Adversarial Machine Learning and Defenses for Automated and Connected Vehicles

by

Dayu Zhang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2024

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Dayu Zhang was the sole author for Chapters 1, 2, 3 and 5 which were written under the supervision of Dr. Nasser L. Azad and were not written for publication. This thesis consists in part of two manuscripts written for publication. Exceptions to sole authorship of material are as follows:

Research Presented in Chapter 4 Dayu Zhang was the primary author of the manuscripts under the supervision Dr. Nasser L. Azad with the funding provided by AVL List GmbH. Dr. Sebastian Fischmeister was the the AVL project lead, as well as the co-author by providing guidance and feedback on the manuscript. Stefan Marksteiner provided insights from AVL List GmbH and contributed to the manuscript through feedbacks. Papers presented in Chapter 4 was submitted to the 2023 International Conference on Informatics in Control, Automation and Robotics (ICINCO) conference. Paper presented in chapter 4 has been published.

Citations:

- Chapter 4: Zhang D, Azad N, Fischmeister S, Marksteiner S. Zeroth-Order Optimization Attacks on Deep Reinforcement Learning-Based Lane Changing Algorithms for Autonomous Vehicles. Proceedings of the 20th International Conference on Informatics in Control, Automation and Robotics - Volume 1: ICINCO. 2023:665-673. doi:10.5220/0012187700003543.

Abstract

This thesis delves into the realm of adversarial machine learning within the context of Connected and Automated Vehicles (CAVs), presenting a comprehensive study on the vulnerabilities and defense mechanisms against adversarial attacks in two critical areas: object detection and decision-making systems.

The research firstly introduces a novel adversarial patch generation technique targeting the YOLOv5 object detection algorithm. It presents a comprehensive study in the different transformations and parameters that change the effectiveness of the patch. The patch is then implemented within the CARLA simulation environment to assess robustness under varied real-world conditions, such as changing weather and lighting. With all the transformation applied during generation, the patch is able to reduce the confidence of YOLO5 detecting the stop sign by 70% comparing to the original stop sign if the lighting condition is good. However if the lighting condition is sub-optimal, for example, during a raining weather, the patch only reduce the confidence by 38% due to the patch being harder to be detected. Overall, the optimized patch still shows a greater effect on detection evasion compares to a random noise patch on any environment conditions. Overall, this part of the research showcase a novel way of generating adversarial patches and a new approach of testing the patches in a open-source simulator, CARLA, for better autonomous vehicle testing against adversarial attacks in the future.

Simultaneously, this thesis investigates the susceptibility of Deep Reinforcement Learning (DRL) algorithms, in particular, Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG) algorithms, to black-box adversarial attacks executed through zeroth-order optimization methods like ZO-SignSGD in a lane-changing scenario. The research first train the policies with finely turned hyper-parameters in the lane-changing environment and achieving a high performance. With a good policy as a base, the black-box attack successfully fooled both algorithms by optimally changing the state value to force the policy going straight while maintaining a small perturbation size compare to the original. While under attack, both DQN and DDPG are unable to perform, achieving an average of reward 108 and 45 comparing to their original performance of 310 and 232 respectively. A preliminary study on the effect of adversarial defense is also performed, which shows resistance against the attack and achieving slightly increase in average reward. This part of research uncovers significant vulnerabilities, demonstrating substantial performance degradation in DRL when used in the decision making of an autonomous vehicle.

At last, the study underscores the importance of enhancing the security and resilience of machine learning algorithms embedded in CAV systems. Through a dual-focus on

offensive and defensive strategies, including the exploration of adversarial training, this work contributes to the foundational understanding of adversarial threats in autonomous driving and advocates for the integration of robust defense mechanisms to ensure the safety and reliability of future autonomous transportation systems.

Acknowledgements

I would like to thank Dr. Nasser L. Azad for the guidence and help throughout the entire research process.

I would also like to thank AVL List GmbH. and Natural Sciences and Engineering Research Council of Canada (NSERC) for funding this project; Dr. Sebastian Fischmeister for providing the knowledge and resources for this project; Stefan Marksteiner from AVL List GmbH for providing support throughout the project.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The rapid advancement in Connected and Automated Vehicles (CAVs) underscores the critical need for robust and secure decision-making algorithms. This necessity is increasingly evident with the deep integration of machine learning, particularly deep neural networks (DNN), in various aspects of CAV technology. The groundbreaking work of Krizhevsky et al. with AlexNet in 2012 marked a significant leap in image classification, catalyzing the adoption of deep learning in machine vision applications [27]. Following this, notable developments such as R-CNN by Girshick et al. in 2013 and the real-time object detection system YOLO in 2015 further revolutionized the field [17, 47]. Around the same time, Mnih et al. at Deepminds firstly combined deep learning techniques with reinforcement learning (RL), enabling complex control policies to be learnt and generalized. These advancements have been pivotal in enhancing the capabilities of CAVs but have also exposed them to new vulnerabilities, particularly to adversarial attacks.

1.2 Background

The concept of adversarial attacks in deep learning, first identified by Szegedy et al. in 2013, introduced a critical area of concern [52]. These attacks involve subtle yet impactful perturbations to inputs that lead machine learning model to make incorrect decisions, such as the example seen in figure 1.1. The imperceptibility of these perturbations to both human observers and standard detection techniques poses a significant challenge. This

field of research soon became a topic of significant interest, as deep learning techniques getting increasingly integrated into real-world applications, from image classification to autonomous driving. In 2015, Goodfellow et al. further explored this concept, generalized and showcased the degree of susceptibility of deep neural networks (DNNs) to adversarial attacks by generating an adversarial example in merely one single step using Fast Gradient Sign Method (FGSM), as well as defending against such attacks through adversarial training [18]. This work was followed by the introduction of the Carlini-Wagner (CW) attack in 2017, which crafted a more complex optimization method to generate adversarial examples with minimal perturbations, further highlighting the vulnerability of DNNs to adversarial attacks [7]. In 2018, Madry et al. introduced a powerful multi-step attack named Projected Gradient Descent (PGD) that provides a more reliable way to find adversarial examples for all DNNs. Furthermore, it demonstrated the capabilities of combining adversarial training with a powerful attack like PGD to enhance the robustness of DNNs not only against PGD itself, but other weaker attacks as well [35]. These developments showcased the potential of these threats to compromise the safety of autonomous systems. However, they require the attacker to have access to the gradient information of the target model, namely "white-box attacks", which is not always the case in real-world scenarios.

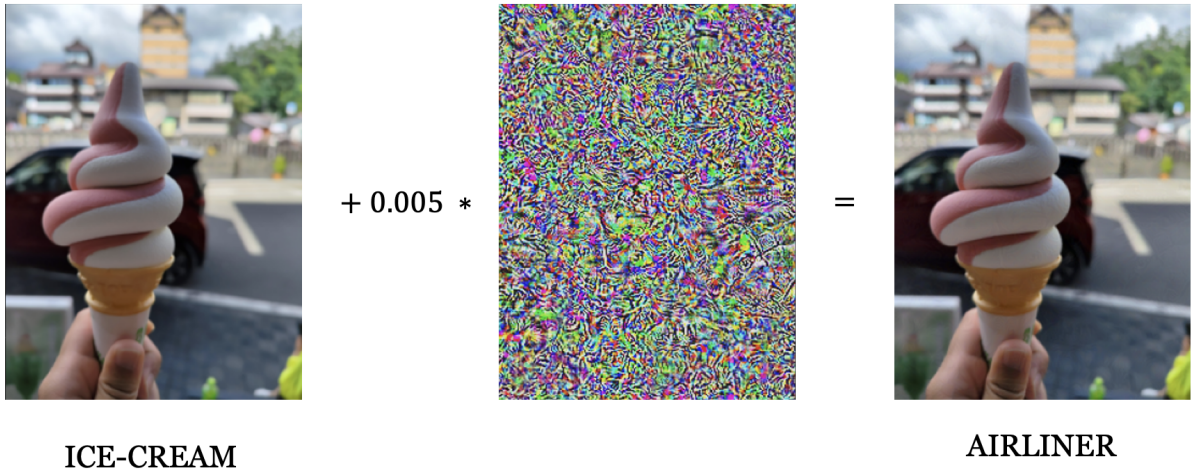


Figure 1.1: An example adversarial attack on the ImageNet classifier [27] to output airliner instead of ice-cream. The perturbed image is indifferent to naked eye compare to the original image.

Not only presented in the field of machine vision, adversarial attacks have also been explored in the context of reinforcement learning (RL) and control. Huang et al. in 2017 demonstrated the susceptibility of RL-based control algorithms to adversarial attacks,

highlighting the potential of these attacks to compromise the safety of autonomous systems [21]. This vulnerability is particularly concerning in the context of AVs, where the safety of the vehicle and its occupants is dependent on the robustness of the decision-making algorithms.

All these vulnerabilities were initially only explored in the digital realm, but soon found practical implications in the physical world. For instance, Evtimov et al. in 2017 demonstrated the feasibility of tricking DNN-based image classifiers with specially crafted stickers on stop signs, a direct threat to the safety of AV systems [13]. Moreover, Athalye et al.'s introduction of "Expectation Over Transformation" in 2017, showcased the robustness of physical adversarial attacks, highlighting the efficacy and feasibility of these threats in the physical world [1]. These papers laid the fundamental groundwork for the exploration of robust adversarial attacks for object detection systems, which are widely used in CAVs. In 2019, adversarial patch were developed to attack object detection systems, which can be printed on a piece of paper and placed in the physical world to fool the object detection system, further demonstrating the feasibility of physical adversarial attacks [54, 28, 10].

At last while white-box attacks are more powerful, they are also more challenging to execute. In contrast, "black-box attacks" are more practical and easier to execute, as they do not require explicit knowledge of the target model's inner workings. In 2016, Papernot et al. introduced the concept of "transferability" in adversarial attacks, where one attack trained specially for one network can also be effective on another network doing the same task; demonstrating the feasibility of black-box attacks [39, 40]. Though still black-box, these early attacks use substitute models to generate adversarial examples, which are attacked by any of the white-box attacks mentioned above to generate adversarial examples for the unknown neural networks. In the 2017 work of Chen et al., the concept of "zeroth-order optimization" was introduced, which query the neural network directly, enables the execution of black-box attacks without the need for substitute models [9]. This direction was further explored by Liu et al. in 2018, who demonstrated more efficient zeroth-order optimization methods like ZO-SignSGD for executing black-box attacks [31]. In the case of zeroth-order optimization, though the gradient information is not present, the attacker still has access to the confidence or the score of the model output, thus allowing the attacker to know where the optimization process is heading. Another direction of the black-box attack is to use the "decision-based" method, which only has access to the final decision of the model, while both the gradient information and confidence are not available. In 2018, Chen et al. introduced the Boundary Attack, where it starts with an adversarial example, and then optimize it back into the original image as much as possible [3]. This attack is particularly powerful as it can be executed in the physical world, where the attacker can only observe the final decision of the model. All these black-box attacks are particularly

concerning in the context of CAVs, as they have much more realistic impact on the safety of the vehicle and its occupants.

1.3 Thesis Structure

This thesis is divided in to two parts. Chapter 3 focuses on the efficacy of adversarial attacks on state-of-the-art object detection systems in CAVs in simulated environments. Chapter 4 devlves into the vulnerability of DRL-based decision-making control algorithms in CAVs.

In the realm of image processing and simulation, chapter 3 researches the application of adversarial examples targeting object detection algorithms such as YOLOv5. The research proposes a methodology for generating adversarial attacks tailored to current object detection systems, considering the complex environments encountered by autonomous vehicles. The research employs the Carla simulator for validation of perception modules in CAVs, simulating a wide range of driving conditions, including different weather conditions, lighting variations, and camera movements. A key aspect of this methodology is the testing of various configurations for generating and applying adversarial patches within the Carla environment. This approach ensures that the adversarial examples are effective and robust across different scenarios, enhancing the resilience of CAV perception systems against adversarial threats [12, 53, 33].

In the realm of control, chapter 4 concentrates on the susceptibility of highway lane-changing algorithms, a fundamental component of AV systems, to black-box adversarial attacks. These attacks are particularly concerning as they do not require explicit knowledge of the system's inner workings, such as gradient information, to be effective. The investigation includes widely implemented DRL policies like Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG), employing zeroth-order optimization methods like ZO-SignSGD for executing these attacks [36, 30, 31]. The experiments demonstrate the potential of these methods to compromise decision-making processes in AVs, thereby underscoring a critical security gap in current systems. Furthermore, the study explores the development of adversarial training techniques as a defensive strategy. Adversarial training, in the context of DRL, presents an opportunity to enhance the robustness of AV systems against such attacks. This aspect of the research contributes to a deeper understanding of the dynamics between adversarial attacks and defenses, emphasizing the importance of incorporating security measures in the design and deployment of these technologies.

In conclusion, this thesis explores the vulnerabilities of CAVs to adversarial attacks in the context of both control and perception. It not only highlights the existing gaps in

security but also proposes methods to identified and mitigate these risks in a simulated environment. By exploring and addressing these challenges, this thesis hopes to contribute to the development of safer and more reliable autonomous driving technologies, ensuring their secure integration into transportation systems.

Chapter 2

Background and Related Works

2.1 Adversarial Examples in Machine Learning

In 2014, Christian Szegedy and his team's groundbreaking paper, "Intriguing properties of neural networks," unveiled two key findings that have had a profound impact on the field of deep learning [52]. Firstly, the paper demonstrated that neural networks are susceptible to adversarial examples—inputs that are almost indistinguishable from natural data but are crafted to cause the network to make errors. These act of crafting such examples are later called adversarial attacks. These adversarial inputs exploit the model's learned features in such a way that even a tiny, carefully constructed perturbation can lead to incorrect outputs with high confidence. Szegedy and his colleagues employed a box-constrained L-BFGS method to generate these adversarial examples, highlighting the ease with which they could be created. The implications of this finding are significant, particularly for the field of computer vision and related applications like autonomous driving, where ensuring the reliability and security of neural network decisions is paramount. The paper not only opened up a new avenue of research into adversarial machine learning but also raised important questions about the fundamental nature of deep learning models.

Building upon Szegedy's work, In 2015,The paper "Explaining and Harnessing Adversarial Examples" by Ian Goodfellow et al. is a seminal work that delves futher into the adversarial examples in neural networks [18]. This research provides a more intuitive explanation for the existence of adversarial examples, attributing them primarily to the linear behavior in high-dimensional spaces of deep neural networks, rather than the non-linearities or overfitting, as previously thought.

To further demonstrate the ease of crafting such example, Goodfellow and his co-authors introduce the concept of the Fast Gradient Sign Method (FGSM), a simple yet effective technique for generating adversarial examples. This method involves taking the gradient of the loss with respect to the input data, then adjusting the input data by a small step in the direction of the gradient's sign. This approach is grounded in the paper's hypothesis that the linear nature of models in high-dimensional spaces makes them vulnerable to such perturbations.

A crucial insight from the paper is that adversarial examples can be generated even with models that have high precision on their training and test data, indicating that these perturbations are a fundamental characteristic of the models rather than a result of overfitting. The paper also explores the intriguing property that adversarial examples generated for one model are often effective on other models, even when they have different architectures or were trained on different subsets of the data, hinting at a kind of universal vulnerability among neural networks.

Furthermore, Goodfellow et al. propose that adversarial training, where models are trained on a mixture of adversarial and clean examples, can serve as a regularization method to improve the robustness of neural networks. This suggestion has significant implications for enhancing the security and reliability of machine learning systems across various applications, including those in critical areas like autonomous vehicles.

Data: x - Input image, y - True label for x, ϵ - Magnitude of the perturbation
Result: x_{adv} - Adversarial image

Procedure FGSM(x)
begin
  Initialize the model to be attacked;
  Compute the loss: $L = \text{Loss}(\text{model}(x), y)$;
  Calculate the gradients of the loss w.r.t. the input image: $\nabla_x L = \frac{\partial L}{\partial x}$;
  Generate the perturbation: $\text{perturbation} = \epsilon \cdot \text{sign}(\nabla_x L)$;
  Create the adversarial example: $x_{adv} = x + \text{perturbation}$;
  Clip x_{adv} to ensure it remains a valid image:
  $x_{adv} = \text{clip}(x_{adv}, \text{min\_value}, \text{max\_value})$;
  return x_{adv}
end

Algorithm 1: Fast Gradient Sign Method (FGSM)

The 2018 paper "Towards Deep Learning Models Resistant to Adversarial Attacks" by Aleksander Madry et al. represents an ongoing effort to enhance the robustness of neural networks against adversarial attacks, building upon the foundational work by Szegedy et

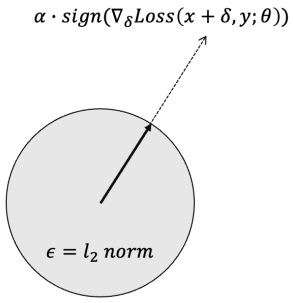


Figure 2.1: Graphical Interpretation of FGSM.

al. in 2014 and Goodfellow et al. in 2015 [35]. Drawing from the earlier discovery by Szegedy et al. that neural networks are prone to errors induced by nearly imperceptible perturbations, Madry and his team advance the understanding of adversarial examples by framing the problem within a robust optimization context.

Central to Madry et al.'s contribution is the formulation of a min-max optimization problem, which seeks to minimize the worst-case loss over all possible adversarial perturbations within a specified bound. This formulation captures the adversarial setting more effectively by directly incorporating the adversarial objective into the training process, thereby leading to models that inherently account for potential adversarial attacks.

The paper also emphasizes the use of Projected Gradient Descent (PGD) as a powerful tool for generating adversarial examples within this robust optimization framework. PGD represents an evolution of the adversarial example generation techniques previously introduced, offering a more systematic and iterative approach to exploring the space of potential adversarial perturbations. By integrating adversarial training—training on a mix of adversarial and clean examples—into their robust optimization framework, the authors demonstrate significant improvements in model resilience against adversarial attacks.

Data: x - Input image, y - True label for x, ϵ - Magnitude of the perturbation, α - Step size, N - Number of iterations

Result: x_{adv} - Adversarial image

Procedure PGD(x)

begin

$x_{adv}^{(0)} = x$;

for $i = 0$ **to** $N - 1$ **do**

Compute the loss: $L = \text{Loss}(\text{model}(x_{adv}^{(i)}), y)$;

Calculate the gradients: $\nabla_{x_{adv}} L = \frac{\partial L}{\partial x_{adv}^{(i)}}$;

Generate the perturbation: $\Delta x = \alpha \cdot \text{sign}(\nabla_{x_{adv}} L)$;

Create the adversarial example: $x_{adv}^{(i+1)} = x_{adv}^{(i)} + \Delta x$;

$x_{adv}^{(i+1)} = \text{clip}(x_{adv}^{(i+1)}, x - \epsilon, x + \epsilon)$;

$x_{adv}^{(i+1)} = \text{clip}(x_{adv}^{(i+1)}, \text{min\_value}, \text{max\_value})$;

end

return $x_{adv}^{(N)}$

end

Algorithm 2: Projected Gradient Descent (PGD)

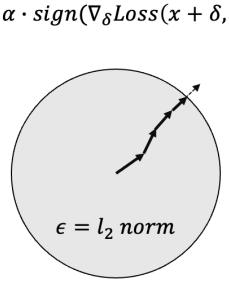


Figure 2.2: Graphical Interpretation of PGD.

2.2 Adversarial Attacks in Physical World

The initial discovery of adversarial examples found that the imperceptible perturbation often constructed using the projected gradient method (PGD) or the fast gradient sign method (FGSM) was able to fool image classification without fail [52][18]. However, such

attacks were found to be useless in the real world due to changes in camera angles and distances [34]. To further investigate the robustness of deep learning models against physical-world attacks, Ivan Evtimov and his team employed a sophisticated method in creating adversarial examples that could mislead a model when placed in the real world, particularly focusing on scenarios applicable to autonomous vehicles, such as manipulating traffic sign recognition systems [13].

The core of their methodology involved designing and printing perturbations that could be applied to real-world objects (like stop signs) in the form of stickers or overlays. These perturbations were crafted using an iterative optimization process that aimed to maximize the model's misclassification rate while ensuring that the modifications remained inconspicuous to human observers. This process was guided by an objective function that considered not only the model's confidence in incorrect classifications but also the physical realizability of the perturbations under different conditions.

To ensure the effectiveness of these adversarial examples under various environmental conditions, Evtimov et al. tested their approach in diverse settings, including different angles, distances, and lighting conditions. This comprehensive testing was crucial for demonstrating that the adversarial examples could consistently fool the deep learning models in real-world scenarios, not just in carefully controlled laboratory conditions. Thus, the adversarial example officially enters the physical environment, posing a threat to CAVs.

To create an even more robust attack, Athalye and his collaborators introduced an optimization technique known as Expectation Over Transformation (EOT), which is central to their methodology [1]. EOT allows for the generation of adversarial examples that are robust to a predefined set of transformations, thereby ensuring their effectiveness in a real-world context where input data to machine learning models often undergo various changes. The adversarial examples generated using EOT are designed to mislead the models not just in a fixed state but across a range of conditions, making the attacks more practical and challenging to defend against. This approach contrasts with traditional methods that typically focus on static adversarial examples, which might lose their adversarial properties when even slight alterations are applied. By considering the transformations during the adversarial example generation process, Athalye's method effectively accounts for the dynamic nature of real-world inputs.

This became the fundamental idea behind most future adversarial patches. Most notably, the ShapeShifter attack on Faster R-CNN proposed by Shang-Tse Chen et al [10]. Compare to image classification deep learning models, Faster R-CNN is particularly difficult as it involves multiple stages, including region proposal, feature extraction, and classification [49]. Each stage must be successfully navigated by the adversarial attack,

making it more complex than attacking a straightforward classification network. Due to the nature of region proposals and the subsequent refinement processes, R-CNN models are inherently robust to some variations in object appearance and position. An effective attack must therefore generate perturbations that can survive this robustness. At the same time, since Faster R-CNN focuses on localized regions within an image, any adversarial attack must be carefully crafted to impact the specific regions used for object detection, rather than the entire image. To tackle this problem, Chen's methodology incorporates the EOT framework, which allows it to account for various physical transformations (like changes in angle, distance, and lighting) that an image might undergo before being processed by the detector, ensuring the adversarial perturbations are effective even under different real-world conditions. Compare to previous deep learning attacks, instead of perturbing the entire image or object, Chen's approach often uses an adversarial patch that can be placed on the object. Making the attack easily realizable in the physical world.

Another popular object detection algorithm YOLO can also be attacked, as seen by the patches made by Simen Thys et al. to evade human detection [54] In this work, Thys and his colleagues attacked YOLOv2, also known as YOLO9000, which is an improved version of the original YOLO (You Only Look Once) object detection system, designed to be faster and more accurate [47, 48].

YOLO uses a single convolutional neural network (CNN) to predict multiple bounding boxes and class probabilities for those boxes simultaneously[47]. The image is firstly divided into a grid, and each grid cell is responsible for predicting bounding boxes and probabilities for objects whose center falls within the cell. Each bounding box prediction includes coordinates, size, and a confidence score, along with class probabilities. For each grid cell, YOLO predicts multiple bounding boxes and class probabilities. The confidence score reflects the accuracy of the bounding box and whether the box contains a specific class of object. Compare to the original YOLO, YOLOv2 incorporates batch normalization in all convolutional layers, which helps with model convergence and reduces the need for other forms of regularization [48]. It also introduces anchor boxes (priors) to predict bounding boxes, which helps the model learn to predict more accurate box sizes and aspect ratios, compared to the grid-based approach in the original YOLO.

Thys and his team developed a patch that contains specially designed patterns that disrupt the YOLOv2 detection algorithm [54]. When the patch is within the field of view of the camera, the patterns interfere with the model's ability to correctly interpret the presence of a person. The adversarial patterns are optimized to exploit the weaknesses in how YOLOv2 processes visual information. Since YOLOv2 makes its predictions based on the entire image in one pass, an effectively designed adversarial patch can create significant confusion for the model, leading to detection failures. By utilizing EOT, The patches are

designed to be effective in real-world conditions, meaning they can still fool the YOLOv2 model under different angles, distances, and lighting conditions. Recent developments include making more natural patches and applications in fields like evading drone camera object detections [20] [58].

2.3 Adversarial Attacks in Reinforcement Learning

Reinforcement Learning (RL) is a machine learning algorithm where an agent learns to interact with an environment to maximize the reward. Given a state, the agent produces an action, and based on this action, the environment provides a corresponding reward. The agent then updates its policy based on the reward. The agent is trained by interacting with the environment for several episodes. A classic example of RL is Q-learning invented in 1989 [56]. Q-learning is a model-free reinforcement learning algorithm that seeks to learn the quality of actions, denoting how good an action is in a given state, without requiring a model of the environment. It operates on the principle of a Markov Decision Process (MDP), which provides a mathematical framework for modeling decision-making situations where outcomes are partly random and partly under the control of a decision-maker. An MDP is characterized by a set of states, a set of available actions in each state, transition probabilities that define the likelihood of moving from one state to another after taking an action, and rewards that quantify the immediate gain of taking an action in a state. In Q-learning, the agent uses the Q-function to estimate the expected utility of taking a given action in a given state and following a certain policy thereafter. The Q-function is iteratively updated using the Bellman equation, written as the equation **??**, which incorporates the immediate reward and the discounted future rewards. Over time, this process converges to the optimal Q-function, which prescribes the best action to take in each state to maximize the cumulative reward.

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a') \tag{2.1}$$

where:

- $Q(s,a)$ is the value of taking action a in state s
- $r(s,a)$ is the immediate reward received after taking action a in state s.
- γ is the discount factor, a value between 0 and 1 that represents the difference in importance between future rewards and present rewards. A value close to 0 makes

the agent short-sighted by only considering immediate rewards, while a value close to 1 makes it more farsighted by considering future rewards more strongly.

- s' is the new state after action a is taken.
- $\max_{a'} Q(s', a')$ denotes the maximum expected future reward obtainable from the new state s'

Following the booming development of deep learning, Deep Reinforcement Learning, which is a combination of classic reinforment learning and deep learning architectures, began its development in the aim of creating a better control algorithm that is capable of dealing with more complex environments. A major breakthrough come from Volodymyr Mnih et al., where Mnih et al. introduced the first successful integration of deep neural networks with reinforcement learning, leading to the development of the Deep Q-Network (DQN) algorithm [36]. This approach allowed computers to learn optimal strategies directly from high-dimensional sensory inputs through end-to-end reinforcement learning, which was a significant breakthrough at the time.

The DQN algorithm, as presented in the paper, was applied to playing Atari 2600 video games, demonstrating that a single neural network architecture, combined with RL, could learn successful policies directly from raw pixel values to control the game. The key innovations introduced in the paper include the use of a deep convolutional neural network to approximate the Q-function, the employment of experience replay to break the correlation between consecutive samples, and the utilization of a separate target network to stabilize the learning process.

This work laid the foundation for numerous advancements in deep reinforcement learning, influencing a wide range of applications beyond gaming, from robotics to natural language processing. It showcased the potential of deep learning models to tackle complex control tasks by learning directly from high-dimensional, unstructured data.

On the other hand, DDPG, an actor-critic algorithm, effectively manages continuous action spaces [30]. DDPG adapts the foundational concepts of Q-learning and the deterministic policy gradient to work with deep function approximators, allowing it to handle the high-dimensional, continuous action spaces inherent in many real-world problems such as robotics, autonomous vehicles, and energy management. The paper highlights the use of a replay buffer to store and reuse past experiences, mitigating the issues of data correlation and non-stationary distributions, which are common in online learning environments. Furthermore, it introduces the concept of target networks to stabilize the training of deep function approximators, a technique that has become a standard in deep reinforcement

learning practices. Lillicrap's work on DDPG is pivotal as it demonstrates the feasibility of using deep learning approaches for complex control tasks that require precise, continuous actions, rather than the discrete decisions showcased in Mnih's Atari game environments. This has significantly broadened the applicability of deep reinforcement learning, opening new avenues for research and development in areas that demand nuanced control strategies.

Just like the filed of deep learning, th field of reinforcement learning is also at risk of adversarial attack. In 2017, Huang et al. showed that the DQN agent is vulnerable to the same attack applied to neural networks, such as the FGSM attack [21]. Similar defenses, such as adversarial training, are also shown to be effective in the field of reinforcement learning [43]. Lately, more research has been done to increase the robustness of reinforcement learning agents in the context of AV [19, 5].

2.4 Black-box Adversarial Attacks

The exploration of adversarial attacks has, for the most part, been rooted in first-order optimization methods. These methods, while powerful, often necessitate the availability of gradient information, making them impractical for real-world scenarios where such information may not always be accessible. The quest for gradient-free alternatives predates the recent strides in deep learning and adversarial machine learning. Traditional methods such as COBYLA and various Bayesian optimization techniques have been investigated extensively. However, these methods have demonstrated scalability limitations in dealing with modern, complex models that exhibit an ever-increasing dimensionality [44, 50].

Zeroth-order (ZO) optimization methods have emerged as a promising alternative, offering efficiency in computational resources while maintaining a competitive convergence rate [32]. A surge of interest in recent years has led to the development of several ZO optimization techniques, including but not limited to Zeroth Order Stochastic Gradient Descent (ZOSGD), ZO-SignSGD, and ZO-ADMM [32]. The appeal of these techniques lies in their ability to operate without explicit gradient information, thus bridging the gap between the theoretical world of optimization and the pragmatic constraints of real-world applications.

In this thesis, we focus mainly on the ZO-SignSGD method. Unlike other methods that use exact estimated gradient values, ZO-SignSGD utilizes the sign of the gradient to update the model parameters. This feature provides both computational advantages and practical feasibility, allowing the perturbation to converge in a relatively small amount of iterations [31]. We venture into an underexplored area by employing ZO-SignSGD as a

tool to study adversarial attacks on DRL algorithms in AVs, potentially expanding the understanding and application of black-box attacks in real-world scenarios.

2.5 Gaps in Existing Research

As discussed in the sections above, since the discovery of adversarial machine learning, many researches have been conducted to uncover the security concerns and vulnerability in a wide selection of machine learning algorithms such as image classification, object detection and deep reinforcement learning. However, as the field of adversarial machine learning is rather young, the focus of researches mostly are mostly in the field of computer science to understand the implications and underlying mechanisms for adversarial machine learning. Though autonomous vehicles has been increasingly utilizing deep learning based methods, there is not as much researches in the context of autonomous vehicles.

Many autonomous vehicles utilize object detection as a part of their vision system to fast and efficiently differentiates different objects on the roads. There have been many researches about the physical robust adversarial patches as mentioned in the previous sections, but they are hard to be tested for context of autonomous vehicles, as it is generally unsafe and difficult to be testing such attack on a real vehicles. At the same time, there have not been much researches published for generating, applying and testing the adversarial patch attacks for the purpose of autonomous vehicle research in a simulated environment. Therefore, it would be logical to create a tool to generate, apply and test the adversarial patches for autonomous vehicles.

For deep reinforcement learning, despite the initial interest in adversarial attacks on DRL, the amount of researches on this topic is waning, especially in the context of autonomous vehicle. Though many researchers have been actively studying the application of DRL in autonomous vehicle, not many papers are about the vulnerabilities in this scenario. Adversarial attack, being a relatively new form of vulnerability, has almost no paper in the realm of autonomous vehicle. As a result, it is essential to perform more research on this topic for the purpose of creating a safer environment for autonomous vehicles.

Chapter 3

White-box Adversarial Patch Generation and Testing in CARLA

3.1 Problem

Although adversarial examples are proven to pose a serious threat to CAVs in the real world, rigorous testing of the perception module may not be practical due to the danger it imposes and the amount of effort needed to craft and apply the example in the real world under all the different environments. Therefore, very recently, efforts have been made to use simulators such as the CARLA simulator to conduct adversarial machine learning research [12] [53] [33]. These efforts show a promising future for the physical adversarial attack to be tested and validated in the simulated environment to allow a faster and more comprehensive validation of the perception modules in autonomous vehicles.

Through the effort of Xiruo Liu et al. from Intel, a data collection platform was released for CARLA in 2022, but no research has yet been conducted with the platform [33]. At the same time, a wide collection of object detection adversarial attacks was substantially outdated and failed to work against the newer version of the algorithms as their architecture changed significantly. The attacks were also often tested in a controlled environment with limited camera movement, but real-life vehicles do not have such a limitation, thus needing more validation, such as different weather, lighting conditions, and camera movement. Therefore, this thesis first proposes an updated method to generate adversarial attacks against the state-of-the-art object detection algorithm, YOLOv5 [23]. Then, implement such attack in a new workflow to test and validate the adversarial attack with the consideration of real-life vehicle environment in CARLA.

3.2 Tools and Software

Over the years, many software have been developed by various companies or groups to help accelerate the research progress of the autonomous vehicles. As the purpose of this research is to validate the efficacy of adversarial attacks in a simulated environment, a photo-realistic 3D simulator is required. The CARLA simulator is the most popular simulator that serves as an open source platform designed to facilitate research and development in autonomous driving technologies [12]. It provides a realistic urban environment, complete with a variety of vehicles, pedestrians, and various weather conditions, allowing the comprehensive testing of driving models under different scenarios. With its extensible and customizable nature, CARLA has become a vital tool for researchers in the field of autonomous vehicles, enabling the rigorous evaluation of various machine learning algorithms, including those for perception, control, and adversarial resilience.

3.2.1 CARLA

CARLA, as the most popular photo-realistic open source simulation software in the space autonomous vehicle research, provides a whole suite of functionalities and API to help engineers and scientists across the world to perform CAV researches without the need to write a simulator from ground up. Built on the Unreal Engine for realistic simulations and adhering to the OpenDRIVE standard (version 1.4) for road and urban environment definitions, the simulator caters to a broad spectrum of autonomous driving applications, such as policy learning and perception algorithm training [12].

Being open-source, CARLA provides great documentation and tutorials. It also provides good interface to other popular programs in the robotics and autonomous vehicle field, such as Robot Operating System (ROS), MathWorks, SUMO, ANSYS etc. ROS is the most commonly used open-source software framework for robotics development, which includes autonomous vehicle development. The CARLA bridge allows a list of items to be easily communicated to ROS and back, this includes:

- Delivers LIDAR and Semantic LIDAR sensor information, along with data from Cameras (including depth, segmentation, rgb, and dvs), GNSS, Radar, and IMU.
- Supplies data on objects including their transformations, traffic light conditions, visual markers, collisions, and lane intrusions.
- Enables the management of Autonomous Driving (AD) agents via steering adjustments, acceleration, and braking controls.

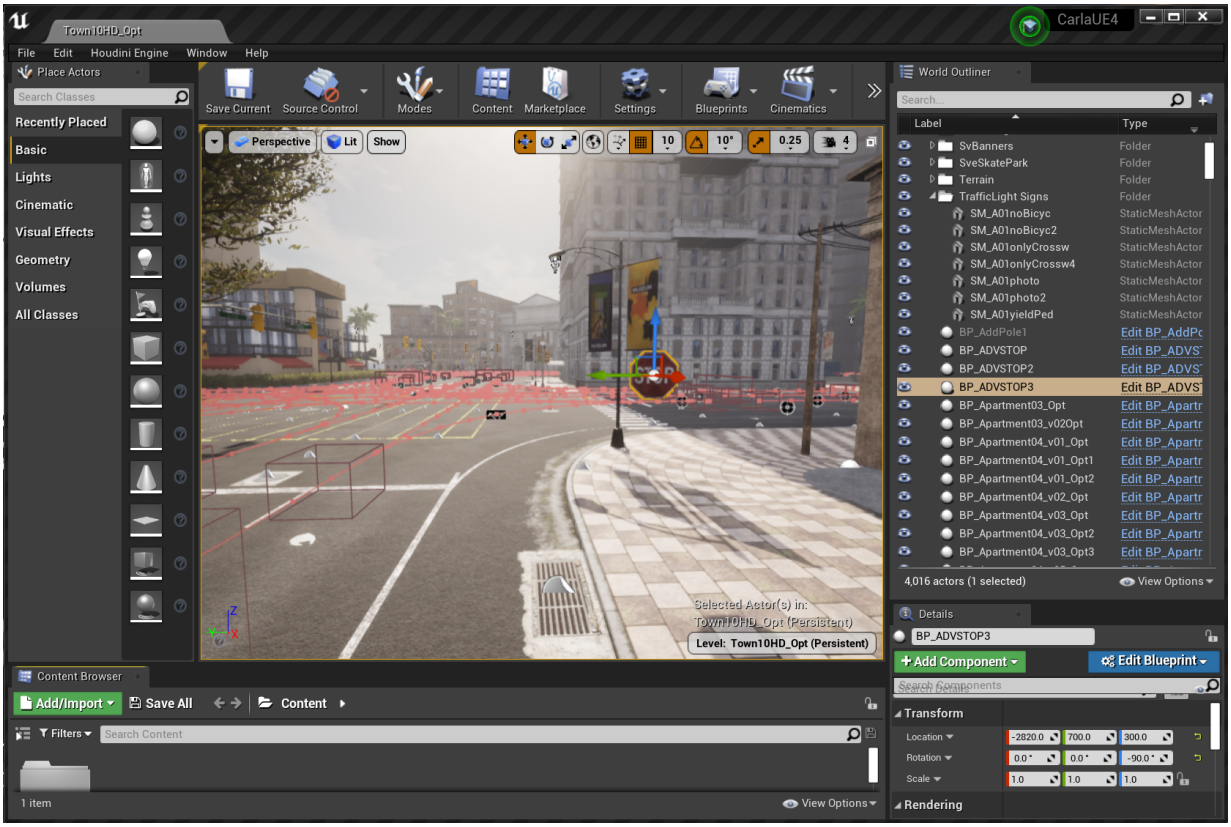


Figure 3.1: User Interface of the Unreal Engine of CARLA

- Allows manipulation of CARLA simulation settings such as synchronous mode operation, simulation playback and pause, and customization of simulation parameters.

Because this research is only a preliminar investigation on the adversarial attacks against DRL in CAV, this component is not used in this research. But having this functionality would eventually allows more sophisticated control algorithm to be developed, implemented and tested in CARLA. Making it easier to port to real life autonomous vehicle control once the simulation is done in CARLA. For future work on DRL and adversarial attack, the ROS bridge in CARLA would make accelerate the development time of the research by eliminating the work needed to work out the communication tool between these two software.

As seen in Figure 3.1, the unreal engine portion of CARLA allows more user control of the environment. The 3D environment produced by unreal engine is rather realistic for

a simulator, allows this research to perform the adversarial patch attack. As with normal unreal engine, the user can import 3D model assets, applying texture and adding material with ease. The modifications done by CARLA to the unreal engine also allows the custom assets to be packaged and used by other CARLA clients by importing the package for a easier time on the transferring of assets.

The most important feature of CARLA is the ability of controlling it through its API. Using the python API, CARLA is capable of doing everything related to autonomous research, such as spawning a car and drive it. CARLA itself consists of server and client. The server is the unreal engine simulation and the client is the code that user runs in order to gather information or make changes to the simulation. In the CARLA simulation, actors are entities that engage in activities and can influence other actors. This category encompasses vehicles, pedestrians, as well as sensors, traffic signs, traffic lights, and the spectator.

Vehicle actor is an actor that contains the most amount of parameters and physics, which is expected from a autonomous vehicle simulator. The API can directly control the vehicle by sending commands of throttle, steering or braking. It can also change the gearing of the vehicle or the wheel physics to simulate different tires and road conditions. The vehicle can also be set in auto driving mode control by the CARLA traffic manager which is crucial for the precise and effective training and evaluation of autonomous driving systems. The traffic manager offers various alternatives for mimicking traffic and particular traffic situations.

Pedestrian actor is similar to the vehicle actor in the sense that their can be programmable. They can have a certain direction and speed. It is also possible to use a built-in movement controller to control their movement by setting a goal and speed.

Sensors actor is another important aspect of CARLA. It allows the simulation of the sensors that are used in real-life autonomous vehicles. The sensors it can spawn are the following, but not limited to :

- Depth: Shows the depth of the objects in the environment in a grey-scale image
- RGB: Like real-life camera, shows a picture of environment
- Optical Flow: Shows the movement of the pixel in the camera.
- Semantic segmentation: Shows the ground truth tag of the objects in the camera view

- Instance segmentation: Shows the ground truth tag and the object ID of the objects in the camera view
- DVS: Dynamic Vision Sensor (DVS), operates in a fundamentally different manner from traditional cameras. Rather than capturing images at regular intervals, it asynchronously detects changes in light intensity, producing a continuous flow of events that represent changes in pixel brightness.
- Lidar: A rotating LIDAR sensor produces a 4D point cloud, where each point in the cloud includes coordinates and intensity information to accurately represent the environment.
- Semantic LIDAR: A rotating LIDAR sensor creates a 3D point cloud, enriched with additional details on instance and semantic segmentation for each point, to comprehensively model the environment.

The API itself also allows changes to the environment. As all objects in the environments are tagged, they can searched and toggled on or off during the simulation. The map itself can also be changed, since CARLA has multiple built-in maps, from a busy urban city center to a rural town with only small houses and few intersections. A major function utilized in this research is the ability to change the texture of a building through its API. By carefully inserting a object and change mutiple parameters, it is possible to change its texture on the fly inside the simulator without having to manually change it in the Unreal Engine interface.

3.2.2 Open Source Data Collection Platform

Due to the cost and danger of testing physical adversarial attacks in real life on autonomous vehicles, many attempts have been made to use CARLA as an alternative platform to conduct research on these attacks. Tong Wu et al. collected data in CARLA and performed a black-box searching algorithm to generate adversarial texture on vehicles that evade object detection [57]. Michael Threet et al. created a new way of inserting adversarial patches into CARLA, but it is hard to replicate without knowing the inner works of the rendering engine [53]. Xiruo Liu et al. provided an open source platform for data collection for the purpose of benign adversarial machine learning research in CARLA [33].

3.3 Method

The thesis first generates a patch based on the idea of expectation over transformation and backpropagates using the gradient of the YOLOv5 network. Then the patch is applied to a custom stop sign in CARLA to allow the change of texture on the fly with CARLA API. Finally, a predetermined scenario is generated and recorded using the platform created by Xiruo Liu et al. [33]

3.3.1 Patch Generation

To generate the patch, the idea behind other robust physical patches is "expectation over transformation" (EOT) as seen in equation 3.1 presented in its original paper [1].

$$\delta = \mathbb{E}_{t\sim T}[d(f(t(x')), t(x))] \tag{3.1}$$

This equation calculates expectation for the distance between the transformed adversarial example output, $f(t(x'))$, and the transformed original output $t(x)$ over a series of transformation. It directly measures the robustness of the patch through this calculation, constrains the optimization of the patched image and the original image to be similar to the classifier while undergoing a series of transformation, making the adversarial patch robust against transformations.

Though the original concept was used on a classifier, the same concept is used in other adversarial patch generation for object detection [10] [54]. For this particular patch generation, the patch goes through a wide range of transformations that to ensure its robustness in the real world. The transformations include random rotation, random scaling, random translation, random brightness, random contrast, random saturation, and random hue. The patch is also constrained to be within the range of 0 and 1 to ensure the patch is valid.

To prevent the stop sign from getting detected, the loss function focus on reduce the final score of the grid calculated by YOLOv5, where it is the multiplication of the objectiveness score and the class category confidence score. To further reduce the chance of detection, the loss function also includes a total variation(TV) loss to ensure the patch is smooth and not noisy. A noisy image is not only harder to print or displace in real life, the noisy details are also often lost when viewed by cameras at distance. The final objective function that includes the losses is shown in equation 3.2.

$$x' = \arg\min_{x'} \mathbb{E}_{t\sim\mathcal{T}} \left[\frac{1}{N} \sum_{i=1}^{N} \text{MaxFS}_i(t(x')) + \lambda \text{TV}(t(x')) \right] \tag{3.2}$$

Where:

- $\text{MaxFS}_i(t(x'))$ is the maximum final score for the i^{th} example in the batch after transformation t.
- $\frac{1}{N}\sum_{i=1}^{N}$ averages the maximum final scores across the batch of N examples.
- $\text{TV}(t(x'))$ is the total variation(TV) loss for the transformed adversarial patch.
- λ is a hyperparameter to balance the two loss terms.
- $\mathbb{E}_{t\sim\mathcal{T}}$ is the expectation over the transformations t sampled from $\mathcal{T}$.

Using this objective function, the patch is generated through a number of iterations using the Adam optimizer. The patch is initialized as a random noise image and is optimized to minimize the objective function, as seen in algorithm 3.

3.3.2 CARLA

CARLA Simulator is an open source simulator for autonomous driving research [12]. It provides a realistic urban environment, complete with a variety of vehicles, pedestrians, and various weather conditions, allowing the comprehensive testing of driving models under different scenarios. With its extensible and customizable nature, CARLA has become a vital tool for researchers in the field of autonomous vehicles, enabling the rigorous evaluation of various machine learning algorithms, including those for perception, control, and recently adversarial resilience.

Recently, Xiruo Liu et al. created a data collection platform for adversarial machine learning research in CARLA [33]. The data collection platform allows the user to create a scenario with a variety of parameters, including the number of vehicles, pedestrians, weather, and lighting conditions. This platform also allows the user to insert a custom object into the environment, which can be used to test the adversarial patch. The platform then records the scenario and the data collected by the sensors in CARLA, including the RGB image, depth image, and semantic segmentation image. This platform is used to test the adversarial patch generated in the previous section.

Data: *model*, parameters including patch shape, learning rate, etc.
Result: Optimized adversarial patch
Initialization
begin
    Initialize patch with random values Prepare optimization tools: optimizer, scheduler, etc.
end
Procedure GeneratePatch(x)
begin
    for *each step in max iterations* **do**
        Create a new batch of transformations using the current patch
        `// Compute loss`
        Evaluate the model on the batch to compute objectness and confidence
        final score = objectness*confidence
        Calculate the total variation loss losses = final score + variation loss
        `// Update the patch`
        Backpropagate the combined loss to get patch gradients
        Update the patch using the optimizer
        `// Post-processing`
        Clamp patch values to maintain valid pixel range
        Adjust learning rate using the scheduler
    end
    Save the final optimized patch
end

Algorithm 3: Adversarial Patch Generation Loop

To insert the patch into CARLA, a custom model is imported into CARLA as a static object. The model is a stop sign with a custom texture that is generated using the adversarial patch. The model is then inserted into the environment using the CARLA API. The model is then placed in front of the ego vehicle to ensure the ego vehicle will detect the stop sign. The ego vehicle is then controlled by the data collection platform to drive towards the stop sign. The platform then records the data collected by the sensors in CARLA, including the RGB images. The images are then used to evaluate the effectiveness of the adversarial patch by running the images through the YOLOv5 object detector.

3.4 Results

The pictures were first generated using the adversarial patch generation algorithm. As seen in figure 3.2, the stop sign was firstly applied with the patch then transformed randomly to ensure its robustness. An example of the losses over the iterations can be seen in figure 3.3. It is clear to see that both total variation loss and max score for the class stop sign are decreasing over the iterations. The final patch applied on the stop sign can be seen in figure 3.4. This texture is then applied into a custom model of a stop sign that allows texture change through the CARLA API. The image is then recorded using the data collection platform created by Xiruo Liu et al. [33]. In CARLA, the simulator is ran through a predetermined route and environment configuration such as weather, lighting, and camera movement. The simulator is ran for 10 frames, then the images run through the YOLOv5 object detector to evaluate the effectiveness of the patch. A sample of such detection can be seen in figure 3.7.

To ensure the robustness of the patch, different transformations have been applied to the patched texture. This includes random rotation, random scaling, random translation, random brightness, random contrast, random saturation, and random hue. The transformations were turned on and off to investigate their effect on the patch generation. Other parameters such as the total variation loss and patch size are also investigated. The patched image was then gone through the YOLOv5 detector to get a preliminary result on the effectiveness of the patch before applied into CARLA. The results of the different configurations can be seen in table 3.1. All the configurations and their static image detection can be seen in figure 3.5 and figure 3.6. Random noise patch is also provided as a reference. As seen in figure 3.5h, a random noise patch does not reduce or evade any detection. From figure 3.5, it is easy to see that with no transformations, the detector can still not detect the stop sign, the patch itself also does not resemble any recognizable shapes as the optimizer mostly focus on lowering the total variation loss as soon as the detection

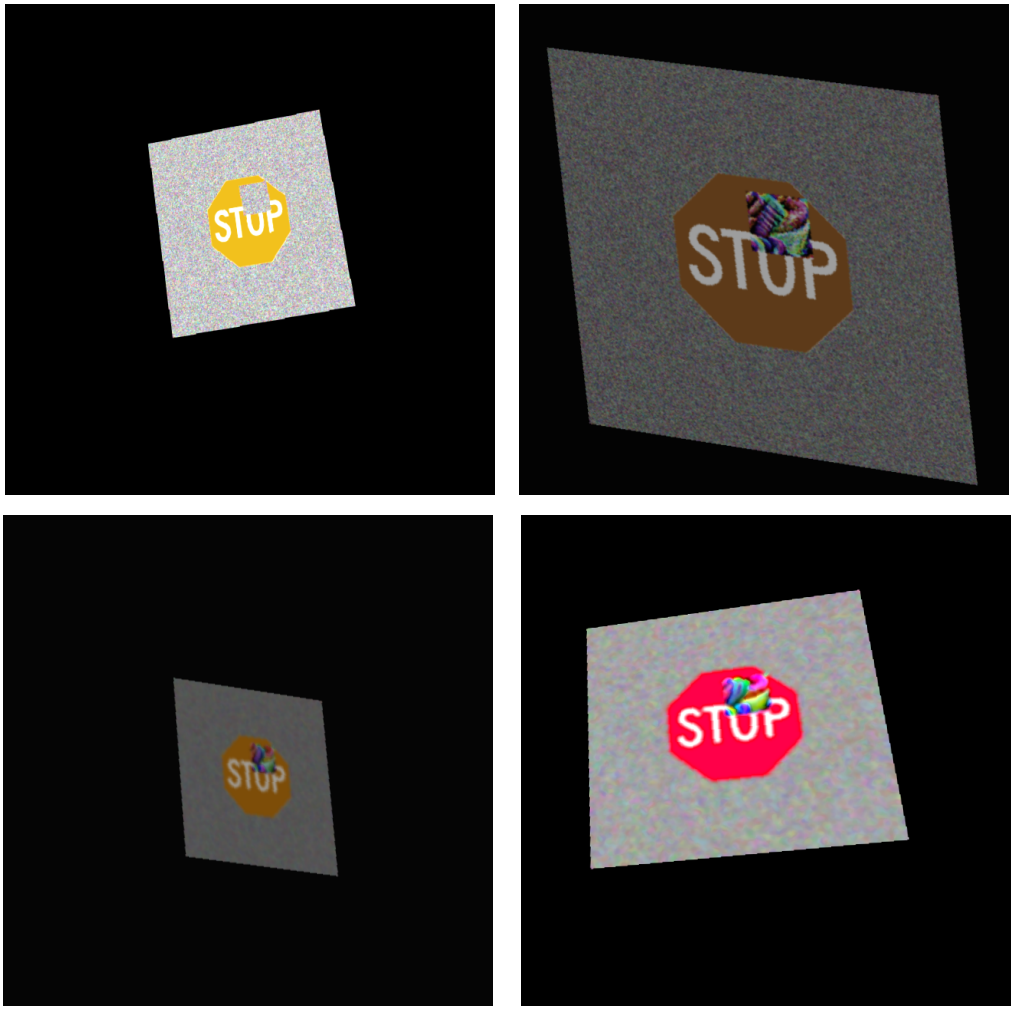


Figure 3.2: Examples of transformation used for adversarial patch generation

on this single image is evaded. It is also worth noting that though the transformations and the starting noise of the patch are different, this untargeted attack is able to quite consistently generate a patch that fools the detector to the closest item in the COCO data set, which is a cake. With the total variation loss weight change seen in figure 3.6a and figure 3.6b, we can also see a big drop in noisy colors seen in the patch generated while still maintain an evasion of the detector. A smaller patch size also does not perform as well as the original patch size, as seen in figure 3.6c, but figure 3.6d performs well in this static image detection task.

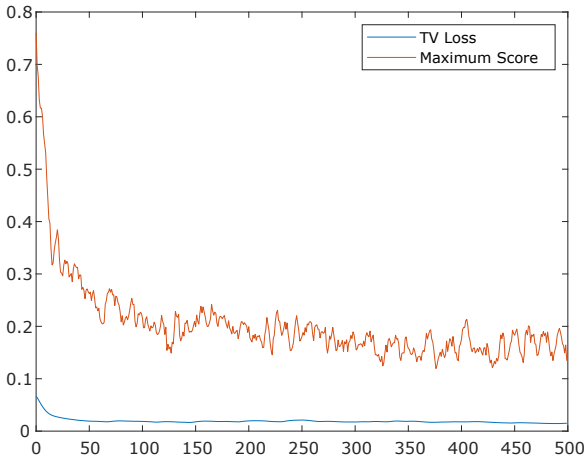


Figure 3.3: Losses over iterations for one single patch generation

Figure 3.4: Final adversarial patch generated

After patches are generated, they are imported into CARLA through the use the data collection platform and tested in a sunny and rainy environment as seen in figure 3.7. The results of the different configurations can be seen in table 3.2.The confidence are recorded and averaged out over the 10 frames that are recorded by the data collection platform in CARLA. If the stop sign is not detected by YOLOv5, the confidence recorded is 0.

As seen from the table 3.1, the more transformation we add the patch, the higher the final loss is. As seen in equation 3.2, the final loss is the sum of the maximum final score and the total variation loss. The maximum final score is the multiplication of the objectiveness score and the stop sign class category confidence score. Though, it may seem that the higher the loss, the worse performing the patch is; when there is no transformation at all, it is way easier to reach a lower loss, but the patch is not robust at all. The patch is only robust when there are transformations applied to it. The more transformations applied, the more robust the patch is, as it can be seen in table 3.2. When all transformations are applied, the patch is more robust in both sunny and rainy weathers compare to patches generated with less transformations. For both weathers, an optimized patch got an average confidence reduction of roughly 50% and 30% than a random noise patch. Furthermore, comparing to a stop sign with no patch, the patch significantly reduce the confidence of the stop sign detection by 70% and 38% in a sunny and rainy weather respectively. The patch does not work as well in the rainy weather comparing to sunny weather, due to the worse lighting condition, amount of fog and other artifacts presented in the rainy weather that

Table 3.1: Different configurations on the patch generation and its effectiveness on a static image outside of CARLA

	Max Score	Total Variation loss	Total Loss	Top Confidence	Label
All Transformations	0.1409	0.0152	0.1561	0.827	Cake
No Color Jitter	0.0593	0.0164	0.0756	0.634	Cake
No Color Jitter, Noise	0.0586	0.0184	0.077	0.825	Cake
No Color Jitter, Noise, Perspective	0.1055	0.0174	0.1229	0.813	Cake
No Color Jitter, Noise, Perspective, Rotation	0.0503	0.0185	0.0688	0.509	Snowboard
No Color Jitter,Noise, Perspective, Rotation, Resise	0.0006	0.0038	0.0045	0.664	Cell phone
No Transformations At All	0.0001	0.0064	0.0065	n/a	
3 x Total Variation Loss Weight	0.1541	0.0381	0.1922	0.847	Cake
5 x Total Variation Loss Weight	0.1066	0.0474	0.154	0.678	Cake
32 x 32 Patch Size	0.7112	0.0097	0.7209	0.934	Stop Sign
64 x 64 Patch Size	0.4072	0.0187	0.4259	0.873	Cake
Random Noise	n/a	n/a	n/a	0.92	Stop Sign

prevents the patch from being revealed completely and thus decrease the evasion power of the patch. The patch also works better in the sunny weather due to the higher contrast between the stop sign, the patch and the background, allowing the patch to be seen much more easily by the YOLOv5 detector.

Total variation loss is also tuned to be more significant in the training processes to see if it plays a role in the effectiveness of the patch. Because the total variation loss measures the amount of variation of the colors in the patch, a bigger weight allows the patch to be smoother, as the trainer emphasize more on the total variation of the patch. In table 3.1, the increase weight in total variation loss, does not reflect a significant performance change in the static image detection as both images successfully evades the detection by YOLOv5. When imported into CARLA, due to the smoothness of the patch, the patch can be seen at different angles and distance with much changes to its effectiveness. At three times of the original weight, the patch performs better in both sunny and rainy weather, though not by much. As the weight of the total variation loss increase more, the patch becomes less effective.

The size of the patch is also investigated to see how much it affects the patch performance. The original size of the patch is 100 x 100, then size of the patch is changed from 32 x 32 to 64 x 64 with all other parameters unchanged. As seen in table 3.1 and table 3.2, both smaller patch sizes performed worse than the original. This is expected because less area of the stop sign is covered up by the patch. At the same time, a smaller patch is also harder to be seen by the detector at further distances, therefore allowing the stop sign to be detected in more scenarios. As shown in table 3.2, when the patch is too small at 32 x 32, the detector can detect stop sign without any hurdles as if there is nothing on the

Table 3.2: Different configurations on the patch generation and its effectiveness inside CARLA

	avg. stop sign confidence (sunny)	Median	avg. stop sign confidence (rain)	Median
All Transformations	0.203	0.000	0.489	0.489
No Color Jitter	0.257	0.487	0.518	0.615
No Color Jitter, Noise	0.211	0.257	0.423	0.466
No Color Jitter, Noise, Perspective	0.268	0.000	0.470	0.423
No Color Jitter, Noise, Perspective, Rotation	0.288	0.268	0.577	0.470
No Color Jitter, Noise, Perspective, Rotation, Resise	0.397	0.288	0.625	0.577
No Transformations At All	0.401	0.417	0.548	0.625
3 x Total Variation Loss Weight	0.193	0.000	0.420	0.420
5 x Total Variation Loss Weight	0.253	0.253	0.431	0.431
32 x 32 Patch Size	0.614	0.614	0.770	0.777
64 x 64 Patch Size	0.433	0.489	0.688	0.688
Random Noise	0.441	0.483	0.564	0.579
No Patch	0.657	0.658	0.791	0.791

stop sign. However, when the patch is doubled in size, the patch performs much better in comparison.

Table 3.3: Stop sign detection confidence in different locations in CARLA

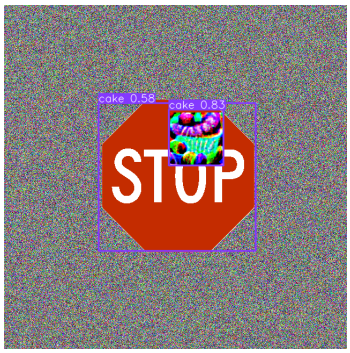
Location	1		2		3	
	Sun	Rain	Sun	Rain	Sun	Rain
No Patch	0.657	0.791	0.696	0.699	0.613	0.615
Patch	0.203	0.489	0.585	0.542	0.572	0.557
Noise	0.441	0.564	0.610	0.666	0.594	0.598

The patch is also applied to additional locations seen in figure 3.8 compare to the first location seen in figure 3.7. The results of the different locations can be seen in table 3.3. The major difference in between the locations are the background and the lighting affected by the relative location of the sun in the setup. In location 1, the sun is able to directly shine on the stop sign, allowing the patch to be seen in its full brightness. In location 2, the sun is on the side of the stop sign, partially lighting up the sign. While in location 3, the sun is behind the stop and the behind completely block the sun. As a result, seen in table 3.3, the patch performs the best in location 1, and the worst in location 3, where the reduction of the confidence is the most significant in location 1 and the least in location 3. As the patch is already relatively dark in the sunny weather for both location 2 and 3, the effect of the rain does not affect the patch as much as it does in location 1. It is still worth noting that the patch is also able to perform better than a random noise patch in

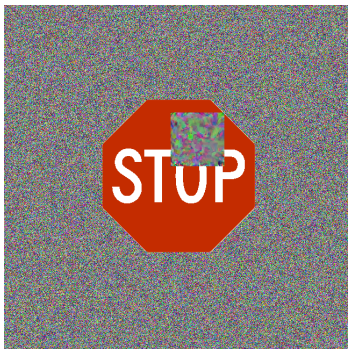
all locations and weathers, though the difference is small in some cases, it still showcase the relative effectiveness of the patch in different locations and weathers.

3.5 Summary

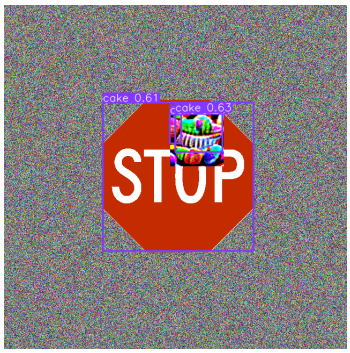
This study presented a novel approach to generating adversarial patches that effectively evade detections by state-of-the-art object detection algorithms in CAVs, mainly focusing on the YOLOv5 model. Through rigorous experimentation and optimization, including varying patch sizes, transformations, and total variation loss weightings, we successfully demonstrated the ability of these patches to reduce detection confidence significantly under varied simulated environmental conditions. We also showcase the impact of patch size on the confidence of the object detector under the same optimization and simulation condition. The use of the CARLA simulator proved instrumental in providing a realistic and safe testing environment for these adversarial attacks. The visibility and brightness of the patch greatly affects the effectiveness of the patch on the stop sign, where it can only be reliably tested in a simulator. Our findings highlight the ongoing need for robust testing and validation of machine learning models used in CAVs, emphasizing the importance of considering real-life variables such as weather and lighting. The results also underscore the potential vulnerabilities in current CAV perception systems and pave the way for further research into more resilient and secure machine learning applications in autonomous vehicle technologies.



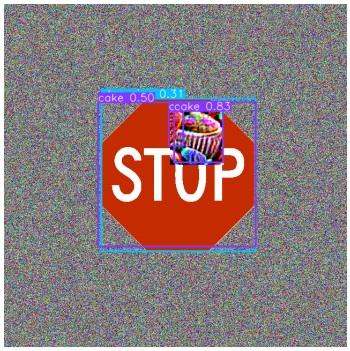
(a) All Transformations



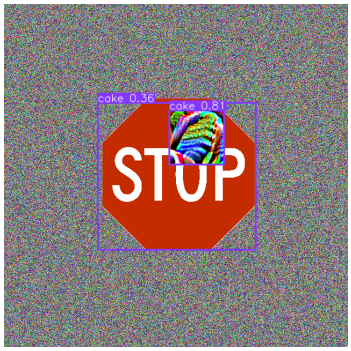
(b) No Transformations



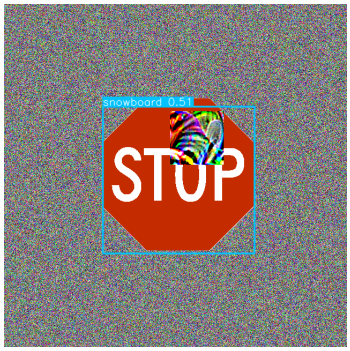
(c) No Color Jitter



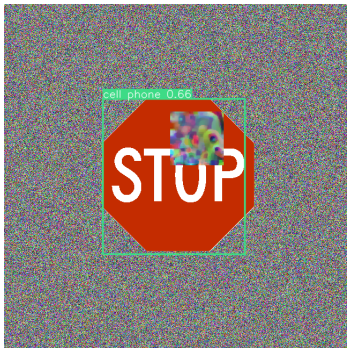
(d) No Color Jitter, Noise



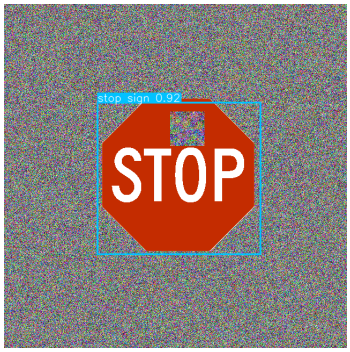
(e) No Color Jitter, Noise, Perspective



(f) No Color Jitter, Noise, Perspective, Rotation

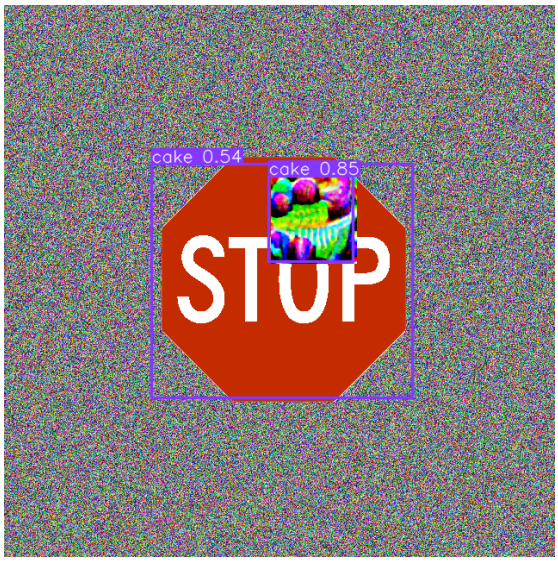


(g) No Color Jitter, Noise, Perspective, Rotation, Resize

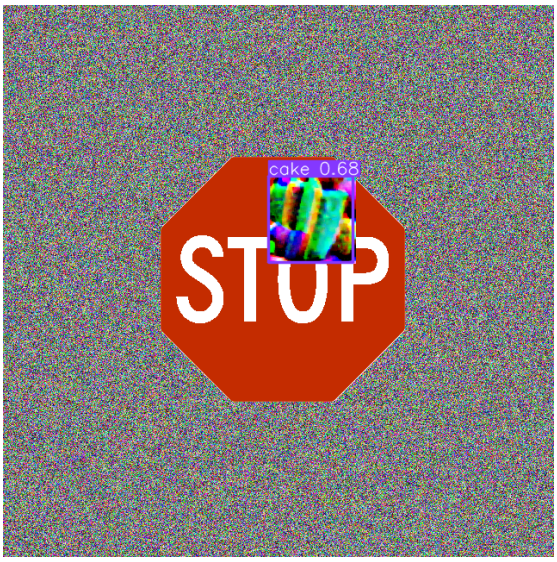


(h) Random Noise

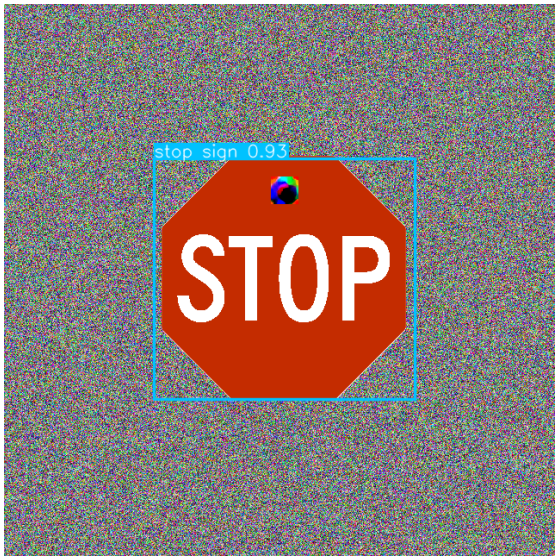
Figure 3.5: Different transformation configurations on the patch generation and its effectiveness on a static image outside of CARLA



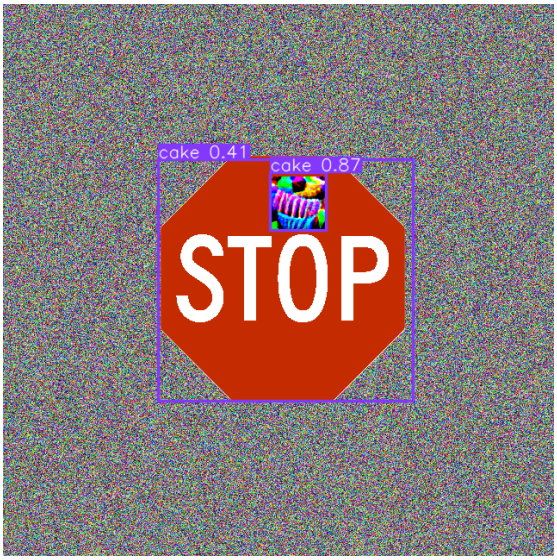
(a) 3 x Total Variation Loss Weight



(b) 5 x Total Variation Loss Weight

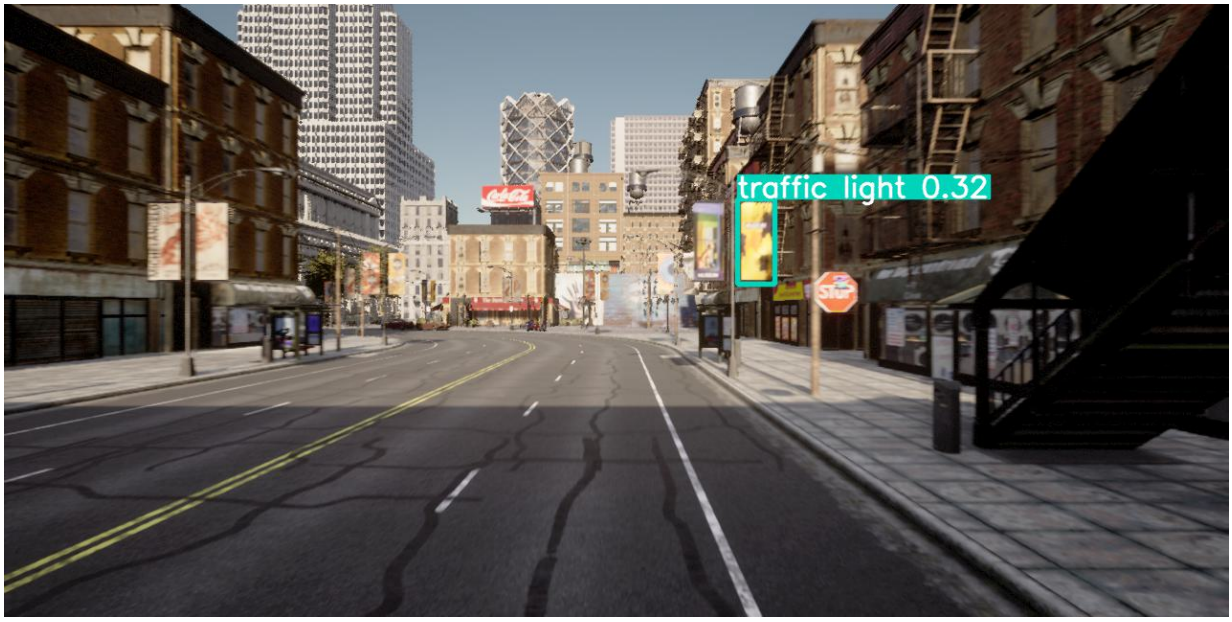


(c) 32 x 32 Patch Size



(d) 64 x 64 Patch Size

Figure 3.6: Additional configurations on the patch generation and its effectiveness on a static image outside of CARLA



(a) Sunny

(b) Rainy and Foggy

Figure 3.7: Stop sign with adversarial patch applied in CARLA

(a) Patch applied in location 2

(b) Patch applied in location 3

Figure 3.8: Stop sign with adversarial patch applied in different locations in CARLA

Chapter 4

Black-box Adversarial Attacks and Defenses on DRL

4.1 Problem

This chapter focuses on applying, attacking, and defending a highway lane-changing deep Q network (DQN) agent and a Deep Deterministic Policy Gradient (DDPG) agent. In this chapter, we assume that the vehicle has been compromised without detection, allowing the adversary to access and manipulate sensor data, thereby altering the states perceived by the DRL agent. Given the increasing adoption of detection algorithms for common attacks, adversarial machine-learning strategies are employed to maximize damage to DRL agents whilst minimizing the chance of detection theoretically. These strategies introduce minimal perturbations to maintain stealth and reduce the likelihood of detection during the attack. It's worth noting that this chapter does not delve into the specifics of the vehicle's attack surface or penetration method. To assure the performance of the unattacked agent, the reinforcement learning algorithms are based on Stable Baselines 3, an online RL library written in Python [45]. The training environment is based on the 'highway-env' library to allow faster deployment, hyperparameter tuning, and debugging, as seen in Figure 4.1 [29].

4.1.1 Environment

The environment is a lightweight highway lane-changing environment compatible with the OpenAI gym interface [29]. The gym environment also allows it to interface with

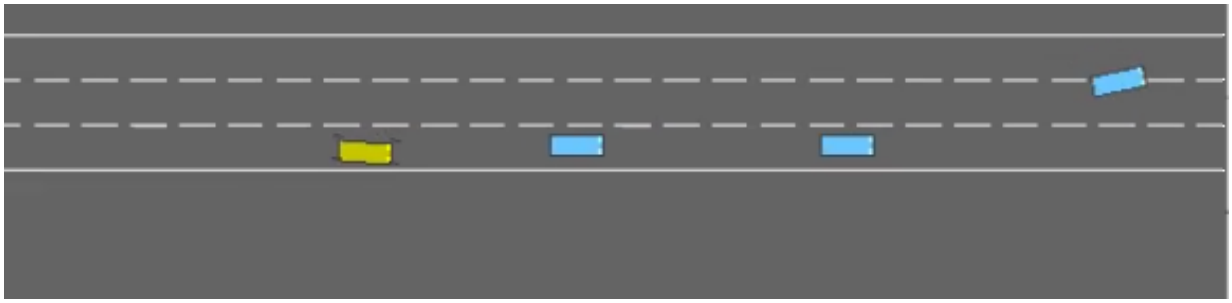


Figure 4.1: An example render of the highway lane changing environment

the popular reinforcement learning training package, Stable Baselines 3, without much additional tweaking. Environment vectorization provided with Stable Baselines 3 also significantly expedites the training process by allowing multiprocessing of the DRL training.

The environment's observation space tracks the vehicle's kinematics on the highway. That includes the position and velocity of the ego vehicle. It also records the relative position and velocity of other vehicles on the highway. The observation space is normalized relative to the ego vehicle. The position is normalized with the bound of $[-100, 100]$, and the velocity is normalized with the bound of $[0, 20]$.

During initialization, all vehicles, including the ego vehicle, are randomly positioned on the highway, ensuring a minimum separation between them. Vehicles, excluding the ego vehicle, adhere to a randomly initialized Intelligent Driver Model and the Minimizing Overall Braking Induced by Lane change (MOBIL) model [29].

For DQN, the environment's action space is a high-level discrete action space with five actions. The actions are defined as:

- Action 1: change lane to the left
- Action 2: idle (do nothing)
- Action 3: change lane to the right
- Action 4: accelerate
- Action 5: decelerate

Simple proportional controllers control the lower-level actions such as specific heading, velocity, and acceleration when each action is chosen, so the reinforcement learning agent only needs to make a high-level decision on which action to take.

4.1.2 Reward

The reward function rewards the agent for staying in the right lane at a faster speed while penalizing the agent for collision. The reward function is defined as for DQN:

$$\begin{aligned} R(s,a) =& RightLaneReward + \\ & 0.4 \cdot \frac{v - v_{min}}{v_{max} - v_{min}} + collision \end{aligned} \tag{4.1}$$

The collision reward is set to -1. So that the agent will seek to move faster while avoiding a collision. RightLaneReward is set to 0.1 when the agent is traveling on the right-most lane. Due to the limited action space capabilities of DQN, such reward function is well suited for DQN.

For DDPG, the action is a continuous action space for the kinematics of the ego vehicle with a dimension of 2. The first dimension is the acceleration of the ego vehicle, and the second dimension is the steering angle of the ego vehicle. Both actions are normalized to $[-1, 1]$.

Since now the agent have the freedom of steering whenever it wants, the reward function needs to prevent the agent from deviating too much from the middle of the lane as well as changing the steering too rapidly. As a result, compare to the DQN reward function, the DDPG reward adds two more terms to the reward function, lane centering reward and abrupt action penalty. The reward function is defined as:

$$\begin{aligned} R(s,a) =& RightLaneReward + \\ & 0.4 \cdot \frac{v - v_{min}}{v_{max} - v_{min}} + collision + \\ & laneCenteringReward + abruptActionPenalty \end{aligned} \tag{4.2}$$

where:

$$\begin{aligned} laneCenteringReward =& 0.01 \cdot \frac{1}{1 + 4 * (Distance\_from\_Center)^2} \\ abruptActionPenalty =& 0.019 * abs(steering\_change) + \\ & 0.001 * abs(acceleration_change) \end{aligned} \tag{4.3}$$

As seen in Equation 4.3, the lane centering reward is inversely proportional to the distance from the center of the lane. By penalizing the square of the lateral distance,

small deviations from the center are lightly penalized, while larger deviations result in a significantly lower reward, thus strongly encouraging the vehicle to stay close to the center of the lane. The abrupt action penalty is proportional to the change in steering and acceleration. This encourages the policy to make smoother, more gradual steering adjustments, which is desirable for comfort and safety. Overall, the reward function is designed to encourage the agent to stay in the right lane, maintain a high speed, and avoid collisions while penalizing the agent for deviating from the center of the lane and making abrupt actions.

4.2 Policies

4.2.1 DQN Policy

The agent is trained with the DQN algorithm [36]. Similar to Q learning, the underlying structure of the model is Markov Decision Process Equation 4.4.

$$\begin{aligned} Q^{new}(s_t, a_t) = Q(s_t, a_t) + \\ \alpha(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - \\ Q(s_t, a_t)) \end{aligned} \tag{4.4}$$

- $Q(s_t, a_t)$: Q value of the current state and action
- α: learning rate
- r_t: reward of the current state and action
- γ: discount factor
- s: state
- a: action

However, for deep Q learning, the Q function is approximated by a neural network. The neural network is trained with the DQN algorithm. This algorithm uses a replay buffer to store the experience of the agent. The replay buffer samples a batch of experiences to train the neural network. The DQN loss function is defined as:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \cdot \left[\left(r + \gamma \cdot \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right] \tag{4.5}$$

- θ: the parameters of the neural network
- θ^-: the parameters of the target network
- D: the replay buffer

Like Q learning, the network loss is the reward plus the discounted maximum Q value of the next state minus the current Q value. But in this case, the gradient descent method minimizes the loss function. The network is trained with the Adam optimizer [26]. For a more straightforward implementation, Stable Baseline 3 is used to train the model [45].

4.2.2 DDPG Policy

The agent is trained with the stable baseline library implementation of the DDPG algorithm [30, 45]. In contrast to DQN, which only deals with discrete action spaces, DDPG allows the handling of continuous action spaces, making it particularly suitable for AV, where actions are often continuous, like acceleration and steering angle. Another main difference is that DDPG combines the actor-critic approach with insights from Deep Q-Networks (DQN). The actor in this setup is responsible for determining the best action given the current state, while the critic evaluates the chosen action's quality. As seen in Equation 4.6, the actor updates in the direction that maximizes the Q value of the current state and action. On the other hand, the critic is updated based on the Temporal Difference (TD) error, which is the difference between the critic's current estimate of the Q-value and the improved estimate yielded by the latest action from the actor. This process, similar to Q-learning, involves the use of a learning rate to balance the weight between the old and new estimates:

$$\nabla_{\theta^\mu} J \approx \frac{\sum_i \nabla_a Q(s, a|\theta^Q)|s = s_i, a = \mu(s_i) \nabla \theta^\mu \mu(s|\theta^\mu)|_{s_i}}{N} \tag{4.6}$$

4.3 Zeroth Order Attack

This work uses the zeroth order optimization as an adversarial attack.

4.3.1 Attacker Model

The attacker model defines the attacker's opportunity, intent, and capability to place the work in context. The attacker has the opportunity to influence the system during operation at the level of sensor outputs. The attacker's short-term goal is to disrupt the agent to the point that the agent will cause a collision on the road. The capabilities of the attacker consist of the following: (1) the attacker can influence a sensor value up to a 10% deviation of its actual value, (2) an attacker can influence sensor values immediately (i.e., from the start of the vehicle until a shutdown), continuously (i.e., no cool down periods), and indefinitely (i.e., for as long as the attacker wants), and (3) the attacker can influence any sensor, and all sensors at the same time (i.e., there no assumption that any single sensor in the set provides an actual value).

4.3.2 Zeroth Order SignSGD

The ZO-SignSGD method is a gradient-free (zeroth order) optimization method that uses the sign of the gradient to update the model [31].

The ZO-SignSGD method implemented is defined as:

Algorithm 4 shows the implementation of the ZO-SignSGD attack for lane changing. The input variables, such as learning rate, initial value, and number of iterations, are tweaked to ensure a fast convergence while minimizing the perturbation size. As a black box optimization algorithm, the first step is to estimate the gradient. A gradient of a function can be estimated by adding a small perturbation to the input data. For a high-dimension function such as the neural network used in the DRL, the gradient must be computed by summing all estimated gradients over perturbations of random directions. To achieve a fast convergence of the algorithm, similar to the Fast Gradient Sign Method, only the sign of the gradient is used. This also avoids the error introduced by the numerical value of the estimated gradient. The perturbation is then calculated by multiplying the sign of the gradient with the learning rate to minimize the objective function seen in Equation 4.10. This process is repeated until it reaches the maximum number of iterations. The perturbed observation is then used to get the action from the policy, thus, continuing into the next step.

Using this algorithm, the specific objective for this attack is crafted with two losses in mind. The first loss is the distance between the target action and the original action, which can be seen in Equation 4.7. The same is true for both DQN and DDPG. "a" is the constant that controls the weight of the loss.

Data: ZO-SignSGD
Input: learning rate $\{\delta_k\}$, initial value x_0, and number of iterations: T
def *GradEstimate(x, μ, q, d)*:
    for $k = 1, 2, ..., q$ **do**
        u = normalized(random number);
        $\hat{g}_k = \hat{g} + \frac{d(f(x+\mu u)-f(x))}{\mu} u$;
    end
def *optimization(x)*:
    for *k=1, 2, ..., T* **do**
        $\hat{g}_k$ = GradEstimate(x_k);
        $x_{k+1} = x_k - \delta_k \, sign(\hat{g}_k)$;
    end
def *Main*:
    for *i in range of timesteps* **do**
        while *not done* **do**
            action = model.predict(observation);
            env.step(action);
            perturbed_obs = optimization(observation);
            observation = perturbed_obs;
        end
    end

Algorithm 4: Implantation of ZO-SignSGD for Lane Keeping. Adversarial observation is calculated for each step.

$$\mathcal{L}1_{DQN} = a \cdot norm(Q(x + \delta, y_{target}; \theta) - Q(x + \delta, y; \theta)) \tag{4.7}$$

$$\mathcal{L}1_{DDPG} = a \cdot norm(action(x + \delta; \theta) - action(x; \theta)) \tag{4.8}$$

The second loss is the distortion caused by the perturbation, as seen in Equation 4.9. This calculates the distance between the original observation and the perturbed observation.

$$\mathcal{L}2 = norm((perturbed\ obs - original\ obs)^2) \tag{4.9}$$

$$Objective : \min(\mathcal{L}1 + \mathcal{L}2) \tag{4.10}$$

When crafting the perturbation, both losses are added together to minimize both during the optimization. For a successful attack, both losses will converge and be minimized. Figure 4.4 shows an example of this convergence. Since Zeroth Order SignSGD is not a constrained optimization method, the perturbation is not guaranteed to be small. At the same time, since it is a gradient-free stochastic optimization method, the attack can fail to converge within the given number of iterations.

4.4 Approach & Evaluation

4.4.1 Adversarial Training

As seen in many adversarial machine learning papers, adversarial training is a method to train a model to be robust to adversarial attacks [7, 43]. A general Procedure can be seen in Algorithm 5.

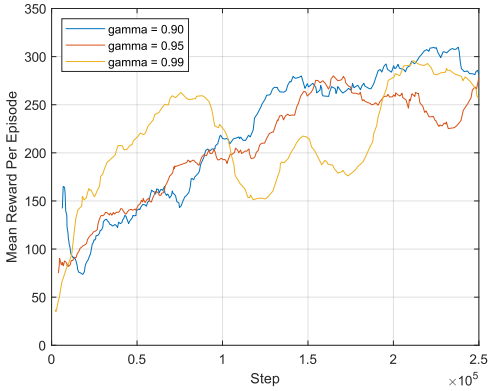
Data: Adversarial-Training
for $i = 1, 2, ..., timestep$ **do**
    attack the observation $Q(obs, a, \theta)$;
    obs' = ZO SignSGD($Q(obs, a, \theta)$);
    a' = $Q(obs', a, \theta)$;
    new obs, reward = env(a',s);
    Train policy as per DQN or DDPG algorithm;
end

Algorithm 5: Adversarial training of the policy.

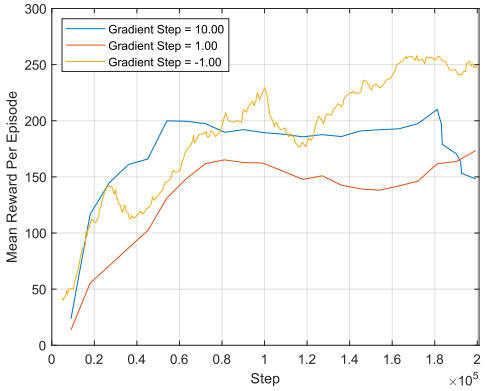
The algorithm is based on the methods discussed in [43]. Though it may appear simple, this algorithm has proven successful against the trained perturbation method for both deep learning and DRL models.

4.4.2 Initial Training

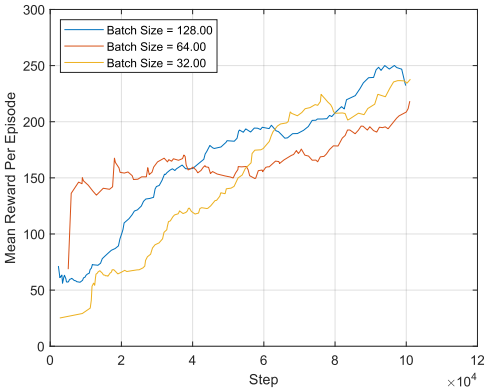
The DQN algorithm is first trained for 20,000 time steps with the default hyperparameter included in stable baseline 3. The model can learn the environment and achieve a mean



(a) The effect of the discount factor on the mean reward per episode.



(b) The effect of the gradient steps on the mean reward per episode.



(c) The effect of the batch size on the mean reward per episode.

Figure 4.2: Comparative effects of various parameters on the mean reward per episode.

reward of 310.58 per episode. For most of the episodes, The policy can navigate the highway for the entire episode without fail.

The DDPG model is trained for 120,000 time steps. A higher time step allows the actor and critic to converge on this lane-changing task. Due to the continuous action space control by the DDPG algorithm, the reward is not as stable as DQN. However, the model can still achieve a mean reward of 232.78 per episode. Note that DDPG is trained with noise added to the action space as part of the exploration strategy.

As DDPG is more complex and requires more training, environment vectorization is

performed to speed up the training process. The environment vectorization allows the model to train with multiple environments at the same time, thus speeding up the training process, as seen in Figure 4.3. The model is trained with up to 20 environments at the same time. This allows the model to learn from different experiences at the same time, thus speeding up the training process. This speed up also allows additional hyperparameter tuning to be performed, such as the discount factor, gradient steps, and batch size. The effect of these hyperparameters on the mean reward per episode can be seen in Figure 4.2.

Discount factor, gamma, is an important parameter that controls the importance of future rewards, where gamma close to 1 encourages the agent to consider future rewards more strongly, promoting policies that may sacrifice immediate rewards for greater long-term returns. In this task, three different gamma values of 0.9, 0.95, and 0.99 are tested. As seen in Figure 4.2a, for this specific setup, all three gamma values ended up in around the same mean rewards. However, when the gamma is set to 0.99, the model is less stable and has a higher variance in the mean reward per episode. This is likely due to the model overestimating the future reward, causing the model to be less stable. Therefore, a gamma of 0.95 is chosen for the final training.

The gradient steps parameter in Stable Baselines3, particularly for algorithms like DDPG, controls the number of gradient updates that are performed after each rollout. When gradient step is set to 1, regardless of how many steps were taken in the environment during the rollout, only one gradient update will be performed afterwards. This means that after each rollout, the model's parameters are updated just once. However when it is set to -1, the number of gradient updates will match the number of steps taken in the environment during the rollout. Shown in Figure 4.2b, the more gradient updates, the better the model can learn from the environment, allowing the policy to achieve higher mean rewards per episode. Thus, the gradient steps are set to -1 for the final training.

Batch size is the number of samples that are processed before the model is updated. A larger batch size allows the model to learn from more experiences at the same time, which can often allow more stable training process and faster convergence time. As seen in Figure 4.2c, the larger the batch size, the better the model can learn from the environment, allowing the policy to achieve higher mean rewards per episode for the majority of the training process. Thus, the batch size is set to 128 for the final training.

The maximum reward obtainable by the agents per episode would be 450, assuming it never crashes, is always on the right-most lane, and always travels at high speed. However, such theoretical maximum reward is unobtainable as the agent must slow down to change lanes and avoid crashes. At the same time, As long as other cars are in the right-most lane, the agent is unlikely to obtain the full reward for the step. Getting a higher reward also

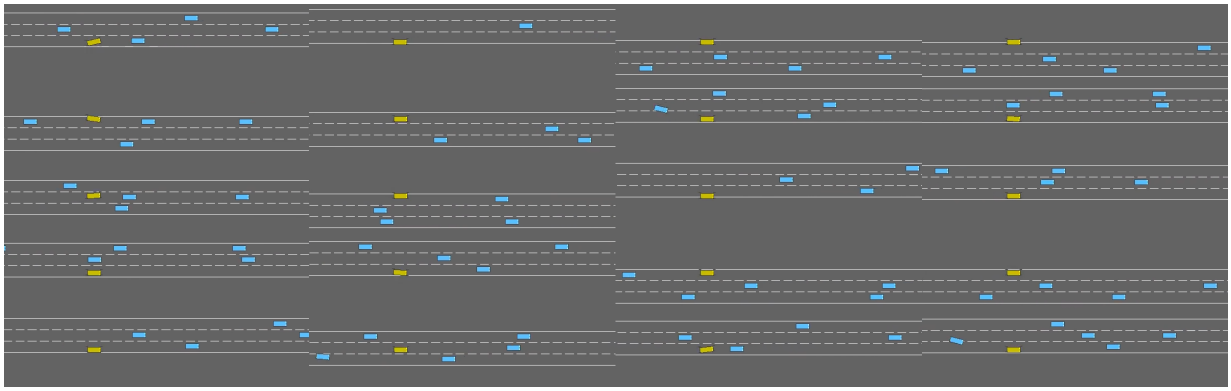


Figure 4.3: Training with vectorization of 20 environments.

requires the agent to have a constant heading going forward. This has proven to be difficult to maintain for a DDPG agent where the policy has control of the steering. Therefore, both agents performed relatively well in this task.

4.4.3 Attacks

The idea of the attack is to mimic a real-world scenario where the attacker has access to the vehicle's sensors, enabling them to craft perturbations to the observation space. The perturbation is generated at every step with the consideration of perturbation sizes. The maximum iteration per step for perturbation crafting is set to 100. However, most successful perturbations are created within 50 iterations. An example of the loss for a successful perturbation is shown in Figure 4.4. The perturbation converges under 100 iterations. As the iterations go on, the distortion becomes the focus of the optimization program and, as a result, shrinks with iterations. The targeted action for the DDPG policy is set to be $[1, 0.5]$, meaning full throttle and turning right. For DQN, since the action space is high level, the targeted attack is chosen to be "accelerate" (Action 3) to prevent the ego vehicle from changing lanes at all. The attack successfully causes the model to turn right most of the time, causing the mean reward per episode to plunge, as seen in 4.2.

The attack is unconstrained with the loss function defined in Equation 4.7. However, the size of the perturbation is directly correlated to the parameters for ZO SignSGD. The larger step each iteration of the gradient takes, the bigger the perturbations. Therefore, the parameters are carefully tuned to allow the attack to converge quickly while maintaining a reasonable perturbation size to enable a distortion to within 0.2, representing the 10% deviation from its original value, while allowing the attack to be carried out within 100

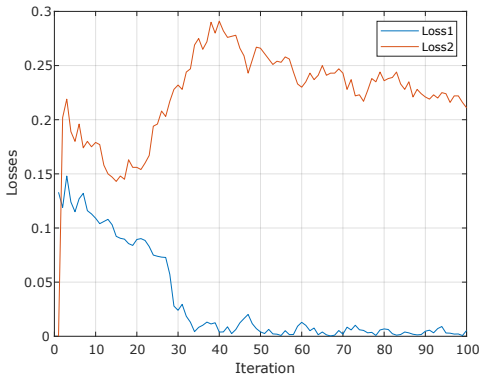


Figure 4.4: Loss convergence for successful perturbation for both Loss1 and Loss2.

Figure 4.5: Result of the perturbation. The ego vehicle hits another car.

iterations. Small perturbations created by adversarial machine learning like this may help the attack avoid possible detection. Since ZO SignSGD can minimize both Loss1 and Loss2 as defined in Equation 4.7 and 4.9, as the iteration grows, distortion can be minimized if parameters are tuned in such a way. This trend already can already be seen in Figure 4.4.

4.4.4 Defenses

Employing the approach outlined in Algorithm 5, we further trained the DQN and DDPG policies that were initially compromised by the adversarial attack for an extra 5,000 and 10,000 time steps, respectively, this time incorporating adversarial observations into the learning process. The progression of the reward per episode during this adversarial training phase can be visualized in Figure 4.6 and Figure 4.7.

Despite the initial attack, both policies exhibited a marginal increase in the reward per episode following adversarial training, suggesting some degree of learned resilience against adversarial manipulation. However, it is noteworthy that this increase was rather insubstantial, particularly in the case of DQN. Moreover, despite maintaining identical training parameters, the mean reward per episode for both policies during adversarial

Attack Skip Probablity	Unattacked Mean Reward	Attacked Mean Reward
0	109.65	56.79
0.3	164.42	34.03
0.5	145.00	41.44

Table 4.1: Mean reward of adversarial training with attack skip chance.

training was lower than that achieved during the initial training phase.

This decline in performance is likely to be attributed to overfitting to the perturbed observations. The model's parameters have essentially learned to respond specifically to the adversarial patterns in the observations, thereby diminishing its performance under normal conditions. This is particularly evident in Figure 4.7, where the reward per episode shows a declining trend, a classic indicator of overfitting.

To further investigate this hypothesis on model performance during adversarial training, we conducted a series of experiments with varying probabilities of omitting the attack at each step. Specifically, we executed adversarial training sessions consisting of 50,000 steps, setting the attack skip probability to 0, 0.3, and 0.5, respectively. The results, summarized in Table 4.1, reveal a notable trend. With no attack skipping (a probability of 0), the model exhibited lower performance in scenarios without attacks, achieving a mean reward of 109.65, but demonstrated improved resilience in the presence of attacks, with a mean reward of 56.79. In contrast, increasing the attack skip probability to 0.3 and 0.5 resulted in enhanced performance in unattacked scenarios, with mean rewards of 164.42 and 145.00, respectively. However, this improvement came at the cost of reduced effectiveness under attack conditions, where the mean rewards decreased to 34.03 and 41.44, respectively. These findings proves a potential overfitting of the model to the adversarial conditions when the attack is consistently applied during training. By introducing a variable probability of skipping the attack, it becomes possible to modulate the balance between the model's robustness to attacks and its overall performance in standard, non-adversarial conditions. This adjustment mechanism can be instrumental in optimizing the model for a desired level of resilience versus performance in applications where the threat landscape and operational conditions may vary.

The trade-off between robustness and performance in adversarial settings is a well-documented challenge in machine learning literature. A seminal 2019 paper elucidated this issue by demonstrating worsened generalization performance of deep learning networks under adversarial training [46]. More recently, in 2022, potential explanations for this trade-off were proposed, such as the lower utility of robust features for generalization tasks

or the insufficiency of datasets for adversarial training [11]. This dilemma is manifested in our experiment, wherein the adversarially trained policies underperformed compared to their non-adversarially trained counterparts. This underscores the complexity of designing reinforcement learning policies that are both robust to adversarial attacks and proficient at their designated tasks.

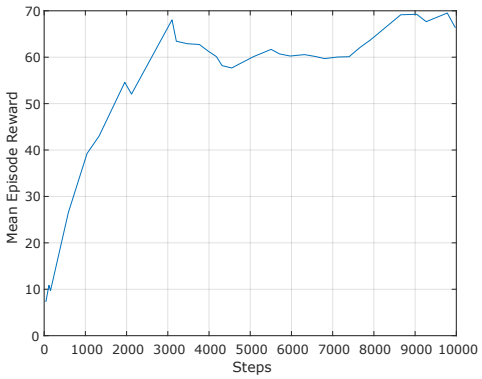


Figure 4.6: Reward/Episode for adversarial training of DDPG.

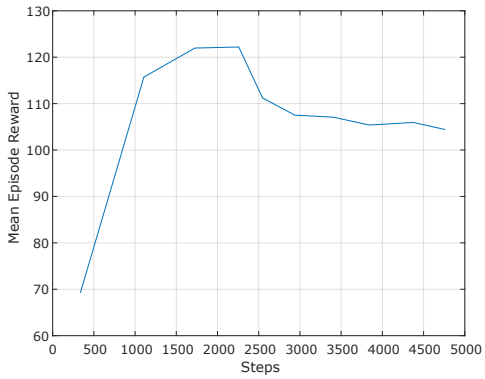


Figure 4.7: Reward/Episode for adversarial training of DQN.

	Policy			
	DQN		DDPG	
Scenarios	Reward	[% of theoretic max]	Reward	[% of theoretic max]
Initial Training	310.58	69.01%	232.78	51.73%
Under Attack	108.22	24.89 %	45.44	10.09 %
Adversarial Training	111.08	22.78 %	62.40	13.87 %
After Training (No Attack)	149.33	33.18 %	91.22	20.27 %

Table 4.2: Mean reward of each model with the maximum theoretical reward of 450.

4.4.5 Results

A table of mean rewards for each model is shown in Table 4.2. The highest obtainable reward per episode is 450, with one reward per step if the agent reaches high speed, stays in the right lane, and does not crash. The environment has completely random vehicle

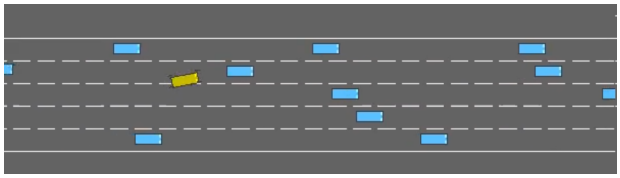


Figure 4.8: Example of the environment with 5 lanes and a vehicle density of 2.

layouts every single time. Therefore the agent would not be able to obtain the total 450 rewards as it is often required to slow down, change lanes out of the right lane to avoid a collision or maintain the high-speed reward.

The reward is calculated by finding the mean of rewards for 100 episodes. The environment calculates the reward, as shown in Equation 4.1. For *Initial Training* and *After Training*, no attacks are performed on the observation. The rewards are collected to show the generalized performance of the model. For *Under Attack* and *Adversarial Training*, the attack is performed on the observation to force the policy into performing only one action, if the perturbation converges within the given number of iterations.

As seen in Table 4.2 in the scenario *Initial Training*, the agents performed relatively well, setting up a good baseline performance of the policies. However, both policies fail to perform as the attacks take place in *Under Attack*. Some marginal performance gain is seen after the policies undergo *Adversrial Training*. But overall generalized performance is suffering as seen in *After Training*.

The DDPG agent in this work is trained in three lanes as seen in Figure 4.1, but to test the robustness of the policy, it is necessary to test it on unseen environments. Therefore, the environment is tested on 4 and 5 lanes, as well as with a higher vehicle density. The results are shown in Table 4.3. An example of the environment with 5 lanes and a vehicle density of 2 is shown in Figure 4.8. As seen in Table 4.3, the DDPG policy is able to generalize to the unseen environment. But, an reduction in the mean reward per episode is also seen accross all scenarios. This policy in particular does not fare well in a high traffic environment, due to a lack of training in such an environment. The DDPG policy is still vulnerable to the attack, as seen in the *Attacked* scenario. The mean reward per episode is significantly lower than the *Unattacked* scenario. This shows that the DDPG policy is still vulnerable to the attack, even in the unseen environment.

Table 4.3: Lane and Vehicle Density effect on mean reward

Lane	3		4		5	
Vehicle Density	1	2	1	2	1	2
Unattacked	272.32	95.00	157.23	55.36	200.65	98.58
Attacked	31.24	25.71	51.16	30.19	49.95	29.33

4.5 Summary

In this work, we harnessed the ZO-SignSGD method to craft perturbations capable of triggering the failure of trained reinforcement learning models. Remarkably, these attacks were successfully carried out on both DQN and DDPG models by introducing perturbations to the observation space, even without access to the actual gradient information of the models. While the untouched models achieved high rewards — approximately 310 and 250, respectively — the targeted attacks significantly disrupted the performance of the ego vehicle, forcing it to follow the attacker's actions and plummeting the reward to near zero. This unique vulnerability underscores the vulnerabilities of reinforcement learning models to adversarial attacks even when the attacker lacks detailed model information.

In response to these successful attacks, we trained the models using these adversarial examples to enhance their robustness. Both models demonstrated an increased resilience, improving their rewards in the face of adversarial observations. However, it's important to note that adversarial training proved to be a time-intensive process, and the resulting models underperformed their original versions. This trade-off, where adversarial training dampens a model's generalization performance, mirrors findings observed in other machine learning applications [11].

Our adversarial attacks, while effective, are not yet optimized. Future work could draw inspiration from the adversarial attack strategies in the broader machine learning field, potentially leading to stronger and more efficient attacks. This could involve, for instance, targeting keyframes during the vehicles' operation. Moreover, testing the transferability of adversarial examples across different models could provide critical insights into the vulnerability of autonomous vehicles, particularly since deep reinforcement learning models often perform identical tasks. To prevent fast and catastrophic perturbations by attackers, it will be crucial to test these examples in real-world scenarios.

As demonstrated in this thesis, adversarial training is not a panacea for these adversarial threats. It may cause an unexpected loss of rewards if the model adapts too much

to the adversarial observation. Given the requirement for autonomous vehicles to function flawlessly under all circumstances. Further investigation into other defensive measures is imperative to build more robust and secure systems. These could include intrusion detection systems, model distillation, and model verification. Each of these could potentially contribute to a more comprehensive solution, mitigating the risks of adversarial attacks.

Chapter 5

Conclusion

In this thesis, I firstly introduced an innovative method for crafting adversarial patches that can successfully bypass the detection mechanisms of leading object detection systems employed in Connected and Autonomous Vehicles (CAVs), with a particular emphasis on the YOLOv5 framework. Through comprehensive experimentation, I meticulously explored various factors such as patch dimensions, transformations, and the influence of total variation loss adjustments. Our experiments, conducted under a spectrum of simulated environmental scenarios, convincingly showed that these patches could substantially diminish the detection confidence levels. Additionally, this study delved into the effect of patch size on the detection confidence, maintaining consistent optimization and simulation parameters throughout the investigation.

5.1 Adversarial Patch in Carla

The employment of the Carla simulator was pivotal in this research, providing a realistic yet controlled platform for conducting these adversarial trials, ensuring both safety and fidelity in testing. I observed that factors like the patch's visibility and luminosity play a crucial role in its efficacy, particularly noted in the case of stop sign detection, where reliable assessments were feasible solely within the simulated environment. These insights underscore the imperative for rigorous and comprehensive testing protocols for machine learning models in CAVs, taking into account the myriad of real-world conditions such as varying weather patterns and lighting conditions.

This findings shed light on the existing susceptibilities within the perception frameworks of contemporary CAV systems, setting the stage for ensuing inquiries into devising more

robust and secure machine learning solutions for autonomous vehicular technologies. This research underscores the criticality of adaptive and resilient defense mechanisms in safeguarding against the evolving landscape of adversarial threats in the realm of autonomous navigation.

5.2 Adversarial Attack on DRL

In next part of this thesis, I explored the efficacy of the ZO-SignSGD method in generating perturbations that compromise the integrity of trained reinforcement learning models, specifically targeting DQN and DDPG frameworks. These adversarial attacks, executed by altering the observation space, highlighted a critical vulnerability in reinforcement learning systems, significantly degrading performance without necessitating direct access to the model's gradient information. The original models, which performed optimally in standard settings, saw their rewards drastically reduced to near-zero levels under adversarial conditions, emphasizing the susceptibility of such models to even minimally informed attacks.

To counter these breaches, I incorporated adversarial examples into the training regimen, aiming to bolster the models' resilience against similar future attacks. This adversarial training, while enhancing robustness, introduced a trade-off by diminishing the models' overall performance, a phenomenon consistent with broader machine learning observations. Despite these efforts, the optimized state of this adversarial strategies remains unachieved, suggesting a potential avenue for future research to refine these tactics for more potent and efficient disruptions, especially in critical applications like autonomous vehicle navigation.

The outcomes underscore the complex challenge adversarial threats pose to reinforcement learning-based systems, particularly in high-stakes environments. While adversarial training offers a measure of defense, it falls short of a comprehensive solution, sometimes at the expense of the model's general effectiveness. This necessitates a broader exploration of defensive strategies beyond adversarial training, such as advanced intrusion detection, model distillation, and rigorous verification processes, to ensure the reliability and security of systems in adversarial settings. The pursuit of such multifaceted defense mechanisms is crucial for the advancement of robust, attack-resilient autonomous systems, safeguarding them against the evolving landscape of adversarial threats.

5.3 Future Work

Both parts of the research uncovers the vulnerabilities in the neural network based algorithms used in the autonomous vehicle, but in a very limited degree.

In the first research about the object detection algorithm in autonomous vehicle, though the patch works fine under good lighting, the patch fails to work under more harsh environments. To more vigorously test the capabilities of the vision system of an autonomous vehicle, the patch generation algorithm could be enhanced. At the same time, though YOLOv5 is a great object detection algorithm, it would be better if more object detection algorithms can be brought in and test the effectiveness of the patch on all the different algorithms. It would also be interesting to see the possibility of creating an universal patch based off the transferability of adversarial attacks that would work on the majorities of the object detection algorithms out there using a more rigorous patch generation algorithm.

For the second part of the research focusing on the vulnerabilities of DRL used in autonomous vehicle, a number of the setup can be improved to delve deeper into the danger of the attack and the defense against such attack. To begin, the research uses only a simple 2D gym environment to train, attack and defend the DRL policies. A more realistic simulation environment such as CARLA should be used to further test the practicality of such attack. This research also only tested two of the most common DRL algorithms, DQN and DDPG, while many other DRL algorithms are used for autonomous vehicle research. Therefore, it is only logical to test more DRL algorithms against adversarial attacks and see if they suffer from a similar vulnerability. Furthermore, other attacks can also be performed and tested on DRL algorithms to get a more comprehensive idea of the DRL vulnerability.

Most importantly, future research should improve on the defenses against adversarial attacks, for object detection or DRL, by applying a better adversarial training regim or through other means of defense to ensure a safe and secure environment for the future of autonomous vehicles.

References

[1] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. *CoRR*, abs/1707.07397, 2017.

[2] Adith Boloor, Karthik Garimella, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. Attacking vision-based perception in end-to-end autonomous driving models. *Journal of systems architecture*, 110:101766–, 2020.

[3] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models, 2018.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016. arXiv 1606.01540.

[5] Prasanth Buddareddygari, Travis Zhang, Yezhou Yang, and Yi Ren. Targeted Attack on Deep RL-based Autonomous Driving with Learned Visual Patterns. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10571–10577, 2022.

[6] Yulong Cao, Chaowei Xiao, Benjamin Cyr, Yimeng Zhou, Won Park, Sara Rampazzi, Qi Alfred Chen, Kevin Fu, and Z. Morley Mao. Adversarial Sensor Attack on LiDAR-Based Perception in Autonomous Driving. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2267–2281, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017.

[8] Yevgen Chebotar, Karol Hausman, Yao Lu, Ted Xiao, Dmitry Kalashnikov, Jake Varley, Alex Irpan, Benjamin Eysenbach, Ryan Julian, Chelsea Finn, and Sergey Levine. Actionable Models: Unsupervised Offline Reinforcement Learning of Robotic Skills, 2021. arXiv 2104.07749.

[9] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, CCS '17. ACM, November 2017.

[10] Shang-Tse Chen, Cory Cornelius, Jason Martin, and Duen Horng Chau. Robust physical adversarial attack on faster R-CNN object detector. *CoRR*, abs/1804.05810, 2018.

[11] Jacob Clarysse, Julia Hörrmann, and Fanny Yang. Why adversarial training can hurt robust accuracy, 2022. arXiv 2203.02006.

[12] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938, 2017.

[13] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. *CoRR*, abs/1707.08945, 2017.

[14] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Florian Tramèr, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Physical adversarial examples for object detectors. *CoRR*, abs/1807.07769, 2018.

[15] Marc Fischer, Matthew Mirman, Steven Stalder, and Martin Vechev. Online Robustness Training for Deep Reinforcement Learning, 2019.

[16] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[17] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[18] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[19] Xiangkun He, Haohan Yang, Zhongxu Hu, and Chen Lv. Robust Lane Change Decision Making for Autonomous Vehicles: An Observation Adversarial Reinforcement Learning Approach. *IEEE Transactions on Intelligent Vehicles*, 8(1):184–193, 2023.

[20] Yu-Chih-Tuan Hu, Bo-Han Kung, Daniel Stanley Tan, Jun-Cheng Chen, Kai-Lung Hua, and Wen-Huang Cheng. Naturalistic physical adversarial patch for object detectors. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7848–7857, October 2021.

[21] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial Attacks on Neural Network Policies, 2017. arXiv 1702.02284.

[22] David Isele, Alireza Nakhaei, and Kikuo Fujimura. Safe Reinforcement Learning on Autonomous Vehicles. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–6, 2018.

[23] Glenn Jocher. YOLOv5 by Ultralytics. DOI: 10.5281/zenodo.3908559, May 2020. Version 7.0. [Online]. Available: https://github.com/ultralytics/yolov5.

[24] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale, 2021. arXiv 2104.08212.

[25] Arne Kesting, Martin Treiber, and Dirk Helbing. General lane-changing model mobil for car-following models. *Transportation Research Record*, 1999(1):86–94, 2007.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2017. arXiv 1412.6980.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[28] Mark Lee and J. Zico Kolter. On physical adversarial patches for object detection. *CoRR*, abs/1906.11897, 2019.

[29] Edouard Leurent. An Environment for Autonomous Driving Decision-Making, 2018. GitHub.

[30] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019. arXiv 1509.02971.

[31] Sijia Liu, Pin-Yu Chen, Xiangyi Chen, and Mingyi Hong. signSGD via Zeroth-Order Oracle. In *International Conference on Learning Representations*, 2019.

[32] Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred O. Hero III, and Pramod K. Varshney. A Primer on Zeroth-Order Optimization in Signal Processing and Machine Learning: Principals, Recent Advances, and Applications. *IEEE Signal Processing Magazine*, 37(5):43–54, 2020.

[33] Xiruo Liu, Shibani Singh, Cory Cornelius, Colin Busho, Mike Tan, Anindya Paul, and Jason Martin. Synthetic dataset generation for adversarial machine learning research, 2022.

[34] Jiajun Lu, Hussein Sibai, Evan Fabry, and David A. Forsyth. NO need to worry about adversarial examples in object detection in autonomous vehicles. *CoRR*, abs/1707.03501, 2017.

[35] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*, 2018.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, 2013. arXiv 1312.5602.

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[38] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[39] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, 2016.

[40] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning, 2017.

[41] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks, 2016. arXiv 1511.04508.

[42] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528, 2015.

[43] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust Deep Reinforcement Learning with Adversarial Attacks, 2017. arXiv 1712.03632.

[44] M. J. D. Powell. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*, pages 51–67. Springer Netherlands, Dordrecht, 1994.

[45] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[46] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John C. Duchi, and Percy Liang. Adversarial Training Can Hurt Generalization, 2019. arXiv 1906.06032.

[47] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[48] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[49] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[50] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[51] Aman Sinha, Hongseok Namkoong, Riccardo Volpi, and John Duchi. Certifying Some Distributional Robustness with Principled Adversarial Training, 2020. arXiv 1710.10571.

[52] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014. arXiv 1312.6199.

[53] Michael Threet, Colin Busho, Josh Harguess, Melanie Jutras, Nicole Lape, Sara Leary, Keith Manville, Mike Tan, and Chris Ward. Physical adversarial attacks in simulated environments. In *2021 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, pages 1–5, 2021.

[54] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. Fooling automated surveillance cameras: adversarial patches to attack person detection. *CoRR*, abs/1904.08653, 2019.

[55] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805–1824, aug 2000.

[56] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

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

[58] Yichuang Zhang, Yu Zhang, Jiahao Qi, Kangcheng Bin, Hao Wen, Xunqian Tong, and Ping Zhong. Adversarial patch attack on multi-scale object detection for uav remote sensing images. *Remote Sensing*, 14(21):5298, 2022.