distribution learned from the virtual world. In Figure 6, we show a real-world photo of the toy car and the corresponding rendered virtual-world image, both of which are similarly fooled with the same erroneous class label. Please refer to the **supplement** for a similar analysis of other classes, the transfer of attacks between different CAD models and the effect of occlusion on detection.

Nearest Neighbor Visualization. In Figure 7, we visualize the NN in the parameter space for four different generated samples by our BBGAN. We see that the generated and the NN in training are different for the 4 samples with L_2 norm differences of (0.76,0.60,0.81,0.56) and (378,162,99,174) in parameter space and in image space respectively (all range from -1 to 1). This shows that our BBGAN can generate novel examples that fool the trained agent.

Insights and Future Work

Object Detection with YOLOV3. For most objects, top-rear or top-front views of the object tend to fool the YOLOV3 detector. The color of the object does not play a significant role in fooling the detector, but usually colors that are closer to the background color tend to be preferred by the BBGAN samples.

Self-Driving. Weather is the least important parameter for fooling the driving policy indicating that the policy was trained to be insensitive to this factor. Interestingly, the learned policy is very sensitive to slight perturbations in the camera pose (height and pitch), indicating a systemic weakness that should be ratified with more robust training.

UAV Autonomous Navigation. We observe that the UAV fails if the track has very sharp turns. This makes intuitive sense and the results that were produced by our BBGAN consistently produce such tracks. While the tracks that are only parameterized by three control points can not achieve sharp turns, our BBGAN is still able to make the UAV agent fail by placing the racing gates very close to each other,

thereby increasing the probability of hitting them.

Future Work. Our work can be extended to other AI agents and test for their semantic vulnerability to such attacks. This can be used to establish more interpretable deep models and allow for safety-tests for AI models before deployment in real world safety-critical applications.

Acknowledgments

This work was supported by the King Abdullah University of Science and Technology (KAUST) Office of Sponsored Research under Award No. RGC/3/3570-01-01.

References

Alcorn, M. A.; Li, Q.; Gong, Z.; Wang, C.; Mai, L.; Ku, W.-S.; and Nguyen, A. 2019. Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein gan. arXiv:1701.07875.

Blender Online Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam.

Carlini, N., and Wagner, D. 2017. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP)*.

Chang, A. X.; Funkhouser, T.; Guibas, L.; Hanrahan, P.; Huang, Q.; Li, Z.; Savarese, S.; Savva, M.; Song, S.; Su, H.; Xiao, J.; Yi, L.; and Yu, F. 2015. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago.

Chen, S.; Cornelius, C.; Martin, J.; and Chau, D. H. 2018. Robust physical adversarial attack on faster R-CNN object detector. *CoRR* abs/1804.05810.

Codevilla, F.; Müller, M.; Dosovitskiy, A.; López, A.; and Koltun, V. 2018. End-to-end driving via conditional imitation learning. In *ICRA*.

Dosovitskiy, A.; Ros, G.; Codevilla, F.; López, A.; and Koltun, V. 2017. CARLA: An open urban driving simulator. In *CoRL*.

Franke, U. 2017. Autonomous driving. In *Computer Vision in Vehicle Technology*.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 2672–2680.

Goodfellow, I.; Shlens, J.; and Szegedy, C. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Gurumurthy, S.; Kiran Sarvadevabhatla, R.; and Venkatesh Babu, R. 2017. Deligan : Generative adversarial networks for diverse and limited data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hamdi, A., and Ghanem, B. 2019. Towards analyzing semantic robustness of deep neural networks. *CoRR* abs/1904.04621.

Held, D.; Geng, X.; Florensa, C.; and Abbeel, P. 2017. Automatic goal generation for reinforcement learning agents. *CoRR* abs/1705.06366.

Jones, D. R.; Schonlau, M.; and Welch, W. J. 1998. Efficient global optimization of expensive black-box functions. *J. of Global Optimization* 13(4):455–492.

Kaufmann, E.; Loquercio, A.; Ranftl, R.; Dosovitskiy, A.; Koltun, V.; and Scaramuzza, D. 2018. Deep drone racing: Learning agile flight in dynamic environments. In Billard, A.; Dragan, A.; Peters, J.; and Morimoto, J., eds., *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, 133–145. PMLR.

Kurakin, A.; Goodfellow, I. J.; and Bengio, S. 2016. Adversarial machine learning at scale. *CoRR* abs/1611.01236.

Mania, H.; Guy, A.; and Recht, B. 2018. Simple random search provides a competitive approach to reinforcement learning. *CoRR* abs/1803.07055.

Martin, J.; Wang, J.; and Englot, B. J. 2018. Sparse gaussian process temporal difference learning for marine robot navigation. *CoRR* abs/1810.01217.

Morimoto, J., and Doya, K. 2001. Robust reinforcement learning. In *Advances in Neural Information Processing Systems*, 1061–1067.

Müller, M.; Casser, V.; Smith, N.; Michels, D. L.; and Ghanem, B. 2017. Teaching UAVs to Race Using Sim4CV. *ArXiv e-prints*.

Müller, M.; Casser, V.; Smith, N.; Michels, D. L.; and Ghanem, B. 2018a. Teaching UAVs to Race: End-to-End Regression of Agile Controls in Simulation. In *European Conference on Computer Vision Workshop (ECCVW)*.

Müller, M.; Casser, V.; Lahoud, J.; Smith, N.; and Ghanem, B. 2018b. Sim4cv: A photo-realistic simulator for computer vision applications. *Int. J. Comput. Vision* 126(9):902–919.

Pinto, L.; Davidson, J.; Sukthankar, R.; and Gupta, A. 2017. Robust adversarial reinforcement learning. *CoRR* abs/1703.02702.

Redmon, J., and Farhadi, A. 2018. Yolov3: An incremental improvement. *CoRR* abs/1804.02767.

Sadeghi, F., and Levine, S. 2017. CAD2RL: Real single-image flight without a single real image. In *RSS*.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I. J.; and Fergus, R. 2013. Intriguing properties of neural networks. *CoRR* abs/1312.6199.

Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; and Abbeel, P. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*.

Xiang, Y.; Mottaghi, R.; and Savarese, S. 2014. Beyond pascal: A benchmark for 3d object detection in the wild. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*.

Xiang, C.; Qi, C. R.; and Li, B. 2019. Generating 3d adversarial point clouds. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xiao, C.; Yang, D.; Li, B.; Deng, J.; and Liu, M. 2019. Meshadv: Adversarial meshes for visual recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zeng, X.; Liu, C.; Wang, Y.-S.; Qiu, W.; Xie, L.; Tai, Y.-W.; Tang, C.-K.; and Yuille, A. L. 2019. Adversarial attacks beyond the image space. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.