Learning When to Use Adaptive Adversarial Image Perturbations against Autonomous Vehicles

Hyung-Jin Yoon[1], Hamidreza Jafarnejadsani[2], and Petros Voulgaris[1]

***Abstract*— Deep neural network (DNN) models are widely used in autonomous vehicles for object detection using camera images. However, these models are vulnerable to adversarial image perturbations. Existing methods for generating these perturbations use each incoming image frame as the decision variable, resulting in a computationally expensive optimization process that starts over for each new image. Few approaches have been developed for attacking online image streams while considering the physical dynamics of autonomous vehicles, their mission, and the environment. To address these challenges, we propose a multi-level stochastic optimization framework that monitors the attacker's capability to generate adversarial perturbations. Our framework introduces a binary decision attack/not attack based on the attacker's capability level to enhance its effectiveness. We evaluate our proposed framework using simulations for vision-guided autonomous vehicles and actual tests with a small indoor drone in an office environment. Our results demonstrate that our method is capable of generating real-time image attacks while monitoring the attacker's proficiency given state estimates.**

***Index Terms*— Adversarial Machine Learning, Reinforcement learning, Autonomous Vehicle**

I. Introduction

Machine learning (ML) tools that detect objects using high-dimensional sensors, such as camera images [1] or point clouds measured by LiDAR [2], are extensively used in autonomous vehicles [3], [4]. As vision-based autonomous vehicles become more integrated into society, it is crucial to ensure the robustness of these systems, which rely on various sensor signals in uncertain environments. Analyzing worst-case scenarios within uncertainties has been a useful approach to robustify control systems [5] and reinforcement learning [6]. To follow this approach, researchers have revealed the vulnerability of machine learning methods, especially deep learning tools developed for computer vision tasks such as object detection and classification, to data perturbed by adversaries. For instance, small perturbations can be added to images that are unnoticeable to human eyes but result in incorrect image classifications [7], [8], [9]. Moreover, recent works have demonstrated adversarial image perturbations against autonomous vehicles, including (1) modifying physical objects, such as putting stickers on a road [10] or a road sign [11], to fool an ML image classifier or end-to-end vision-based autonomous car; and (2) fooling object tracking algorithms in autonomous driving systems [12]. Adversarial machine learning commonly focuses on creating stealthy and natural-looking perturbations to evade human detection. Such attacks are designed to resemble out-of-distribution samples that may occur in real-world environments. As a consequence, ensuring the robustness of ML-based autonomous vehicle systems against adversarial attacks has become increasingly critical.

While the aforementioned adversarial image perturbations against autonomous cars [10], [11], [12] successfully reveal weaknesses in vision-guided navigation in autonomous vehicles, these perturbed images are generated offline. However, offline methods [11], [12] do not consider the effect of real-time attacks on dynamically changing environments during driving or flight of the vehicles. To prevent accidents [13] caused by vision-guided autonomous vehicles due to defective perception systems and their vulnerabilities, we need to study attack and defense techniques that go beyond offline methods for deep neural networks.

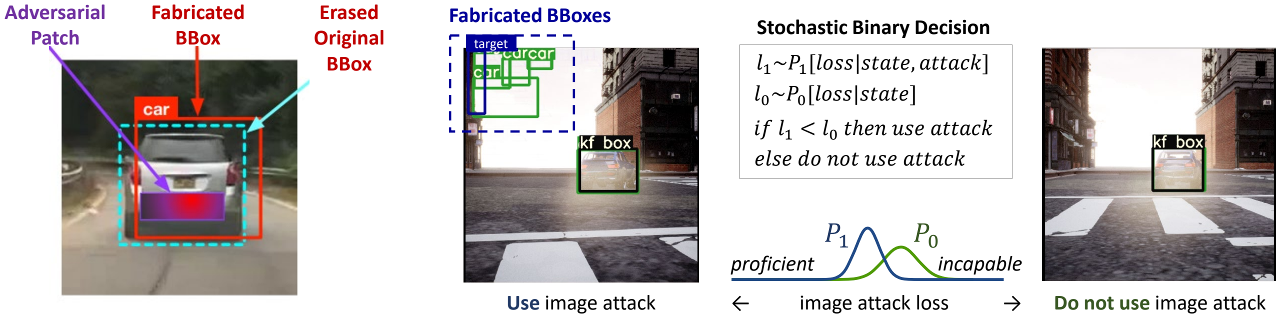
There are two approaches to generating adversarial image perturbations, depending on the attacker's access to the victim perception model. In the *white-box* attack approach, the attacker has full access to the victim ML classifier (or object detector) and generates adversarial image perturbations through iterative optimization [8], [12]. In this method, images are the decision variables of optimization, and the training loss function is reused with incorrect labels set by the attacker. The optimization takes iterative gradient steps with respect to the image variables calculated using back-propagation through the known victim ML classifier [8] (or object detector [12]). On the other hand, in the *black-box* approaches [14], [15], the attacker only has access to input and output pairs of the victim model and must estimate the gradient. However, estimating the gradients in *black-box* attacks requires a large number of samples, which may not be available from autonomous systems operating in dynamic environments.

Statement of Contribution: To our knowledge, this paper is the first to propose a stealthy attack scheme on image streams used for object detection/tracking in autonomous vehicles (e.g., self-driving cars and drones) that can be deployed *online*, and the *physical dynamics of the system* and the *varying surrounding environment* are taken into account in the optimization phase of the attack scheme. In this paper, we present a framework that utilizes generative adversarial networks (GANs) to generate adversarial images in real-time scenarios without the need for iterative steps.

*Research supported by NSF CPS #1932529, NSF CMMI #1663460, NSF CMMI #2137753 and UNR internal funding.

[1]Hyung-Jin Yoon and Petros Voulgaris are with the Department of Mechanical Engineering, University of Nevada, Reno, NV 89557, USA {hyungjiny, pvoulgaris}@unr.edu

[2]Hamidreza Jafarnejadsani is with the Department of Mechanical Engineering, Stevens Institute of Technology, NJ 07030, USA hjafarne@stevens.edu



(a) Adversarial patch example.

(b) Choosing when to use the adversarial image perturbation .

Fig. 1: Image attacks: (a) Adversarial patch in [12], (b) Adversarial perturbation with binary decision in this paper. *kf box* denotes Kalman filtered bounding box (BBox).

Building on the approach outlined in [16], our proposed multi-level framework consists of several components. First, the GAN functions as an online image generator. Second, a reinforcement learning agent is trained to misguide the vehicle according to the adversary's objective. Lastly, a binary decision-maker determines when to use image attacks based on the proficiency of the image attack generator, given the current state estimate. Our framework provides a more efficient and practical alternative to iterative *white-box* methods for generating adversarial images.

Our contributions can be summarized as follows:

- We propose a *real-time* adversarial image perturbation framework that allows for implementation on real-world robots, in contrast to existing offline methods.
- We introduce a *state estimation-based reinforcement learning* approach that learns to decide on the image frame area to fabricate bounding boxes. This approach eliminates the need for manual annotation of patch areas.
- We incorporate a constraint on the strength of the image perturbation, making the attacked image frame *less noticeable and more stealthy* compared to existing methods. This is demonstrated in Figure 1.

II. Related Works

Adversarial image perturbations have been extensively studied to attack autonomous vehicles that rely on camera images for navigation [17], [12], [18]. For instance, in [17], an optimization problem was formulated to place black marks on the road, which caused an end-to-end autonomous driving car to veer off the road in a virtual reality environment. This approach was inspired by the demonstration of attacking Tesla's autonomous driving systems with just three small stickers [10]. In another work [12], the authors demonstrated the effectiveness of a *white-box* adversarial image perturbation method on object tracking of an autonomous system that uses *Kalman* filter (KF) to disrupt the object tracking. This method was also shown to be effective in attacking an industry-level perception module that uses vision-based object detection fused with LIDAR, GPS, and IMU [18].

The aforementioned *white-box* methods [12], [18] require full sets of iterative optimization computations for every new image, rendering them unsuitable for dynamic environments with evolving situations and control loops of autonomous vehicles. These approaches do not consider the varying computation time of the iterative optimizations, which might have different termination steps for online applications. Additionally, the attack methods in [12], [18] often require additional state information that is not always readily available, unlike the image stream. For instance, generating the adversarial patch in Figure 1a in [12] requires the attacker to know the exact anchor index associated with the target bounding box (BBox) and the location to place the BBox. As mentioned in the *open review* [19] by the authors in [12], the adversarial patch area was manually annotated in each video frame.

Although there are various other adversarial image attack methods available, many of them are offline methods that require additional information such as labeled training data to generate adversarial images.

III. Real-time Adversarial Image Attack

Our goal is to develop a real-time solution that can learn to generate adversarial image perturbations and decide when to use the attack based on the proficiency of the attack generator, as illustrated in Figure 1b. The adversarial image perturbations are designed to manipulate the perception of autonomous vehicles to misguide them according to the adversary's objectives, such as causing collisions or making the vehicle deviate from its original path. To formally formulate the problem, We consider the following assumptions and settings.

A. Problem description and proposed framework

We focus on an autonomous vehicle that utilizes an object detection ML method to track a target object using camera images, as shown in Figure 2. We used a recent version of the *YOLO* object detection model [1], which was downloaded from [20], for our experiments[1]. The output of the object detector network is a multi-dimensional tensor that is processed using non-max suppression [1] to obtain a list of bounding box coordinates. The box with the highest confidence score for the target class is then selected from the list of detected bounding boxes to generate tracking

[1] Another popular object detection model, *Faster R-CNN*, can be attacked using similar White box attack method as in [21]. Hence, our proposed method can be implemented with *Faster R-CNN*.

control commands. The autonomous guidance system uses the vehicle's actuators, including the acceleration pedal, brake, and steering wheel, to keep the target's bounding box centered in the camera view and within a specified size range. Consequently, the vehicle moves towards and tracks the target object.

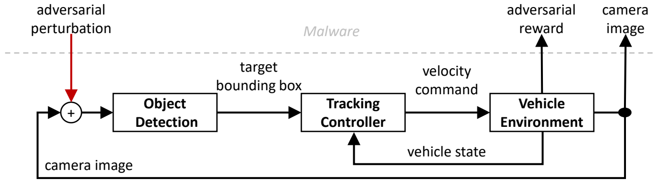


Fig. 2: Attacker (malware) and victim system (guidance)

We assume the adversary's objective is to disrupt the target tracking control in Figure 2. The attacker is assumed to be embedded as *Malware* and has access to the image stream, enabling them to perturb the input to the object detection module of the victim system, as illustrated in Figure 2. Given the image streams denoted as $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_t$, the attacker's goal is to generate adversarial image perturbations $\mathbf{w}_0, \mathbf{w}_1, ..., \mathbf{w}_t$ that mislead the victim vehicle according to adversarial objectives expressed in terms of adversarial rewards $r_1, r_2, ..., r_t$. The reward function is based on the vehicle's state, such as position, velocity, or collision states, and actions that involve the coordinates used to fabricate the bounding boxes through the image attack generator, as shown in Figure 3. These rewards are crucial for applying *reinforcement learning* (RL), which learns the correlation between actions and rewards for different states of the system. In this framework, a binary decision-maker determines when to attack based on the attack proficiency (represented as loss in Figure 3). The problem addressed in this framework can be summarized as follows:

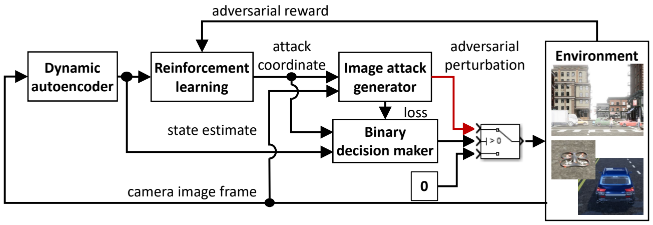


Fig. 3: Image attack framework with binary decision maker.

Problem: Develop machine learning methods that learn to increase the sum of rewards $\{r_t\}$ for the adversary by generating adversarial perturbation $\{\mathbf{w}_t\}$ while selecting when to use the attack at the time step t, as shown in Figure 3. The ML method assumes to use only the image stream $\{\mathbf{x}_t\}$ from the autonomous vehicle that has a guidance system and malware shown in Figure 2.

B. *Online image attack with binary decision making*

Our framework involves binary decision-making that depends on the proficiency of the image attack generator. This type of decision-making belongs to the multi-armed bandit class of problems [22], where the decision-maker selects the most profitable action. However, unlike the classical multi-armed bandit, where the rewards are generated from independent-stationary distributions, our decision-maker must consider non-stationary system state distributions. Specifically, given the attack coordinate chosen by RL and the state estimate from the dynamic autoencoder, the decision-maker must determine whether using the attack is profitable or not. To tackle this challenge, the authors in [23] used a deep neural network (DNN) to learn the correlation between the state, decision, and profit. They also employed random dropout [24] with the DNN to estimate the profit distributions for each decision. This multi-armed bandit algorithm, which uses DNN with random dropout, is known as *Neural Thompson Sampling* (NTS).

We sought to implement NTS for binary decision-making, using the proficiency of the image attack generator as the profits in the multi-armed bandit. While the direct application of NTS to our framework is appealing, there is a causality issue to consider. Specifically, the loss value is independent of binary decision-making, as it depends on the attack coordinates and the image frame. In our experiments, we tested NTS, but it did not demonstrate the desired behavior of selecting to attack when the expected loss value is low.

Therefore, we propose an alternative method to NTS that involves comparing two conditional expectations. Specifically, our method compares $E[l_t|\mathbf{h}_t, \mathbf{a}_t]$ with $E[l_t|\mathbf{h}_t]$. Here, l_t represents the loss function used to measure the proficiency of the image attack GAN. The state estimates that are low-dimensional representations of all previous observations, denoted by $\mathbf{h}_t$, are obtained using the dynamic autoencoder shown in Figure 3. Since the true states $\mathbf{s}_t$ are only partially observed through the image $\mathbf{x}_t$, $\mathbf{h}_t$ provides a better estimate of the state. Additionally, $\mathbf{a}_t$ represents the action determined by reinforcement learning agent in Figure 3. This action is the attack coordinate, which is the position and size of the fabricated bounding box. The goal of this approach is to compare the expected loss given the attack coordinate $\mathbf{a}_t$ suggested by RL with the expected loss averaged over all other possible attack coordinates. If the expected loss given $\mathbf{a}_t$ is lower than the average loss, then $\mathbf{a}_t$ is considered a promising attack coordinate to be used at this point. We refer to this decision-making method as *Conditional Sampling* (CS). The loss estimation and decision-making procedure of CS are as follows:

Estimation: The estimation for CS involves the following optimizations:

$$\begin{aligned} &\arg\min_{\theta^{\text{dec}}} \|l_t - \hat{l}_0(\mathbf{h}_t;\theta^{\text{dec}})\|^2, \\ &\arg\min_{\theta^{\text{dec}}} \|l_t - \hat{l}_1(\mathbf{h}_t, \mathbf{a}_t;\theta^{\text{dec}})\|^2, \end{aligned} \tag{1}$$

where $\hat{l}_0$ and $\hat{l}_1$ are DNNs trained to predict the loss functions values l_t given the state estimate $\mathbf{h}_t$ and the attack coordinate $\mathbf{a}_t$ respectively. The DNNs have parameters denoted as θ^{dec} that need to be optimized.

Decision making: The decision to launch an attack is determined by random sampling. To obtain sample image attack losses $\tilde{l}_0$ and $\tilde{l}_1$, we follow the same approach as in NTS, using the current state estimate $\mathbf{h}_t$ and the attack coordinate $\mathbf{a}_t$. Specifically, we generate output samples by applying random dropout in DNNs, i.e., $\hat{l}_0$ and $\hat{l}_1$, and estimate Gaussian distributions based on these samples. We then sample from the estimated Gaussian distributions to obtain $\tilde{l}_0$ and $\tilde{l}_1$, which can be expressed as:

$$\tilde{l}_0 \sim \hat{l}_0(\mathbf{h}_t;\theta^{\text{dec}}) \quad \text{and} \quad \tilde{l}_1 \sim \hat{l}_1(\mathbf{h}_t,\mathbf{a}_t;\theta^{\text{dec}}). \tag{2}$$

Further details of the sampling procedure can be found in [23]. To make a decision, we select the option with the lower loss value, i.e., if $\tilde{l}_0 < \tilde{l}_1$, then the attack will not be launched in the time step; otherwise, the image attack will be performed. The conditional probability distributions for the samplings are denoted as $P_0[l_t|\mathbf{h}_t]$ and $P_1[l_t|\mathbf{h}_t,\mathbf{a}_t]$, as shown in Figure 1b.

The proposed framework integrates estimation models for binary decision-making within computation networks consisting of DNNs, as depicted in Figure 4. A major advantage of this framework is the ability to generate real-time adversarial image perturbations through recursive computations. The process involves feeding the image $\mathbf{x}_t$ at time t into encoder networks, $\mathbf{Enc}_0$ and $\mathbf{Enc}_1$, for dimension reduction. $\mathbf{Enc}_0$ is used for state estimation, while $\mathbf{Enc}_1$ generates the perturbed image $\mathbf{w}_t$. The dynamic autoencoder comprises $\mathbf{Enc}_0$, $\mathbf{GRU}$ (gated recurrent unit), and $\mathbf{Dec}_0$ (decoder). The $\mathbf{GRU}$ recursively updates the hidden state $\mathbf{h}_t$ using the encoded image $\mathbf{Enc}_0(\mathbf{x}_t)$ and the high-level attack action $\mathbf{a}_t$, as shown in Figure 4. The estimated state information in $\mathbf{h}_t$ is then used by the actor (policy) to generate the high-level action, i.e., $\mathbf{a}_t = \mathbf{Actor}(\mathbf{h}_t)$.

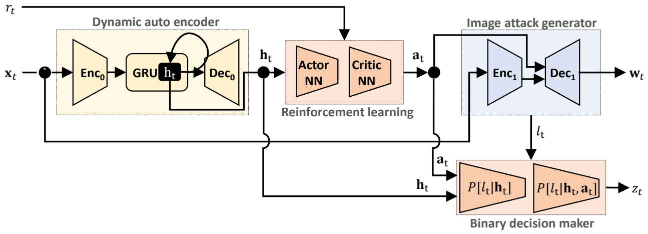


Fig. 4: Multi-level image attack computation network. The computation network for multi-level image attack consists of image encoders and decoders within the dynamic autoencoder and image attack generator, which are adapted from [25].

As shown in Figure 4, the adversarial image perturbation $\mathbf{w}_t$ is generated by $\mathbf{Dec}_1$ using the high-level attack action $\mathbf{a}_t$ and another encoded image from $\mathbf{Enc}_1$, i.e., $\mathbf{w}_t = \mathbf{Dec}_1(\mathbf{Enc}_1(\mathbf{x}_t),\mathbf{a}_t)$. The perturbed image frame is obtained by applying the perturbation to the original image with a scale factor α, i.e., $\tilde{\mathbf{x}}_t = \max(\min(\mathbf{x}_t + \alpha\mathbf{w}_t, 1), 0)$. The binary decision maker selects the decision variable z_t using the conditional sampling (CS) described in (2), where $z_t = 1$ indicates that the attack is used and $z_t = 0$ indicates that the attack is not used.

The recursive process of generating adversarial image perturbation using only camera image is summarized in Algorithm 1. The entire computation at each time-step uses only the current observation or the state values in the previous time-step without iterative optimization, enabling real-time generation of image attacks.

Algorithm 1 Recursive Image Attack

```
Initialize: $t \leftarrow 0$ ; load the pre-trained parameters of the recursive attack networks.
repeat
    Generate attack command using RL policy (Actor)
        $\mathbf{a}_t \leftarrow \mathbf{Actor}(\mathbf{h}_t)$
    Encode the observed image $\mathbf{x}_t$ from the environment
        $\zeta_t \leftarrow \mathbf{Enc}_1(\mathbf{x}_t)$
    Generate adversarial image perturbation
        $\mathbf{w}_t \leftarrow \mathbf{Dec}_1(\zeta_t,\mathbf{a}_t)$
    Feed $\mathbf{w}_t$ to the environment and get new image $\mathbf{x}_{t+1}$
        $\mathbf{x}_{t+1},\mathbf{s}_{t+1},r_t,\text{done} \leftarrow \mathbf{Environment}(\mathbf{s}_t,\mathbf{w}_t)$
    Recursively update the state predictor $\mathbf{h}_{t+1}$ with $\mathbf{x}_{t+1}$
        $\mathbf{h}_{t+1} \leftarrow \mathbf{GRU}(\mathbf{h}_t,\mathbf{Enc}_0(\mathbf{x}_{t+1}),\mathbf{a}_t)$
    Sample from the conditional distribution as in (2), i.e.,
        $l_0 \sim P_0[l_t|\mathbf{h}_t]$ and $l_1 \sim P_1[l_t|\mathbf{h}_t,\mathbf{a}_t]$.
    Use $\mathbf{w}_t$ if $l_0 < l_1$. Otherwise do not use it.
until done is True, i.e., the episode terminates with a terminal condition.
```

C. Multi-time scale optimization to train the attacker

We employ a multi-level stochastic optimization approach that separates the time scales of the updates for the various components depicted in Figure 3. Our stochastic optimization method trains the multi-level image attack computational networks illustrated in Figure 4. During training, the learning components and the environment are coupled and update their parameters simultaneously. The choice of time scales for the updates can have a significant impact on the behavior of the multi-time scale optimization process. For instance, in actor-critic [26], the critic has a faster update rate than the actor. In contrast, in the generative adversarial network described in [27], the discriminator has a faster update rate than the generator. Following the heuristics and theories described in [26], [27], we set slower parameter update rates for the lower-level components.

Let us denote the parameters of the various components as follows: θ_n^{img} represents the parameters of $\mathbf{Enc}_1(\cdot)$ and $\mathbf{Dec}_1(\cdot)$, θ_n^{sys} represents the parameters of the dynamic autoencoder comprising $\mathbf{Enc}_0(\cdot)$, $\mathbf{GRU}(\cdot)$, and $\mathbf{Dec}_0(\cdot)$, θ_n^{actor} represents the parameters of the actor denoted by $\mathbf{Actor}(\cdot)$, and θ_n^{critic} represents the parameters of the critic denoted by $Q(\cdot,\cdot)$, which is the action-value function for policy evaluation. We update these parameters using different step sizes, based on the pace of the update rates, as follows:

$$\begin{array}{rcrcll}
\theta_{n+1}^{\text{img}} & = & \theta_n^{\text{img}} & + & \varepsilon_n^{\text{img}} & S_n^{\text{img}}(\mathscr{M}_{\text{trajectory}}) \\
\theta_{n+1}^{\text{dec}} & = & \theta_n^{\text{dec}} & + & \varepsilon_n^{\text{dec}} & S_n^{\text{dec}}(\mathscr{M}_{\text{decision}}) \\
\theta_{n+1}^{\text{actor}} & = & \theta_n^{\text{actor}} & + & \varepsilon_n^{\text{actor}} & S_n^{\text{actor}}(\mathscr{M}_{\text{transition}}) \\
\theta_{n+1}^{\text{critic}} & = & \theta_n^{\text{critic}} & + & \varepsilon_n^{\text{critic}} & S_n^{\text{critic}}(\mathscr{M}_{\text{transition}}) \\
\theta_{n+1}^{\text{sys}} & = & \theta_n^{\text{sys}} & + & \varepsilon_n^{\text{sys}} & S_n^{\text{sys}}(\mathscr{M}_{\text{trajectory}})
\end{array} \tag{3}$$

where the update functions S_n^{img}, S_n^{sys}, S_n^{actor}, S_n^{critic} and S_n^{dec} are stochastic gradients with loss functions (to be described in following sections) calculated with data samples from the replay buffers, i.e., $\mathscr{M}_{\text{trajectory}}$, $\mathscr{M}_{\text{transition}}$, and $\mathscr{M}_{\text{decision}}$. The replay buffers store a finite number of recently observed tuples of $(\mathbf{x}_t, \mathbf{a}_t)$, $(\mathbf{h}_{t-1}, \mathbf{a}_t, r_t, \mathbf{h}_t)$, and $(\mathbf{h}_t, \mathbf{a}_t, l_t)$ in $\mathscr{M}_{\text{trajectory}}$, $\mathscr{M}_{\text{transition}}$, and $\mathscr{M}_{\text{decision}}$ respectively.

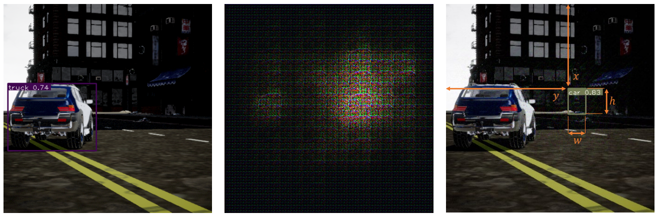
The step sizes for the various components of our multi-time scale optimization, namely $\varepsilon_n^{\text{img}}$, $\varepsilon_n^{\text{dec}}$, $\varepsilon_n^{\text{actor}}$, $\varepsilon_n^{\text{critic}}$, and $\varepsilon_n^{\text{sys}}$, are determined as follows. The generation of an adversarial image perturbation depends on the generator with parameter θ_n^{img}, the actor that determines the attack coordinates with parameter θ_n^{actor}, and the binary decision maker that chooses whether to use the adversarial perturbation or not with parameter θ_n^{dec}. As the generation of the adversarial image perturbation and its use are governed by a policy with parameters θ_n^{img}, θ_n^{actor}, and θ_n^{dec}, we set faster update rates for the parameters that are relevant to policy evaluation, i.e., θ_n^{critic} and θ_n^{sys}. Hence, the step size follows the diminishing rules as $n \to \infty$

$$\frac{\varepsilon_n^{\text{img}}}{\varepsilon_n^{\text{dec}}} \to 0 \quad \frac{\varepsilon_n^{\text{dec}}}{\varepsilon_n^{\text{actor}}} \to 0 \quad \frac{\varepsilon_n^{\text{actor}}}{\varepsilon_n^{\text{critic}}} \to 0 \quad \frac{\varepsilon_n^{\text{critic}}}{\varepsilon_n^{\text{sys}}} \to 0, \tag{4}$$

This is because we intend to set slower update rates for the lower-level components of the policy that generate data for the upper-level components of policy evaluation.

We describe the loss functions of the stochastic gradients for the multi-level stochastic optimization as follows:

1) Image attack generator: We utilize a *white box* model as a proxy object detector to train the attack generator. Specifically, we use the recently released version of *YOLO*, called *YOLOv5* [20], for this purpose.



(a) image (b) perturbation $\mathbf{w}_t$ (c) perturbed image

Fig. 5: Fabrication of the bounding box at (x, y, w, h) with $\mathbf{w}_t$

Our image attack generator fabricates bounding boxes at the target coordinates, injecting adversarial perturbations as shown in Figure 1b. Using reinforcement learning, the high-level attacker (reinforcement learning agent) selects the target coordinates to place the fabricated bounding boxes accordingly. Given the high-level attack $\mathbf{a}_t \in [0,1] \times [0,1] \times [0,1] \times [0,1]$ representing the coordinates x and y of a bounding box, its width and height, the image attack network aims to delete all other bounding boxes but keep the one corresponding to the high-level attack as illustrated in Figure 5. By performing optimization iterations (500 iterations in Figure 5), we can delete the existing bounding box and place a bounding box according to the target coordinates.

The iterative optimization approach that creates a bounding box for online image attacks, as shown in Figure 5, is not suitable as it must be performed within a fixed time step of the control loop in the autonomous system. In our framework, we instead train an attack generator that minimizes the loss function through the image attack generator:

$$\arg\min_{\theta^{\text{img}}} l^{img}\left(\mathbf{w}(\mathbf{x};\theta^{\text{img}}), \mathbf{x}, \mathbf{a}\right), \tag{5}$$

where $\mathbf{w}(\mathbf{x};\theta^{\text{img}})) := \mathbf{Dec}_1\left(\mathbf{Enc}_1(\mathbf{x};\theta^{\text{img}}), \mathbf{a};\theta^{\text{img}}\right)$, and $\mathbf{x}$ and $\mathbf{a}$ are sampled from $\mathscr{M}_{\text{trajectory}}$. This approach differs from the optimization over image space that is suitable for one-time use, i.e., $\arg\min_{\mathbf{w}} l^{img}(\mathbf{w}, \mathbf{x}, \mathbf{a})$. The stochastic gradient $S_n^{\text{img}}(\mathscr{M}_{\text{trajectory}})$ in (3) is associated with the loss function in (5). To fabricate the detected bounding box, i.e., inverted mapping from the fabricated detection to the input image, we use the loss function employed from [1], where the same loss function is used to train the YOLO detector. The specific loss function in (5) from [1] is described in the appendix of the extended version [28].

2) System identification for state estimation: Due to incomplete observations of the state $\mathbf{s}_t$ through image stream $\mathbf{x}_t$, we need to identify the system to construct the estimator. The system identification determines the parameter that maximizes the state estimate's likelihood. We maximize the likelihood of state predictor by minimizing the cross-entropy error between true image streams and the predicted image streams by a stochastic optimization which samples trajectories saved in the memory buffer denoted by $\mathscr{M}_{\text{trajectory}}$ with a loss function to minimize.

The loss function $l^{\text{sys}}(\cdot)$ is calculated using the sampled trajectories from $\mathscr{M}_{\text{trajectory}}$ We calculate the loss function as

$$l^{\text{sys}}(\mathscr{M}_{\text{trajectory}};\theta_{\text{sys}}) = \frac{1}{M}\sum_{m=1}^{M} H(\mathbf{X}_m, \hat{\mathbf{X}}_m) \tag{6}$$

where $\mathbf{X}_m = (\mathbf{x}_0, \ldots, \mathbf{x}_T)_m$ is the m^{th} sample image stream with time length T. Here, $H(\cdot,\cdot)$ is average of the binary cross-entropy $h(\cdot,\cdot)$ between the original image stream $\mathbf{X}_m$ and the predicted image stream $\hat{\mathbf{X}}_m$, which is computed over the RGB pixel values of the image streams.

We generate the predicted trajectory, $\hat{\mathbf{X}}_m = \{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_T\}$, given the original trajectory with image stream $\mathbf{X}_m = (\mathbf{x}_0, \ldots, \mathbf{x}_T)_m$ and action stream $(\mathbf{a}_0, \ldots, \mathbf{a}_T)_m$ by processing them through the encoder, GRU, and the decoder as

$$\begin{aligned} &\mathbf{h}_{t+1} = \text{GRU}(\mathbf{h}_t, \text{Encoder}_1(\mathbf{x}_t), \mathbf{a}_t), \quad \mathbf{h}_0 \sim \mathscr{N}(0, \mathbf{I}), \\ &\hat{\mathbf{x}}_{t+1} = \text{Decoder}_1(\mathbf{h}_{t+1}). \end{aligned}$$

With the loss function in (6), the stochastic gradient for the optimization is defined as $S_n^{\text{sys}} = -\nabla_{\theta_{\text{sys}}} l^{\text{sys}}(\mathscr{M}_{\text{trajectory}};\theta_{\text{sys}})$.

3) Actor-Critic policy improvement: The attack coordinate $\mathbf{a}_t$ is determined by the policy, i.e., $\mathbf{a}_t = \mu(\mathbf{h}_t;\theta_{\text{actor}})$ that maps the state estimate $\mathbf{h}_t$ into an action $\mathbf{a}_t$. To improve the policy, the critic evaluates the policy relying on the principle of optimality [29]. We employed an actor-critic method [30] for the reinforcement learning agent in the proposed framework. The critic network is updated using the state estimate $\mathbf{h}_t$ to apply the optimality principle with the following stochastic gradient as $S_n^{\text{critic}} = -\nabla_{\theta_{\text{critic}}} l^{\text{critic}}(\mathscr{M}_{\text{transition}};\theta_{\text{critic}})$ with the following loss function

$$l^{\text{critic}}(\mathscr{M}_{\text{transition}};\theta^{\text{critic}}) = \frac{1}{M}\sum_{m=1}^{M} (Q(\mathbf{h}_m, \mathbf{a}_m;\theta^{\text{critic}}) - Q_m^{\text{target}})^2,$$

where $Q_m^{\text{target}} = r_m + \gamma Q(\mathbf{h}'_m, \mu_\theta(\mathbf{h}'_m);\theta_{\text{critic}})$ and the state transition samples, i.e., $((\mathbf{h}, \mathbf{a}, \mathbf{h}', r)_0, \ldots, (\mathbf{h}, \mathbf{a}, \mathbf{h}', r)_M)$, are sampled from the replay buffer $\mathscr{M}_{\text{transition}}$.With the same state transition data samples, we calculate the stochastic gradient for the policy update as $S_n^{\text{actor}} = \nabla_{\theta_{\text{actor}}} J(\mathscr{M}_{\text{transition}}; \theta_{\text{actor}})$ with the following estimated value function as

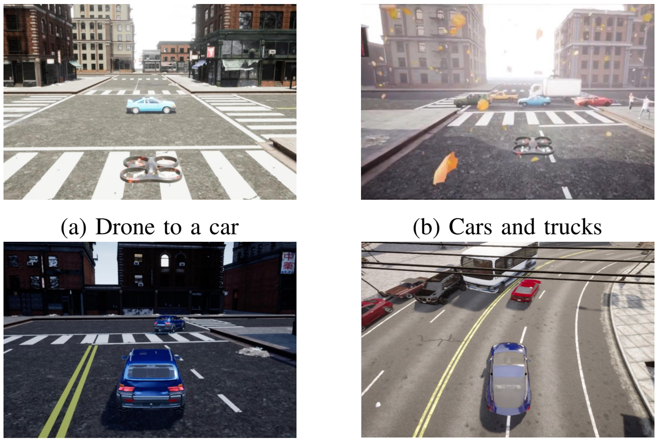
$$J = \frac{1}{M} \sum_{m=1}^{M} Q(\mathbf{h}_m, \mu(\mathbf{h}_m; \theta_{\text{actor}}); \theta_{\text{critic}}),$$

where $Q(\mathbf{h}, \mathbf{a}; \theta_{\text{critic}})$ indicates the value of taking action $\mathbf{a}$ at the state estimated as $\mathbf{h}$.

4) Loss estimators for the binary decision making: The stochastic gradient S_n^{dec} in (3) is associated with the two optimizations described in (1). The entire stochastic optimization with the aforementioned stochastic gradients is summarized as Algorithm 2 in the appendix of the extended version [28].

IV. Experiments

We tested the proposed attack method to determine its ability to mislead autonomous vehicles in line with adversarial objectives. The adversary relied solely on image frames as sensing input and an uncertain actuator, in the form of an adversarial perturbation, to manipulate the paths of autonomous vehicles. Despite the adversary's limited sensor and uncertain actuator, our proposed algorithm successfully misled the autonomous vehicles in various simulation environments shown in Figure 6 (and in illustrative videos[2]).



(a) Drone to a car

(b) Cars and trucks

(c) Following a car

(d) Following a car in traffic

Fig. 6: Simulation environments.

All our experiments consider attacking a vision-based guidance system depicted in Figure 2 that uses *YOLOv5* object detector [20]. To simulate the vehicle environment, we employed a game development editor (*Unreal* [31]) that is capable of building *photo-realistic* 3D environments, along with plug-in tools such as *AirSim* [32] and *CARLA* [33]. For the attack algorithm implementation, we used the robot operating system (*ROS*) to simultaneously implement the attack model learning and executing the attack using multiple modules (nodes) as illustrated in Figure 7. All experiments were conducted on a desktop computer equipped with a GPU capable of rendering the 3D environments and performing DNN training.

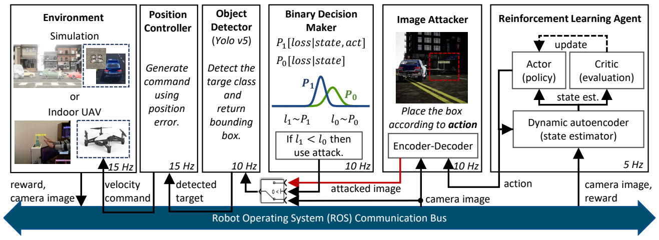


Fig. 7: The framework implemented using *ROS*.

A. Baseline method

Our proposed framework was compared to an image attack method presented in [12], [18], which uses iterative optimization with the image tensor as the decision variable. The effectiveness of these methods depends on the number of iterations and the scale factor α, which limits the size of the adversarial perturbation as $\tilde{\mathbf{x}}_t = \max(\min(\mathbf{x}_t + \alpha \mathbf{w}_t, 1), 0)$. Previous works [12], [18] developed such methods as offline approaches. When an infinite number of iterations are allowed, the offline method can arbitrarily fabricate the bounding box, as illustrated in Figure 5. To the best of our knowledge, no online image attack methods have been applied to autonomous vehicles. Therefore, we set a baseline method by applying the iterative optimization method as an online algorithm. In addition to the number of iterations, the scale factor α is a critical hyperparameter, as a higher value can increase the attacker's ability to fabricate bounding boxes, but it also reduces the stealthiness of the attack. For example, we collect the performance of the base line method with varying number of iteration and the scale factor as shown in Figure 8. For the first three experiments (the first three rows in Table II), we set the hyperparameter $\alpha = 0.8$ and 20 iterations for the baseline method (Iterative optimization). The optimization method generated image perturbations approximately every 1 second. However, for our proposed methods, including Generative Attack, Recursive Attack, Neural Thompson, and Conditional Sampling, we set $\alpha = 0.05$ for stealthiness of the image attack. Additionally, the baseline method required manual annotation of the bounding box to be fabricated, as described in [19]. Therefore, we manually placed the

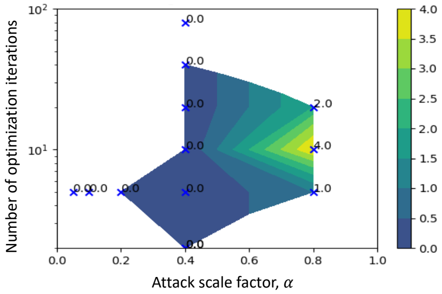


Fig. 8: Terminal rewards in the 3rd environment (Figure 6c) of the baseline method with different of α and the iterations.

[2] (a) *Drone to a car* at https://youtu.be/sjgQGgyLR8Y; (b) *Cars and trucks* at https://youtu.be/Xx9hH6mP0PE; (c) *Following a car* at https://youtu.be/mOPfPDEXkdM; (d) *Following a car in traffic* at https://youtu.be/z61FyoJx$_Y g$.

target area to fabricate the bounding box according to the adversarial objectives, such as placing the bounding box to the left when the adversary needs to move the vehicle to the left. In the last experiment, we set the scale factor of the baseline method equal to that of our proposed methods, i.e., $\alpha = 0.05$. For α values greater than 0.2, the baseline method was as effective as our proposed methods.

B. Ablation study

We evaluated the proposed framework by conducting ablations, as presented in Table I. To test the effectiveness of each method, we conducted experiments in four different environments illustrated in Figure 6. In the 1st environment (Figure 6a), we set the reward as the distance between the host vehicle and the target object. Thus, the adversary can increase the distance to move the vehicle away from the target. In the 2nd environment (Figure 6b), the reward is set as the horizontal coordinate of the host vehicle with respect to the target object. In this scenario, learning to increase the horizontal coordinate would lead to a crash. In the 3rd environment (Figure 6c), the adversary learns to increase the distance from the front car. In the fourth environment, we rewarded the learning agents when the distance between the host vehicle and the front car was greater than 50 meters. As shown in Table II, the four environments have different object tracking methods. The first two environments use only **YOLO** detection. In the 3rd environment, a Kalman filter is used to filter out the changes of the bounding boxes that is the outcome of the detecion. In the last environment, an multi-object tracking (SORT [34]) is implemented to deal with multiple cars in the traffic. We report the performance of the trained attackers (listed in Table I) with the last ten episodes of the entire 200 training episodes in Table II.

Methods	Generative network (Y/N)	State estimator (Y/N)	Attack switch (Y/N)
Iterative optimization	N	N	N
Generative attack	Y	N	N
Recursive attack	Y	Y	N
Neural Thompson sampling	Y	Y	Y
Conditional sampling	Y	Y	Y

TABLE I: List of components for ablation study.

In comparison to the baseline iterative optimization method, the recursive attack methods demonstrated higher terminal rewards, collision rates, and terminal distances. The incorporation of the state estimator improved the overall performance in terms of terminal rewards. Since we desire infrequent use of the image attack for stealthiness, we measure how frequently the attacks are used, i.e., attack rate. The utilization of binary decision-makers such as Neural Thompson Sampling (NTS) or the proposed conditional sampling resulted in decreased attack rates and the difference between the attacked images and the original images in terms of L2 norm and SSIM loss. Moreover, our proposed conditional sampling method showed higher terminal rewards compared to NTS.

Moreover, the conditional sampling shows the higher use of the attack when the image attack loss is lower as we intended as shown in Figure 9. In contrast, the Thompson sampling (NTS) shows the opposite correlation, i.e., using the image attack when the loss values are higher. Further information regarding the simulations can be found in the appendix of the extended version [28].

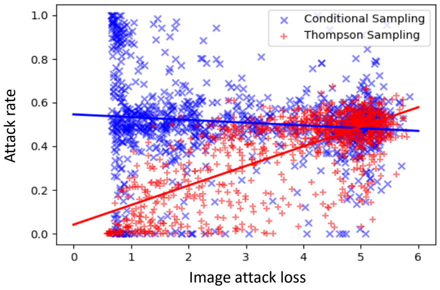


Fig. 9: Attack rate vs. Image attack loss of the 5 training experiments with the 2nd environment (Figure 6b).

C. Real robot experiment

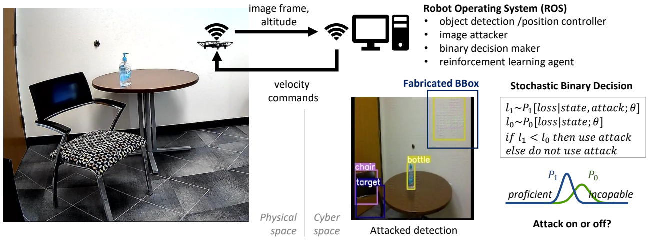


image frame, altitude
velocity commands
Physical space
Cyber space
Robot Operating System (ROS)
object detection /position controller
image attacker
binary decision maker
reinforcement learning agent
Fabricated BBox
chair
target
bottle
Attacked detection
Stochastic Binary Decision
$l_1 \sim P_1[loss|state, attack; \theta]$
$l_0 \sim P_0[loss|state; \theta]$
$if\ l_1 < l_0\ then\ use\ attack$
$else\ do\ not\ use\ attack$
P_1
P_0
proficient
incapable
Attack on or off?

Fig. 10: Attacking UAV's visual-tracking of a bottle.

We implemented the proposed framework with a miniature drone (*DJI Tello*) to validate its efficacy in real-world scenarios, as depicted in Figure 7. The drone employs IMU, optical-flow, and barometer for velocity estimation and to follow velocity commands. The drone was connected to a desktop computer via wifi-networks, as shown in Figure 10. The objective of the online image training was to crash the UAV by teaching it to fabricate the bounding box. In the linked video[3], the online training lasted for ten minutes, and the UAV crashed successfully.

V. Conclusion

This work showed a new online image attack framework that improves the iterative optimization-based methods that are more suitable for offline attack generation. In our proposed framework, the image attacks can be generated in real-time using only the image stream collected from the autonomous vehicle. Furthermore, the proposed conditional sampling for the binary decision making whether to use the attack (or not) improves the stealthiness by waiting until the proficiency increases. This work will serve as a stepping stone towards strengthening the perception in autonomous vehicles by learning worst-case attack scenarios.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016, pp. 779–788.

[2] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *CVPR*, 2017, pp. 652–660.

[3] Illustrative video available at https://youtu.be/4w0pvQRCVHc

Environments	Object Tracking	Methods	Attack	SSIM	L2	Collision	Terminal	Time AVG
			rate	loss	loss	rate	reward	reward
			(%)	(10^{-2}, avg)	(10^{-4}, avg)	(%)	(avg±stdev)	(avg±stdev)
Drone to a car (Figure 6a)	YOLO detection only	Normal	-	-	-	0	15.7 ± 0.0	1.29 ± 0.07
		Iterative optimization	100	13.6	12.1	0	15.7 ± 0.1	1.29 ± 0.07
		Generative attack	100	8.74	3.94	74	$\mathbf{30.1 \pm 8.3}$	$\mathbf{2.58 \pm 0.80}$
		Recursive attack	100	17.4	7.61	**76**	29.7 ± 8.1	2.16 ± 0.45
		Neural Thompson	**31.7**	**6.81**	**2.58**	0	21.5 ± 10.5	1.64 ± 0.49
		Conditional sampling	55.3	9.15	4.11	52	27.8 ± 9.2	2.16 ± 0.57
Cars and trucks (Figure 6b)	YOLO detection only	Normal	-	-	-	0	1.8 ± 1.8	0.10 ± 0.15
		Iterative optimization	100	16.7	30.3	0	-0.4 ± 2.3	-0.03 ± 0.13
		Generative attack	100	7.52	2.68	18	5.4 ± 13.3	0.29 ± 0.75
		Recursive attack	100	7.26	2.74	**36**	$\mathbf{9.4 \pm 6.3}$	$\mathbf{0.87 \pm 0.43}$
		Neural Thompson	49.3	4.94	1.77	8	6.0 ± 9.2	0.25 ± 0.46
		Conditional sampling	**42.7**	**1.97**	**0.81**	20	2.6 ± 10.6	0.10 ± 0.53
Following a car (Figure 6c)	YOLO detection and Kalman filter	Normal	-	-	-	0	0 ± 0	4.25 ± 0.04
		Iterative optimization	100	9.17	7.58	30	3.0 ± 4.6	4.17 ± 0.23
		Generative attack	100	5.96	3.16	66	7.0 ± 4.4	2.92 ± 1.10
		Recursive attack	100	4.88	2.61	74	7.4 ± 4.4	2.92 ± 1.33
		Neural Thompson	**33.5**	**1.66**	**0.96**	52	5.2 ± 5.0	4.01 ± 0.39
		Conditional sampling	72.8	3.10	2.06	**76**	$\mathbf{7.6 \pm 4.3}$	$\mathbf{4.27 \pm 3.54}$
Following a car in traffic (Figure 6d)	YOLO detection and multi-object tracking (SORT)	Normal	-	-	-	6	3.7 ± 4.2	$\mathbf{2.28} \pm 0.30$
		Iterative optimization	100	0.0	0.0	4	3.2 ± 3.5	2.24 ± 0.34
		Generative attack	100	20.1	9.1	**30**	$\mathbf{10.4} \pm 3.7$	0.97 ± 0.72
		Recursive attack	100	23.5	10.6	2	9.9 ± 2.2	0.63 ± 0.60
		Neural Thompson	**34.6**	**8.1**	**3.7**	10	8.1 ± 5.3	1.39 ± 0.83
		Conditional Sampling	49.6	12.4	5.56	10	9.3 ± 4.8	1.09 ± 0.79

TABLE II: Ablation study with the last 10 episodes in 5 random training experiments, i.e., $N = 50$. The stealthiness is evaluated by Attack rate, SSIM loos, and L2 loss. And the attacker's performance to disrupt is evaluated by Collision rate, Terminal reward, and Time averaged reward.

[3] (2016) DJI mavic active tracking. Accessed: 2021-12-01. [Online]. Available: https://youtu.be/ss0J0dAI1DM

[4] (2022) What's next. [Online]. Available: https://waymo.com/intl/en_us/dataset-whats-next/

[5] T. Başar and P. Bernhard, *H-infinity optimal control and related minimax design problems: a dynamic game approach*. Springer Science & Business Media, 2008.

[6] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *ICML*. PMLR, 2017, pp. 2817–2826.

[7] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *5th ICLR*, 2017.

[8] ——, "Adversarial examples in the physical world," in *5th ICLR*, 2017.

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[10] E. Ackerman, "Three small stickers in intersection can cause tesla autopilot to swerve into wrong lane," *IEEE Spectrum, April*, 2019.

[11] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *CVPR*, 2018, pp. 1625–1634.

[12] Y. Jia, Y. Lu, J. Shen, Q. A. Chen, Z. Zhong, and T. Wei, "Fooling detection alone is not enough: First adversarial attack against multiple object tracking," in *ICLR*, 2020.

[13] "Tesla crash driver posted videos of himself riding without hands on wheel," *The Guardian*, Mar 15, 2021. [Online]. Available: https://www.theguardian.com/us-news/2021/may/15/tesla-fatal-california-crash-autopilot

[14] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," in *ICML*, 2018, pp. 2137–2146.

[15] Z. Wei, J. Chen, X. Wei, L. Jiang, T.-S. Chua, F. Zhou, and Y.-G. Jiang, "Heuristic black-box adversarial attacks on video recognition models." in *AAAI*, 2020, pp. 12 338–12 345.

[16] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, "Generating adversarial examples with adversarial networks," in *IJCAI*, 2018, pp. 3905–3911.

[17] A. Boloor, K. Garimella, X. He, C. Gill, Y. Vorobeychik, and X. Zhang, "Attacking vision-based perception in end-to-end autonomous driving models," *Journal of Systems Architecture*, vol. 110, p. 101766, 2020.

[18] S. Jha, S. Cui, S. Banerjee, J. Cyriac, T. Tsai, Z. Kalbarczyk, and R. K. Iyer, "Ml-driven malware that targets av safety," in *International Conference on Dependable Systems and Networks*, 2020, pp. 113–124.

[19] (2021) Fooling Detection Alone is Not Enough: Adversarial Attack against Multiple Object Tracking. Accessed: 2021-12-01. [Online]. Available: https://openreview.net/forum?id=rJl31TNYPr

[20] (2021) YOLOv5 — Pytorch. Accessed: 2021-12-01. [Online]. Available: https://pytorch.org/hub/ultralytics_yolov5

[21] Y. Wang, K. Wang, Z. Zhu, and F.-Y. Wang, "Adversarial attacks on faster r-cnn object detector," *Neurocomputing*, vol. 382, pp. 87–95, 2020.

[22] J. Vermorel and M. Mohri, "Multi-armed bandit algorithms and empirical evaluation," in *ECML*. Springer, 2005, pp. 437–448.

[23] W. Zhang, D. Zhou, L. Li, and Q. Gu, "Neural thompson sampling," *arXiv preprint arXiv:2010.00827*, 2020.

[24] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *ICML*. PMLR, 2016, pp. 1050–1059.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[26] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *NeurIPS*, 2000, pp. 1008–1014.

[27] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *NeurIPS*, 2017, pp. 6626–6637.

[28] H.-J. Yoon, H. Jafarnejadsani, and P. Voulgaris, "Learning when to use adaptive adversarial image perturbations against autonomous vehicles," *arXiv preprint arXiv:2212.13667*, 2022.

[29] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR*, 2016.

[31] (2021) Unreal Engine. Accessed: 2021-12-01. [Online]. Available: https://www.unrealengine.com/

[32] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[33] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[34] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *ICIP*. IEEE, 2016, pp. 3464–3468.