IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 15 June 2023, accepted 8 August 2023, date of publication 21 August 2023, date of current version 25 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3307133

RESEARCH ARTICLE

Priest: Adversarial Attack Detection Techniques for Signal Injection Attacks

JAEHWAN PARK AND CHANGHEE HAHN
Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, South Korea
Corresponding author: Changhee Hahn (chahn@seoultech.ac.kr)

This work was supported by the Research Program funded by the Seoul National University of Science and Technology (SeoulTech).

ABSTRACT Machine learning is widely used for autonomous driving because it can recognize surrounding circumstances feasibly from sensor and determine appropriate actions. Most of these sensors are based on micro-electro-mechanical systems (MEMS), which enable autonomous vehicles to judge objects in conjunction with object-detection algorithms. However, recent studies have shown that MEMS are vulnerable to signal-injection attacks, in which the input images are manipulated to force the object detection algorithms to misclassify the results. These attacks can be critical in the wild because they deteriorate state-of-the-art detection techniques, dropping their detection rates until the objects would no longer be detected at all. In this paper, we propose Priest, a novel detection method against prior signal-injection attacks. Priest uses the similarity of pixel values between two consecutive images. Using only two images ensures a low computational cost. According to our performance analysis, Priest detects state-of-the-art signal-injection attacks in real-time with 99% accuracy on average, achieving practical autonomous driving security.

INDEX TERMS Autonomous vehicle, adversarial attack detection.

I. INTRODUCTION

Autonomous driving is a promising technology based on machine learning (ML), which recognizes surrounding circumstances from sensor and determines appropriate actions. In these settings, the quality of the images captured from the sensors affects the overall object detection accuracy. Inertial sensors based on micro-electro-mechanical systems (MEMS) are widely used to reduce undesirable blur in the images. Specifically, MEMS enhance the values measured by the sensors, thus deriving clean images [14]. In the real world, image stabilization techniques integrated with MEMS based inertial sensors are frequently used in autonomous vehicles, assisting the vehicles as auxiliary sensors. However, sensors based on MEMS are vulnerable to resonant acoustic-injection attacks, significantly degrading object detection mechanisms. Moreover, adversarial attacks that disrupt correct operations are continually reported [6], [34]. These attacks can be disastrous in the wild because they can cause car accidents.

Recently, Ji et al. [14] introduced AMple (injecting physics into adversarial machine learning) attacks, a new class of system-level vulnerability that exploits the weaknesses in the physics of hardware using adversarial attacks in ML. Moreover, they proposed the Poltergeist attacks (PG attacks), the first kind of AMple attacks, which exploits the vulnerability of image stabilizers using adversarial images derived from acoustic manipulation to degrade the object detection algorithms. As a result, they generated blurry images that disrupted the detection algorithms. The resulting three types of PG attacks are hiding attacks (HA), creating attacks (CA), and altering attacks (AA). An HA is an attack in which the adversary disturbs an object detection algorithm, resulting in detection failures. A CA is a method that identifies non-existing objects as if they were present. An AA is a technique that induces misclassification in detection algorithms. These attacks are launched either internally or externally. Launching them in an internal manner requires installing an acoustic generator within a few tens of centimeters inside the target. Fortunately, this internal attacks would not be difficult for the passengers to identify. By contrast, external attacks do not require on-site access to the target,

The associate editor coordinating the review of this manuscript and approving it for publication was Amjad Mehmood.

thereby being difficult to be caught and prevented. Therefore, based on the existing attack methodology, we aim to defend against external attacks especially when the vehicles are stationary or standing. Specifically, among these attacks, an HA is highly devastating because it can severely affect the performance of object detection algorithms, decreasing the state-of-the-art detection mechanism's detection rate until the objects would no longer be detected at all. Moreover, many detection or defense studies [12], [17], [19], [20], [21], [22], [27], [41], [42] have used only a single image and measured their performance based on static datasets (such as MNIST or CIFAR-10), which are unsuitable for real-world driving conditions. Ren et al. [29] recently proposed a de-blurring method as a defense mechanism. This approach uses one-dimensional (1D) and two-dimensional (2D) recurrent neural networks (RNNs) to obtain denser propagation. However, this method is limited by the detection run-time, making it practically unusable in real-world autonomous driving systems. Likewise, the acoustic detection methods may be used as a countermeasure [4], [25], [32], but they have the limitations in terms of computational costs and real-world deployment of specific hardwares (see Section VI for details). Therefore, it is urgent to implement detection or defensive algorithms that are resilient to HA.

In this paper, we propose Priest, a novel detection scheme for autonomous vehicles. Priest uses the similarity of adjacent sequential images in dynamic real-time driving data because when vehicles are stationary and standing, the pixel values of sufficient portions of the image are almost invariant relative to fixed buildings, fixed traffic lights, and other stationary objects. If this similarity is less than a particular value, Priest determines that an attack has occurred. We use the sum of the differences of the pixel values in the same positions of two adjacent images as the metric because the change in pixel values itself indicates similarity. Furthermore, Priest ensures extremely fast run-time detection. To evaluate Priest, we extract data from standard driving image libraries (such as the KITTI and BDD100K datasets), over which we launch an HA. Priest guarantees an average of 99% detection accuracy with a run-time of 0.00563s.

A. CONTRIBUTION

In this paper, we make the following contributions:

- We propose Priest, the first detection mechanism against HA, validated on a real-world driving scenario dataset with sufficient detection rates. Therefore, Priest can be immediately applied to practical autonomous driving systems.
- Unlike most prior methods [1], [13], [19], [27], [42] that rely on only a single static image, Priest uses sequential dynamic images. Because driving in the real world involves not only static but also dynamic images, Priest is much more practical than prior methods.
- By using only two adjacent images for detection, we dramatically reduce the computational cost. According to our experiments with real-world datasets, Priest achieves approximately 99% detection accuracy in a run-time of only 0.00563s. Therefore, our method is appropriate for detecting real-world attacks.

B. ORGANIZATION

The remainder of the paper is organized as follows. In Section II, we explain the background. We then introduce the threat model, called PG attacks, in Section III. In Section IV, we describe the overall structure of our detection model, Priest, in detail, followed by the detection rate and run-time results in Section V. We provide related work in Section VI. Finally, we conclude this paper and propose supplementary work for our model in Section VII.

II. BACKGROUND

This section introduces object detection, adversarial attacks, image stabilization, and MEMS inertial sensors.

A. OBJECT DETECTION

An autonomous vehicle detects the surrounding environment based on an object detection algorithm. The detection algorithm process is as follows: First, the light reflected by the object is converted by sensors, such as charge-coupled devices or complementary metal-oxide-semiconductor (CMOS), into the digital image. Second, ML classifies the objects based on the digital images from the sensors. Finally, the detection algorithm assesses a confidence score, which is a probability value indicating that objects exist within a specific bounding box. If the confidence score increases, it is more likely that the detection succeeds. Typical object detection algorithms are YOLO V3/V4/V5. Among these, YOLO V5 performs best in detection accuracy and run-time.

B. ADVERSARIAL ATTACKS

An adversarial attack reduces the accuracy of the model by inserting a particular noise or perturbation. The adversary manipulates a clean image into an adversarial example, which leads the object detection algorithm to misclassify it. Three popular adversarial methods are the fast gradient sign method (FGSM), Carlini and Wagner (C&W) attacks, and projected gradient descent (PGD).

In [12] and [36], they proposed the adversarial attack method FGSM using the following equation.

$$X = X + \epsilon \, sign(\nabla L(\Theta, I_c, l)),$$

This method uses the sine value of the gradient of the loss of an image, where ϵ controls the size of the noise or perturbation. As a result, modern deep neural networks that use linear behavior to reduce computational gains are vulnerable to the FGSM [1].

Carlini et al. [6] introduced a compelling attack method for each of L_0, L_2, and L_∞. This method has a better attack performance and a smaller perturbation size than previous attacks. Importantly, they also showed that the defensive

distillation method used as a powerful defense method could not resist this attack method.

Madry et al. [20] proposed the PGD method using the saddle point (min-max) function. They found the minimum loss in the adversarial example that causes the maximum loss in the model. Thus, they allowed a powerful adversarial example to be classified as the correct class.

C. IMAGE STABILIZATION

Because the real world is dynamic, many objects move and change. Thus, when the camera captures images, motion within the exposure period causes blur. If the motion increases, the degree of blur also increases. Many modern cameras apply image stabilization techniques to reduce this motion blur. Some common image stabilization techniques are as follows:

- Mechanical image stabilization: This technique uses an external stabilizer to compensate for camera motion. By rigidly observing in one direction, the external device assists the main sensor in acquiring high quality images [2].
- Optical image stabilization: This system reduces the blur by physically shifting the camera lens or sensors. Specifically, gyroscopes (angular sensors) detect the vertical and horizontal movement of the camera body, and then the lens moves in the opposite direction based on the data from the angular sensors [5].
- Digital or electronic image stabilization: This reduces blur using image processing software [8], [23].

These techniques combined with MEMS inertial sensors to compensate for blurry images.

D. MEMS INERTIAL SENSOR

A MEMS inertial sensor detects inertial stimuli and generates a signal to transmit the information using mechanical structures. The MEMS inertial sensor consists of a gyroscope and an accelerometer. A gyroscope operates based on the Corio-lis force or Coriolis effect [33] and has three DOFs (degrees-of-freedom): roll, pitch, and yaw. An accelerometer detects linear acceleration by a mass spring structure [39]. The accelerometer also has three DOFs: the x-, y-, and z-axes. The performance and cost of inertial sensors have been continually enhanced. However, despite this improvement, many studies [14], [33], [38], [39], [40] have shown that MEMS inertial sensors are vulnerable to resonant acoustic injection attacks because the sensing mass is sensitive to an acoustic signal similar to the frequency of the mechanical system. As a result, an adversary can manipulate the outputs from inertial sensors, thus disrupting the operation of a drone or causing misclassification in autonomous vehicles.

Specifically, Ji et al. [14] proposed PG attacks, which cause misclassification in object detection algorithms by injecting manipulated acoustic signals into the MEMS inertial sensors. They introduced false camera motion (FCM) caused by false sensor outputs, denoted by $\vec{M}_f = \{\vec{a}_x, \vec{a}_y, \vec{a}_z, \vec{w}_r, \vec{w}_p, \vec{w}_y\}$. Then, they expressed compensatory camera motion (CCM) equal in size to the FCM but in the opposite direction of the attack compensatory method, denoted by $\vec{M}_c = \{-\vec{a}_x, -\vec{a}_y, -\vec{a}_z, -\vec{w}_r, -\vec{w}_p, -\vec{w}_y\}$. The pitch and yaw DOFs are not used in this attack model because they require additional pixel information, making them unsuitable for an attack in the real world [14]. Therefore, PG attacks use four DOFs: the x-axis, y-axis, z-axis, and roll.

Each DOF causes various motion blur effects. The x- and y-axes cause linear motion blur, the z-axis causes radial motion blur, and roll causes rotational motion blur. These motion blurs can be generated simultaneously, creating a heterogeneous motion blur [14].

III. THREAT MODEL

Both sensors and object detection algorithms are essential in autonomous vehicles. The sensors consist of main and auxiliary sensors. Auxiliary sensors use compensation techniques to improve the quality of the image from the main sensor. Most of these auxiliary sensors are based on MEMS inertial sensors. However, as previously mentioned, MEMS inertial sensors are vulnerable to signal injection attacks. In addition, adversarial attacks on object algorithms have been continually reported.

Ji et al. [14] introduced AMple attacks, a new class of system level attacks resulting from the combination of maliciously crafted signal injection and adversarial attacks. Additionally, they proposed the first type of AMple attacks, the PG attacks, which controls the outputs of the auxiliary sensor. A PG attacks creates blurry images that cause malfunctions in object detection algorithms. Three types of PG attacks are HA, CA, and AA.

- HA: Although objects are detected accurately before an attack, the detection algorithm fails to detect identical objects after the attack.
- CA: After an attack, the object detection algorithm begins to recognize nonexistent objects as if they existed.
- AA: An AA causes an object to be miscategorized as another object. (For example, a fire hydrant might be detected as a person after the attack.)

Among these, an HA is significantly more disastrous than the other attacks because its attack success rate reaches 100% in many cases. Therefore, we propose a detection model against HA attacks. We introduce HA in Figure 1. Before HA is launched, the detection algorithm identifies the object (i.e., the car in the figure) with 92% of confidence score. However, the confidence score decreases drastically such that the car is not detected (i.e., the bounding box disappears), after HA is launched.

Ji et al. [14] proposed three assumptions to implement their attacks. First, they used black-box attacks because, in the real-world, network parameters and frameworks are not always available. However, they could know the confidence scores and classification results. Second, they assumed an adversary could obtain cameras and sensor awareness from the same models used in autonomous vehicles. Third,

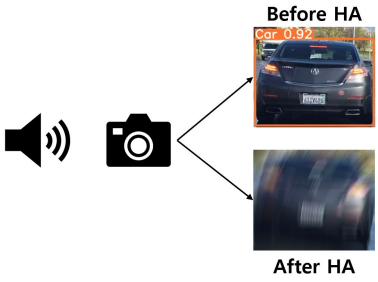


FIGURE 1. HA attacks.

they assumed the adversary had an acoustic attack capability. Thus, they introduced three acoustic signal injection scenarios: (1) installing signal generators along the side of the road, (2) attaching the signal generator to the surface of the target, and (3) controlling the damaged part of an onboard speaker. Furthermore, they experimented by installing a signal generator directly in front of the sensor in the car.

Gradient-free attack parameter optimization is used for PG attacks. Because we are focusing on HA attacks, we introduce HA optimization. A PG attacks is a black box attack because the adversary cannot obtain the network parameters and frameworks. Therefore, they denoted $Y = f(X)$, where X is the image as an input, Y is a prediction space, and f is a black-box object detection algorithm. Each prediction in Y is denoted as $Y_i \in Y$. Y_i is expressed as follows:

$$Y_i = (B_i, S_i^B, C_i, S_i^C),$$

where B_i is a bounding box, and C_i is a class of the prediction. ${S_i}^B$ and ${S_i}^C$ are the confidence scores corresponding to B_i and C_i. B is a blurred image affected by $\{\vec{a}_x, \vec{a}_y, \vec{a}_z, \vec{w}_r\}$, $B = X + \triangle$. The $\triangle$ is the degree of blur, so it can be considered an attack cost. They used objective functions to optimize each attack. The HA objective function is:

$$\begin{aligned} &w_1 S_i^B S_i^C + w_2 \|\triangle\|_p, \\ &|\vec{a}_x + \vec{a}_y + \vec{a}_z| < \xi_1, \\ &|\vec{w}_r| < \xi_2, \end{aligned}$$

where w_1 and w_2 are the weights that determine the leverage between the attack success rate and cost. ξ_1 and ξ_2 are the attack capability limitations of the accelerometer and gyroscope, respectively.

IV. METHODOLOGY AND DESIGN

In this section, we introduce the target environment that Priest aims to protect and the overall structure of our model.

Assumption on the Target Environment: In [14], [33], [38], [39], and [40], the attacks occurred within tens of centimeters or by attaching a signal generator to the target. The PG attacks also used an acoustic generator in front of the sensor. These attack methods are impractical because users or passengers could easily recognize the attacks. Therefore, to achieve more generality, we assume an external attack from outside the car.

However, existing attack methods are very difficult to exploit using an external attack in normal driving situations. Therefore, to use existing methods, we assume the external attacks occur when cars stop or slow down.

Priest Construction: Priest uses the similarity of pixel values between two adjacent images with a one frame difference. In Figure 2, we introduce three consecutive images to show the similarity of adjacent images. In the figure, t_0 and t indicate the current time and the inverse of sampling rate per image, respectively. Other than the car, which is located in the center of the images, buildings, traffic lights, sky, and other stop objects exist in almost the same positions in the images. Thus, significant portions of the images have nearly identical pixel values.

In Figure 3, we show the overall structure of Priest, which consists of six stages: Resize, Split, Sub, Pseudorandom, Add, and Judgment. The Resize function optionally shrinks the image size to decrease the computational cost. The Split function divides the resized image into three units, blue, green, and red, to implement a compelling detection model. The Sub function finds the different values on each pixel through the absolute value of subtracting the Split function image from the previous image. The Pseudorandom function optionally selects limited rows and columns to use in the Add function to reduce the run time. The Add function adds all or selected values of the Sub function to calculate the degree of similarity. The lower the output value is, the higher the similarity between the two images. Finally, the Judgment function determines whether an attack has occurred by identifying whether the degree of similarity is over the threshold value. In what follows, we describe the detailed specification of each algorithm.

Resize(*image*, *div*). Resize takes an *image* and *div* parameter as inputs and returns *re_image*, *resize.row*, and *resize.col* as outputs. The *image* is a current input image, and *div* is the parameter that determines the degree of shrinkage of the input image. The *re_image* is an image reduced by *div*, and *resize.row* and *resize.col* are the values of each row and column in the resized image. We use the resize function from OpenCV for generality. Priest has two techniques to reduce the computational effort. The first is the Resize function and the other is the Pseudorandom function. For example, if the user wants to reduce the cost by resizing the image, they pass a *div* value greater than one. Otherwise, if they do not want to use the Resize function, they set *div* to one and then use the Pseudorandom function later. We show the detection accuracy and run-time for the Resize function in Section V. The Resize function is outlined in Algorithm 1.

Split(*image*). Split takes an *image* from the Resize function as its input and returns *split_img_B*, *split_img_G*, and *split_img_R*, which are the image decomposed into blue, green, and red components, respectively, because our model detects attacks on each. That is, we use three criteria for detection. This function is outlined in Algorithm 2.

Sub(*image*, *image.before*). Sub takes an *image* and *image.before* as inputs and returns *sub.image*. The *image* and *image.before* are the current input image and the image one frame prior, respectively. The *sub.image* is the absolute value

FIGURE 2. Similarity of sequential images.

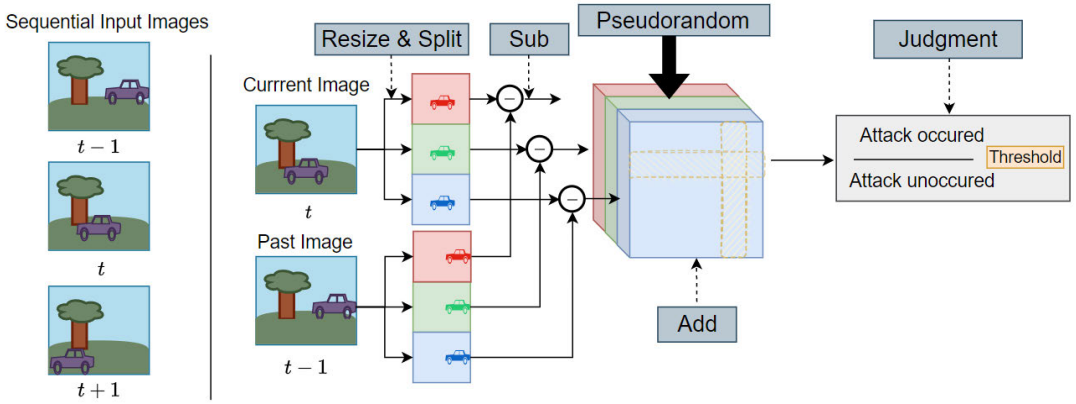


FIGURE 3. Overall structure of priest.

Algorithm 1 Resize Function

Input: *image*, *div*
Output: *re_image*, *resize.row*, *resize.col*
$resize.row \leftarrow int(image.row/div)$
$resize.col \leftarrow int(image.col/div)$
$re\_image \leftarrow resize(image, dsize = (resize.col, resize.row))$
return *re_image*, *resize.row*, *resize.col*

Algorithm 2 Split Function

Input: *image*
Output: *split_img_B*, *split_img_G*, *split_img_R*
$split\_img\_B \leftarrow image[0]$ $split\_img\_G \leftarrow image[1]$
$split\_img\_R \leftarrow image[2]$
return *split_img_B*, *split_img_G*, *split_img_R*

Algorithm 3 Sub Function

Input: *image*, *image.before*
Output: *sub.image*
$sub.image \leftarrow abs|image.before - image|$
return *sub.image*

of the subtraction of the *image* and *image.before*. In the Sub function, the subtraction of pixels corresponding to the same position in each input image is used. We use absolute values because if the signs were preserved, the values would be offset by the Add function, which adds all the values from the Sub function, and we could not obtain the exact similarity value. We show the Sub function in Algorithm 3.

Pseudorandom(*N*, *image.row*, *image.col*). This function takes *N*, *image.row*, and *image.col* as inputs and returns *select.row* and *select.col*. The *N* parameter determines the number of the pixel used in the detection model. The *image.row* and *image.col* parameters are the row and column sizes of the input image, respectively. The *select.row* and *select.col* parameters are randomly selected *N* values of rows and columns in the input image, regardless of order. As we previously mentioned, it is optional to use several random numbers (*N*) set by the user to reduce the computational effort. For example, the user can select values of 100, 120, 240, 720, and so forth. These randomly selected values are used to select pixels that correspond to particular rows and columns in the Add stage. The random rows and columns from the Pseudorandom function are generated at each given period (for example, every midnight or other specific time set by the user). Furthermore, the Pseudorandom function can defend against future optimization attacks because the adversary cannot know which pixels are chosen. Importantly, by using the Pseudorandom function, we can benefit from a reduced computational cost while maintaining the detection rate. We will discuss this advantage through experiments in Section V. We show this function in Algorithm 4.

Add(*sub.image*, *select*, *image.row*, *image.col*). Add takes *sub. image*, *select*, *image.row*, and *image.col* as inputs and returns *add. value*. The *select* parameter indicates whether the Add function uses the Pseudorandom function. If *select* is True, the Add function uses the limited pixels from

Algorithm 4 Pseudorandom Function

Input: N, $image.row$, $image.col$
Output: $select.row$, $select.col$
$select.row \leftarrow random.select(image.row, N)$
$select.col \leftarrow random.select(image.col, N)$
return $select.row$, $select.col$

Pseudorandom; otherwise, it uses all the pixels in the image of the Sub function's input. The *image.row* and *image.col* parameters are the rows and columns of the input images in the Sub functions or randomly selected rows and columns from the Pseudorandom function. The *add.value* is the sum of all the pixel values of the *sub.image* in the range of a given row and column. It indicates how different the pixel values of the two images are. We outline the Add function in Algorithm 5.

Algorithm 5 Add Function

Input: $sub.image$, $select$, $image.row$, $image.col$
Output: $add.value$
$add.value \leftarrow \perp$
if $select == True$ **then**
  for $x, y \leftarrow image.row, image.col$ **do**
    $add.value \leftarrow sub.image[x][y] + add.value$
  end
else
  for $x \leftarrow image.row$ **do**
    for $y \leftarrow image.col$ **do**
      $add.value$
      $\leftarrow sub.image[x][y] + add.value$
    end
  end
end
return $add.value$

Judgment(*add.value.B*, *add.value.G*, *add.value.R*). Judgment takes *add.value.B*, *add.value.G*, and *add.value.R* as inputs and returns *detect.value*. The *add.value.B*, *add.value.G*, and *add.value.R* parameters are the values from the Add function for attack detection in the blue, green, and red components, respectively. The *detect.value* indicates whether an attack has occurred. Only when all values are above the threshold value do we judge that the detection is successful because we want to implement a highly reliable detection model. For example, even if *add.value.B* and *add.value.R* exceed the set threshold value, if *add.value.G* does not exceed the threshold value, the model determines that no attack has been detected. We will propose the optimal threshold values by means of Algorithm 8 in Section V-D. We outline the Judgment function in Algorithm 6.

Furthermore, we propose the algorithm for overall usage in Algorithm 7. The algorithm takes the current *image* and *select* as inputs and returns *detect.value*, which indicates whether an attack has occurred. In the middle of the algorithm, if the user wants to use the Pseudorandom function, they must enter True in *select*, or the function cannot be used.

Algorithm 6 Judgment Function

Input: $add.value.B$, $add.value.G$, $add.value.R$
Output: $detect.value$
if $add.value.B \geq threshold$ &
  $add.value.G \geq threshold$ &
  $add.value.R \geq threshold$ **then**
  $detect.value \leftarrow True$
else
  $detect.value \leftarrow False$
end
return $detect.value$

V. EVALUATION

In this section, we evaluate our detection model. We used a 12th Gen Intel(R) Core(TM) i7-12700K CPU on a desktop.

Object Detection Algorithm: We validated our detection model using a pre-trained YOLO V4/V5 [3], [15] detection algorithm in the experiment. We considered six classes of interest: car, bus, person, truck, traffic light, and stop sign.

Dataset: We used the BDD100K [43] and KITTI [9] datasets to evaluate our model. BDD100K is the largest dataset with the most widely varying driving images. KITTI is a widely used dataset that offers many static and dynamic traffic scenarios. We selected images appropriate for our assumptions from the KITTI and BDD100K datasets. We used approximately 6300 images consisting of 100 images from each of 53 videos from BDD100K and approximately 100 images in each of 10 scenarios from KITTI. To the best of our understanding, KITTI contains significantly fewer data suitable for our assumptions than BDD100K, which is why there was a significant difference in the number of data elements taken from each dataset. We also address this matter in Section V-A using data based evidence.

Attack Simulation: We use the HA introduced in Section III. The images are divided into before and after the attack based on a randomly selected specific point in time. For example, we used 100 images per case, so we randomly selected from 2 to 99 images, excluding the first and last, because if we chose the first or last as an attack point, there would be no previous and next images. Then, based on the fixed value, it was assumed that there was no attack before the selected point and that the attack was continuously performed after the point. We show the process that divides the images into two sets based on the HA simulation attack point in Figure 4.

Selection Algorithm:

To set the optimal threshold values during the detection, we modified the Judgment algorithm, which was introduced in Algorithm 6. Because the *add.value* can exceed the threshold value not only at the point where the attack occurred but also at a point where it did not occur, we add a *correction*.

Algorithm 7 Overall Detection Algorithm

Input: *image*, *select*
Output: *detect.value re_image*, *reszie.row*, *resize.col*
← *Resize*(*image*)
split_img_B, *split_img_G*, *split_img_R*
← *Split*(*re_image*)
sub.image_B
← *Sub*(*split_img_B*, *split_img_B.before*)
sub.image_G
← *Sub*(*split_img_G*, *split_img_G.before*)
sub.image_R
← *Sub*(*split_img_R*, *split_img_R.before*)
select.row, *select.col*
← *Pseudorandom*(*N* , *image.row*, *image.col*)
if *select* == *True* **then**
  add.value_B ←
  Add(*sub.image_B*, *select*, *select.row*, *select.col*)
  add.value_G ←
  Add(*sub.image_G*, *select*, *select.row*, *select.col*)
  add.value_R ←
  Add(*sub.image_R*, *select*, *select.row*, *select.col*)
else
  add.value_B ←
  Add(*sub.image_B*, *select*, *resize.row*, *resize.col*)
  add.value_G ←
  Add(*sub.image_G*, *select*, *resize.row*, *resize.col*)
  add.value_R ←
  Add(*sub.image_R*, *select*, *resize.row*, *resize.col*)
end
detect.value ←
Judgment(*add.value_B*, *add.value_G*, *add.value_R*)
return *detect.value*

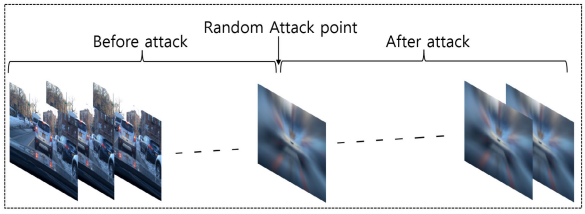


FIGURE 4. Attack point in threat model.

The value *correction* is used to eliminate the *false positive* value. The *correction* increases every time *add.value* exceeds the threshold value. However, if we only use the *correction* without considering other conditions, then it may not work as a remover accurately. To address this concern, we assume that, after the simulations of detection, the detection was successful when the threshold value meets two conditions below. Given this assumption, the *correction* number is set to one when a *detection.point* exceeding the threshold value is equal to a randomly determined attack point. As a result, we only use a *true positive* value as a detection rate in the rest of this section.

Algorithm 8 Selection Function

Input: *add.value.B*, *add.value.G*, *add.value.R*
Output: *detect.value*
correction ←⊥
detection.point ←⊥
for *i* ← 1, 2, . . . , *n* **do**
  if *add.value.B* ≥ *threshold* &
   add.value.G ≥ *threshold* &
   add.value.R ≥ *threshold* **then**
    detect.value ← *True*
    correction ← *correction* + 1
    detection.point ← *i*
  else
    detect.value ← *False*
  end
end
if *detection.point* == *attack.point* &
 correction == 1 **then**
  detect.value ← *True*
else
  detect.value ← *False*
end
return *detect.value*

Natural Difference Value of the Pixel: To analyze the experiments correctly, we should know the difference between pixel values without the attacks. We also consider moving objects such as birds, pedestrians, cars and bicycles, etc. Therefore, we performed the experiments by using 1280 × 720 BDD100K images on 53 videos. As a result, the average values are approximately 1,935,596.3, 1,820,963.7, and 1,914,817.6 in B, G, and R units, respectively.

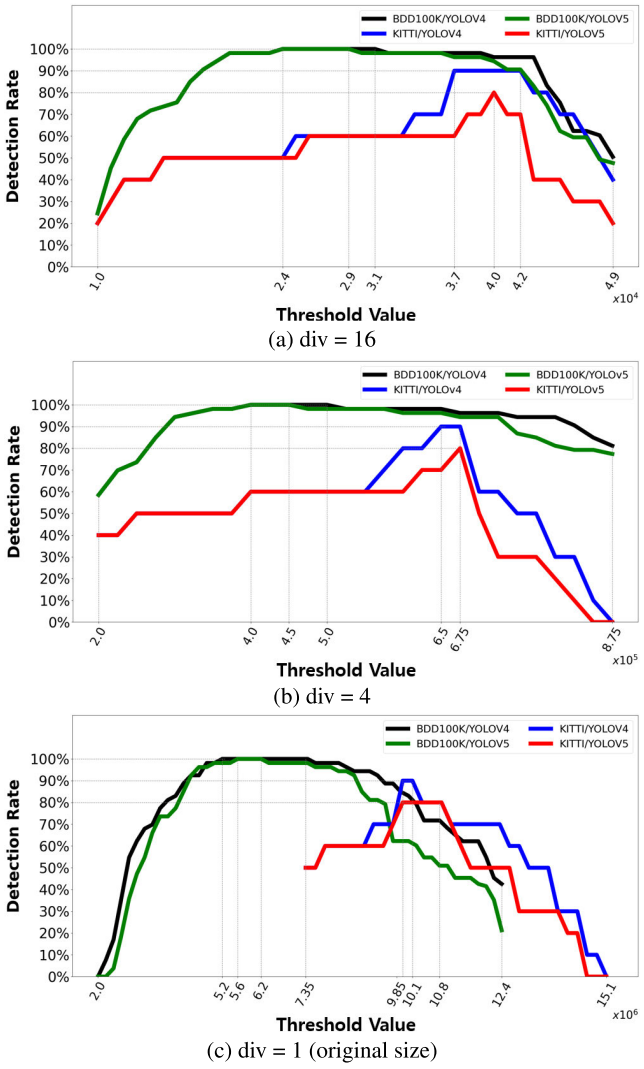
In the remainder of this section, we show our experimental results by (1) the detection rate by image size, (2) the detection rate using the Pseudorandom function, (3) the run-time of each stage and the total for all stages in the first and second situations, and (4) experiments performed in normal driving scenarios.

A. DETECTION RATE BY IMAGE SIZE

When the images are input, we set three folds dividing the image's rows and columns by 16, 4, and 1 (that is, we set div to 16, 4, and 1, respectively). When the value is 1, we use the original image. For example, the BDD100K dataset provides 1280 × 720 images, so we used 80 × 45, 320 × 180, and 1280 × 720 as the input image sizes.

Figure 5 shows the detection rate results according to the image size. The y-axis is the detection rate, and the x-axis is the threshold value. For YOLO V4, the highest detection rates were 100% and 90% for the BDD100K and KITTI datasets, respectively. For YOLO V5, the highest detection rates were 100% and 80% for the BDD100K and KITTI images, respectively.

The shape of the graph is similar to the normal distribution. The reason is as follows: (1) the threshold value, which is the



(a) div = 16

(b) div = 4

(c) div = 1 (original size)

FIGURE 5. Detection rate for image size.

x-axis value, is affected by the Add stage, (2) the output value of the Add function is large or small according to whether the difference in the value of each pixel is large or small, respectively, in the Sub stage, and (3) the difference in the pixel value is affected by the degree of blur in the image. Therefore, the x-axis is not only the number of image pixels but also the degree of the blur. If the threshold value is larger than the attack degree, the attack is not detected. Otherwise, if the threshold value is smaller than the attack degree, the attack and the small degree of blur of a general situation cannot be distinguished.

It is a natural question whether our model depends on the dataset and object detection algorithm because the detection accuracy is different. It does not for two reasons. First, these two datasets provide images at different frame rates. BDD100K provides 30 frames per seconds (FPS), whereas KITTI provides 20 FPS. When the FPS value is higher, more images are taken each second, resulting in greater similarity between adjacent images. Second, YOLO V5 is more vulnerable to PG attacks than YOLO V4. Ji et al. [14] showed that the attack success rate against YOLO V5 was higher than against YOLO V4 in CA and AA. The HA was excepted because the success accuracy of the HA was 100% against all detection algorithms. In fact, when we conducted our own experiment, we succeeded in attacking YOLO V5 even when the degree of attack was smaller than that of YOLO V4. Therefore, on BDD100K, the latter part of the graph is different because the necessary attack degree in YOLO V5 is smaller than in YOLO V4. On the KITTI dataset, the highest detection rates were different because not only are different degrees of attack needed but the frame rate is only 20 FPS, making it more difficult to distinguish between attacks and natural situations. In addition, in KITTI, all attacks are detected by our model, but the best detection rates were 90% and 80% for YOLO V4 and YOLO V5, respectively, because there was one case where the threshold value ranges did not overlap for each detection case. For example, when a second scenario was detected in the range from 37,000 to 40,000, a fifth scenario was detected in the range from 41,000 to 42,000. Therefore, if we advance Priest, this limitation must be eliminated. Furthermore, because most modern cameras provide more than 30 FPS, the detection results on BDD100K are more appropriate to the real world than KITTI.

One may wonder, if the detection rate is the true positive, why the lowest value of the threshold value does not have the highest detection rate. The first reason is that values close to the lowest one is also close to the difference between pixel values, so such values do not meet the condition, i.e., $Correction = 1$ in Algorithm 8. Secondly, because we utilized all B,G, and R units during the detection, there are some cases where two units meet the condition at the same time but one of them does not exceed threshold value.

Moreover, we provide evidence that the number of images in the KITTI dataset is sufficient to evaluate our detection model. KITTI has a frame rate of 1.5 times BDD100K. In Figure 5(b), the threshold range corresponding to a 100% detection rate in the 80×45 size of BDD100K for YOLO V4 is from 24,000 to 31,000. In Figures 5(b) and 5(c), the ranges are from 400,000 to 500,000 and from 5,200,000 to 7,350,000 for the 320×180 and 1280×720 sizes, respectively. Using KITTI, the ranges for the highest detection rate are [37,000, 42,000], [650,000, 675,000], and [9,850,000, 10,100,000] for image sizes of 80×45, 320×180, and 1280×720 in Figures 5(a), 5(b), and 5(c), respectively. The difference in the threshold's range corresponding to each image size is almost 1.5 times the difference; therefore, the number of data samples is sufficiently reliable. Furthermore, Ji et al. [14] used 200 images each from the BDD100K and KITTI datasets for experiments. Therefore, the size of our dataset is sufficiently reliable to verify our detection model.

B. DETECTION RATE USING THE PSEUDORANDOM FUNCTION

In this experiment, we used four values of N: 100, 120, 240, and 720. In Figure 6, for BDD100K, when the value of N is

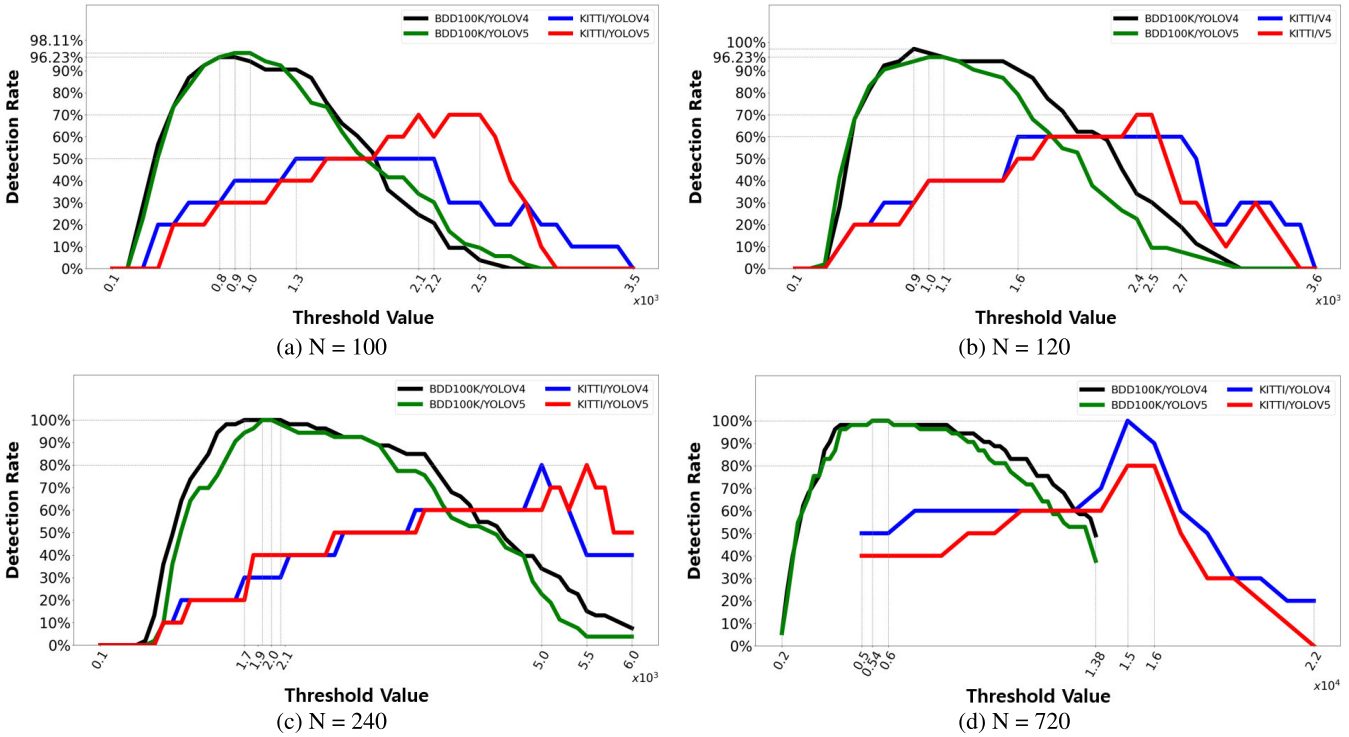


FIGURE 6. Detection rate for N with the pseudorandom function.

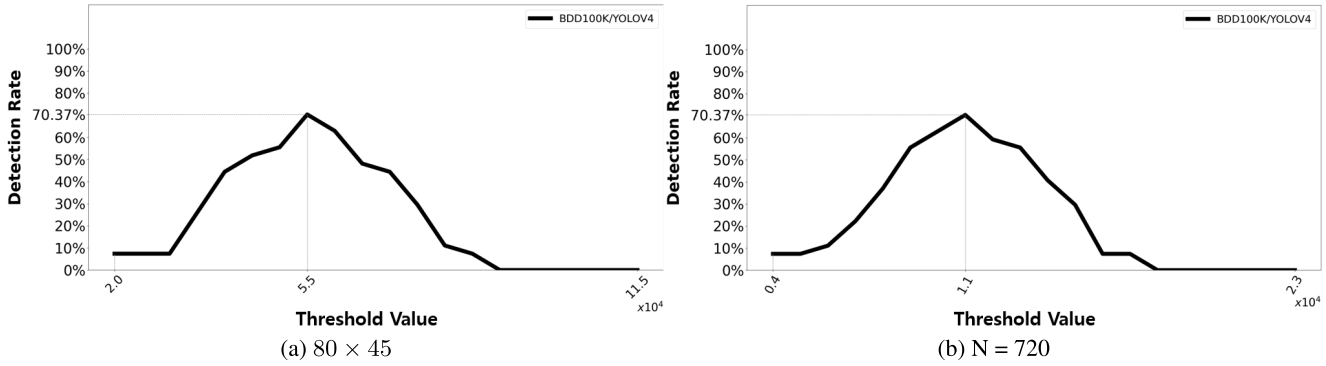


FIGURE 7. Detection rate in normal driving conditions.

over 240, the highest detection accuracy is equal to the results in Section V-A for YOLO V4 and V5. For KITTI, when the value of N is 720, the highest detection accuracy is equal to or higher than the results in Section V-A for YOLO V4 and V5. When the value of N increases, the performance of our model also increases on both datasets.

For YOLO V5, when the value of N is 100, the results are better than for YOLO V4 for BDD100K and KITTI. For YOLO V5, when the value of N is 120, the detection rate is higher than YOLO V4 for KITTI. These results differ from those in Section V-A in that the results for YOLO V4 are better than for YOLO V5. Furthermore, for KITTI, when N is 720, the highest detection rate is 100% for YOLO V4. These are the advantages of using the Pseudorandom function. If we use all pixels for detection, we must use pixels that have only slight variations. For example, in Figure 8, if we use all the pixels, we must use Case 1, where the pixel values before and after the attack are similar. As these values accumulate, it is difficult to distinguish an attack from a normal situation. Otherwise, like in Case 2, when we use the Pseudorandom function, we can select mostly pixels with more significant differences, making it easy to detect the attack.

In Figures 6(a), 6(b), and 6(c), the detection accuracy for KITTI in the latter part of the graphs is higher than the adjacent previous part, even though the threshold value is larger. This is because the sample size is too small to produce sufficiently reliable results.

In the experiments where N is less than 720, the performance difference between BDD100K and KITTI is because frame rates differ. A lower frame rate reduces the similarity,

TABLE 1. Computational cost by the size of the image.

Size of image	Pseudorandom	Resize (s)	Split (s)	Sub (s)	Add (s)	Judgment (s)	Total (s)
1280 × 720	N/A	0.00000	0.00000	0.00300	3.41299	0.00000	3.41599
320 × 180	N/A	0.00000	0.00000	0.00099	0.21399	0.00000	0.21499
80 × 45	N/A	0.00000	0.00000	0.00000	0.013999	0.00000	0.01399

TABLE 2. Computational cost by the number of pixels (N).

Size of image	Pseudorandom	Resize (s)	Split (s)	Sub (s)	Add (s)	Judgment (s)	Total (s)
1280 × 720	N = 120	0.00000	0.00000	0.00217	0.00049	0.00000	0.00266
1280 × 720	N = 240	0.00000	0.00000	0.00245	0.00098	0.00000	0.00343
1280 × 720	N = 720	0.00000	0.00000	0.00267	0.00296	0.00000	0.00563

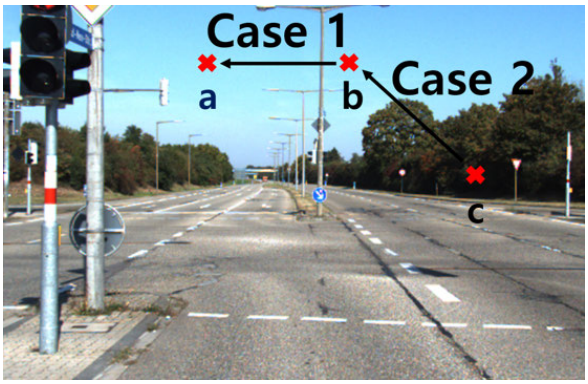


FIGURE 8. Difference in variation by selected pixel value.

making it difficult to detect attacks with a small number of pixels.

C. RUN-TIME

In this section, we introduce the computational cost of our detection model. We ran each experiment 100,000 times and calculated the average value. Table 1 shows the computational cost by the size of the image, and Table 2 shows the cost with the Pseudorandom function.

The KITTI dataset provides various image sizes, such as 1238 × 374, 1242 × 375, and 1224 × 370. In contrast, BDD100K provides consistently sized images, 1280 × 720. We used the 1280 × 720 size in this section, because we wanted to obtain consistent results and the 1280 × 720 size is larger than the other image sizes in the KITTI dataset.

Tables 1 and 2 show that the Add stage consumed most of the total time. The run-time of the Add stage is also proportional to the number of pixels: the Add stage run-time at the 1280 × 720 size was approximately 16 times greater than at the 320 × 180 size and 256 times greater than at the 80 × 45 size. Moreover, the run-time of the Add stage with Pseudorandom (N=720) is approximately three and six times greater than at N = 240 and 120, respectively.

As previously mentioned, the BDD100K dataset provides 30 FPS, and the KITTI dataset provides 20 FPS. If we convert FPS into time units, 30 FPS equals approximately 0.0333s, and 20 FPS equals 0.05s. Therefore, the results of the 80 × 45 size and all results using the Pseudorandom stage do not exceed the time criteria. Then, considering the performance, it is reasonable to use an image size of 80 × 45 and Pseudorandom (N=720). In addition, in both cases, it can be used at 60 FPS, ensuring a more accurate detection rate.

D. DETERMINE THE OPTIMAL THRESHOLD VALUE

In this section, we propose the optimal threshold values. There are three conditions to consider. Firstly, candidates are values which have the highest detection rate in each experiment. Secondly, among such candidates, only the biggest value pass the first criterion. Lastly, the computational cost of candidate should be with in each FPS (for example, 0.0333 seconds). Because if the performance exceeds the fps, it is not applicable to actual driving conditions. According to our experiment as shown in Figure 5(a), 29,000 is the optimal threshold value by above three conditions for BDD100K/YOLOV5.

E. NORMAL DRIVING SITUATIONS

We also performed experiments in *normal driving* situations, where the *normal driving* refers to a circumstance in which we do not focus on vehicles that are standing or stationary. We used (1) image sizes of 80 × 45 with the Resize function and (2) image sizes of 1280 × 720 with the Pseudorandom function (N=720) for YOLO V4 because the two cases are the best scenarios in both detection performance and computational cost. In this experiment, we used a total of 2300 images, 100 from each of 23 videos, on the BDD100K dataset because 30 FPS is closer to data from camera sensors used in the real world. Figure 7 shows that our detection model had a 70.37% detection rate in Cases (1) and (2). Furthermore, in this experiment, the detection model successfully detected all attacks, but the success rate was only 70% because there were some parts where the range and value of the threshold value that succeeded in detecting the attack did not overlap. Therefore, we predict that the detection model will perform better under any driving conditions as the detection technology continues to improve.

F. BENCHMARK TEST

We propose a benchmark test of Priest, the squeezing method [41] and the state-of-the-art de-blurring method [29]. Firstly, we compare our method to the de-blurring method.

As we said earlier, the de-blurring method needs 1.4s for 1280×720 image size. This is not suitable in the real-world driving situations. Because the FPS of sensors, such as 30FPS ($\approx$ 0.033s) is typically more higher than 1FPS ($\approx$ 1s). By contrast, our method requires approximately 178FPS($\approx$ 0.0056s). Secondly, we perform the benchmark test for feature squeezing method [41]. In [41], they showed an outstanding detection accuracy. Specifically, the accuracy reaches 98.8% and 87.5% in MNIST and CIFAR-10, respectively. However, this method is highly dependent to datasets. Furthermore, it is hard to adapt this in the driving conditions because the result of model under CIFAR-10, which may reflect the real world more, is significantly lower than that under MNIST. Conversely, in Priest, we only used the real world driving condition dataset for evaluations and preserve high accuracy.

VI. RELATED WORK

In this section, we introduce previous studies on (1) attack models for MEMS inertial sensors, (2) defense methods for MEMS inertial sensors, (3) defense models against adversarial attacks, and (4) detection models for adversarial attacks.

A. ATTACK METHODS FOR MEMS INERTIAL SENSORS

Extensive control systems depend on feedback from MEMS sensors for critical decisions. MEMS gyroscopes and accelerometers detect changes in mass to measure inertial stimuli. However, sensing mass is affected by frequencies close to the natural frequency of the hardware. That is, MEMS inertial sensors are known to be vulnerable to error signal injection attacks. Specifically, recent studies have shown that injecting a resonant acoustic signal into MEMS sensors can cause incorrect operation [33], [38], [39], [40].

Son et al. [33] analyzed resonant frequencies of MEMS gyroscope sensors. They then injected signals into drones that had the same resonant frequencies as the previously analyzed sensors. As a result, the drones could not operate normally. Trippel et al. [38] introduced an output control attack method that exploited modulated signals. Specifically, they showed that it was possible to control the output of the MEMS accelerometer precisely. Wang et al. [40] further showed how to affect virtual reality devices, drones, and other device. Finally, Tu et al. [39] proposed a side-switching attack that could manipulate the output of MEMS gyroscopes and accelerometers.

B. DEFENSE METHODS FOR MEMS INERTIAL SENSORS

1) BEST SAMPLING

To detect signal-injection attacks, Shoukry et al. [32] introduced PyCRA (physical challenge response authentication), which exploits two types of signals: challenge and response. First, a challenge signal is randomly selected and transmitted to measure the physical response of a measurable entity. After transmission, PyCRA authenticates the response signal.

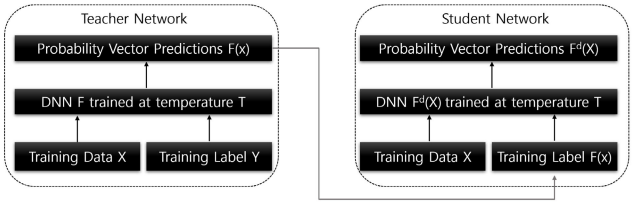


FIGURE 9. Structure of defensive distillation.

Specifically, if a challenge signal is transmitted without a specific range when an attack does not occur, the specific range of the response signal must also be empty. However, if an attack occurs, the specific range of the response signal is not empty. This process can detect attacks without degrading the sensor performance [10]. However, Shin et al. [31] showed that PyCRA actually requires extensive computational effort, preventing its use in practice. Therefore, we introduce a detection model that has a negligible cost.

2) SENSOR FUSION

One detection method is sensor fusion. Sensor fusion uses sensors of various types or multiple identical sensors. Because an adversary must corrupt all sensors, it makes injection attacks harder [4], [31], [39]. However, Nashimoto et al. [25] proposed that the fusion method using Kalman filters could be circumvented. To rectify the problem, Tharayil et al. [37] presented an enhanced algorithm that uses a mathematical relationship between a gyroscope and a magnetometer. However, a vulnerability remains [10]. Moreover, the sensor fusion method is costly because many sensors are required [10]. Therefore, designing a secure and cost-effective detection scheme is highly desirable.

C. DEFENSE MODELS AGAINST ADVERSARIAL ATTACKS

In this section, we introduce defense methods against adversarial attacks. Furthermore, we introduce the de-blurring methods proposed as a countermeasure by Ji et al. [14].

1) DEFENSIVE DISTILLATION

Defensive distillation [13], [27] was originally introduced as a method to deliver knowledge of a complicated model to a simpler model. Parpernot et al. [27] proposed a defensive distillation algorithm that applies previous knowledge distillation algorithms for the model's robustness to adversarial attacks. As shown in Figure 9, they first trained the model in the teacher network. The teacher network then output probability vectors using the follow equation.

$$F(X) = \frac{e^{z_i(x)/T}}{\sum_{i=0}^{N-1} e^{z_i(x)/T}} \tag{1}$$

Above equation looks similar to softmax but has Distillation Temperature T. If the value of T increases, a uniform probability distribution can be obtained. Conversely, as the value of T decreases, the result of Equation 1 becomes like a discrete probability distribution. This output is used as a label in the student network. By using probability vectors

as labels, one can prevent overfitting, thus deriving a robust network by means of better generalization. The defensive distillation algorithm increased defense success rates from 1.55% to 13.79% and from 0.39% to 2.56% on the MNIST and the CIFAR-10 datasets, respectively. However, it has a problem in that the defense success rate of the CIFAR-10 dataset, which is more relevant to the real world, is lower than the results on the MNIST dataset. Because a probability vector includes information about the relationship between data, the MNIST dataset, which has simpler data, produces a higher performance than the CIFAR-10 dataset, which has more complex data. For example, MNIST contains data about numbers. Numbers 1 and 7 have something in common with stick. That is, these features help defensive distillation's performance on the MNIST dataset. Conversely, the CIFAR-10 dataset has more complex data, such as vehicle, cat, dog, and others; thus, these data have fewer similar properties than MNIST, so they perform poorly. Therefore, defensive distillation is not suitable for use in actual driving situations.

2) ADVERSARIAL TRAINING

This process uses adversarial images other than the original images to train ML models. Because adversarial training requires an increased training dataset size, a brute-force strategy is needed [1]. This strategy increases the robustness of the machine learning model to adversarial attacks [12], [30]. Miyao et al. [21] proposed virtual adversarial training, which smoothes the output distributions of the network. Zheng et al. [44] proposed a stability training method that improves the robustness to minor distortions. Furthermore, Madry et al. [20] presented a powerful adversarial training method, a first-order method called PGD training. This method can defend against white-box attacks on the MNIST and CIFAR-10 datasets with probabilities of 85% and 46%, respectively, and against black-box attacks with probabilities of 95% and 64%, respectively. This method is robust to adversarial attacks because the defense accuracy with black-box attacks performs well relative to white-box attacks. However, the accuracy against both white-box and black-box attacks using MNIST is much higher than using CIFAR-10. This can be seen as a dependence on the dataset, because the CIFAR-10 dataset is more similar to the real world than the MNIST dataset. Therefore, we propose a detection model validated using data from real-world driving conditions.

3) DATA-DRIVEN METHODS

We also introduce the de-blurring method proposed as a countermeasure in [14]. Several studies have used deep learning de-blurring methods [11], [24], [29], [35]. Sun et al. [35] measured the probability of specific blur kernels based on convolutional neural networks (CNNs). Gong et al. [11] introduced a deeper-CNN. This deeper-CNN method evaluates the motion flow with no post-processing. However, it has a drawback in an application domain because it is designed for linear kernels.

Furthermore, several methods have been based on end-to-end training [24], [26]. Noroozi et al. [26] used skip connections to generate residual images of blurred images without the characteristics of blur kernels. Nah et al. [24] presented a model with 40 convolutional layers that can be applied at every scale. Moreover, it has 120 layers to recover clean images. Nonetheless, it incurs a high computational cost because the method uses a multi-scale strategy to obtain a sharp image.

4) SPATIALLY VARYING NETWORKS

De-blurring methods have been developed based on spatially varying networks [18], [28]. Ren et al. [28] proposed a Shepard interpolation layer that performs recovery and creates super-resolution. A predefined mask determines whether a pixel is used for interpolation in the spatially varying scene. Liu et al. [18] developed a spatially varying RNN trained by a CNN to come close to low-level filters. This method reduces the number of kernels and channels of models because long distance propagation by the RNN is used for image information delivery. Then, this method is followed in [7] by expanding 1D linear propagation to 2D spatial diffusion in an end-to-end manner. Ren et al. [29] used both 1D and 2D RNNs for de-blurring. They noted that 1D RNNs are underutilized in relationships between neighboring pixels because 1D RNNs connect pixels only from the previous row or column. Specifically, they used 2D RNNs to obtain a large receptive field and learn the denser propagation. As a result, this method reduces computational costs and increases performance over previous studies. When using 1280×720 images, the peak signal-to-noise ratio and structural similarity index maps are 31.7603 and 0.9241, respectively, and the run-time is 1.4s. Despite these enhancements, the run-time is too long to be used for autonomous driving. Because the BDD100K dataset supports 30FPS videos, the run-time is only suitable if it is within appropriately 0.0333s per image. Therefore, we propose a verifiable technique with a run-time of less than approximately 0.0333s.

D. DETECTION MODELS FOR ADVERSARIAL ATTACKS

1) SafetyNet

Lu et al. [19] hypothesized that patterns of rectified linear unit (ReLU) activations in the last stages of networks are different between clean and adversarial images. They appended a radial basis function support vector machine (SVM) classifier to the target model. This SVM network uses a discrete code computed in the last ReLU of the network. By comparing the discrete codes computed by the test and training sets, they could detect adversarial attacks [22].

2) FEATURE SQUEEZING

Xu et al. [42] introduced the squeezing method, which adds two external models to detect adversarial attacks. These reduced the color bit depth of each pixel and carried out spatial smoothing on the image. An original and a

squeezed image are used for prediction in the target model. If the predicted values of the original and squeezed images are significantly different, an attack has occurred. Finally, Xu et al. [41] showed that the feature squeezing method is also effective against C&W attacks [16].

These aforementioned detection methods seem significantly safe from adversarial attacks. However, these methods [19], [22], [41], [42] are detection techniques that focus on static perturbation images that are not even identified by the human eye. The real world is dynamic. When a person or thing moves, the image automatically has more blur than the perturbations considered in the above methods. Furthermore, the blur naturally generated when things move can induce malfunctions in the object detection algorithm. Therefore, it is difficult to use in the dynamic real world.

However, that does not mean a defense model that uses sequential data does not exist. Lin et al. [17] introduced a detection model for adversarial attacks that uses the distribution of sequential data in the game and allows us to perform normal tasks by predicting the next behavior, even while under attack. However, it requires 180 frames for detection and foresight. Because the BDD100K dataset provides 30 FPS and the KITTI dataset provides 20 FPS, it must be assumed that no attack occurs within at least six seconds when using the above detection model. The assumption that there have been no attacks during six seconds is too strong. Therefore, we introduce a detection model that uses only two consecutive images.

VII. CONCLUSION

HA has a 100% attack success rate. It is a very effective attack, and a solution is urgently needed. Therefore, in this paper, we introduce a detection model called Priest. This model uses the similarity between pixels in consecutive images, and it guarantees, on average, a 99% detection rate with a run-time of 0.00563s on the BDD100K dataset, which is close to real-world conditions. As a result, we propose that if we use (1) an image size of 80×45 or (2) a 1280×720 image with the Pseudorandom function (N=720), the model ensures a significant detection accuracy and can be applied at 30 FPS or even 60 FPS. Therefore, our detection model can be fully used in the real world.

REFERENCES

[1] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14410–14430, 2018.

[2] D. Bereska, K. Daniec, S. Fras, K. Jedrasiak, M. Malinowski, and A. Nawrat, "System for multi-axial mechanical stabilization of digital camera," in *Vision Based Systemsfor UAV Applications*. Cham, Switzerland: Springer, 2013, p. 177.

[3] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[4] C. Bolton, S. Rampazzi, C. Li, A. Kwong, W. Xu, and K. Fu, "Blue note: How intentional acoustic interference damages availability and integrity in hard disk drives and operating systems," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 1048–1062.

[5] B. Cardani, "Optical image stabilization for digital cameras," *IEEE Control Syst. Mag.*, vol. 26, no. 2, pp. 21–22, Apr. 2006.

[6] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 39–57.

[7] X. Cheng, P. Wang, and R. Yang, "Learning depth with convolutional spatial propagation network," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 10, pp. 2361–2379, Oct. 2020.

[8] S. Erturk, "Digital image stabilization with sub-image phase correlation based global motion estimation," *IEEE Trans. Consum. Electron.*, vol. 49, no. 4, pp. 1320–1325, Nov. 2003.

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1231–1237, Sep. 2013.

[10] I. Giechaskiel and K. Rasmussen, "Taxonomy and challenges of out-of-band signal injection attacks and defenses," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 645–670, 1st Quart., 2020.

[11] D. Gong, J. Yang, L. Liu, Y. Zhang, I. Reid, C. Shen, A. Van Den Hengel, and Q. Shi, "From motion blur to motion flow: A deep learning solution for removing heterogeneous motion blur," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3806–3815.

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

[13] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[14] X. Ji, Y. Cheng, Y. Zhang, K. Wang, C. Yan, W. Xu, and K. Fu, "Poltergeist: Acoustic adversarial machine learning against cameras and computer vision," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 160–175.

[15] G. Jocher. (Oct. 2020). *Ultralytics/YOLOV5: V3.1—Bug Fixes and Performance Improvements Version v3.1*. [Online]. Available: https://doi.org/10.5281/zenodo.4154370

[16] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial Intelligence Safety and Security*. Boca Raton, FL, USA: Chapman & Hall/CRC, 2018, pp. 99–112.

[17] Y.-C. Lin, M.-Y. Liu, M. Sun, and J.-B. Huang, "Detecting adversarial attacks on neural network policies with visual foresight," 2017, *arXiv:1710.00814*.

[18] S. Liu, J. Pan, and M.-H. Yang, "Learning recursive filters for low-level vision via a hybrid neural network," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 560–576.

[19] J. Lu, T. Issaranon, and D. Forsyth, "SafetyNet: Detecting and rejecting adversarial examples robustly," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 446–454.

[20] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2017, *arXiv:1706.06083*.

[21] T. Miyato, A. M. Dai, and I. Goodfellow, "Adversarial training methods for semi-supervised text classification," 2016, *arXiv:1605.07725*.

[22] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A simple and accurate method to fool deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2574–2582.

[23] C. Morimoto and R. Chellappa, "Fast electronic digital image stabilization," in *Proc. 13th Int. Conf. Pattern Recognit.*, 1996, pp. 284–288.

[24] S. Nah, T. H. Kim, and K. M. Lee, "Deep multi-scale convolutional neural network for dynamic scene deblurring," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 257–265.

[25] S. Nashimoto, D. Suzuki, T. Sugawara, and K. Sakiyama, "Sensor CONfusion: Defeating Kalman filter in signal injection attack," in *Proc. Asia Conf. Comput. Commun. Secur.*, May 2018, pp. 511–524.

[26] M. Noroozi, P. Chandramouli, and P. Favaro, "Motion deblurring in the wild," in *Proc. German Conf. Pattern Recognit.* Cham, Switzerland: Springer, 2017, pp. 65–77.

[27] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 582–597.

[28] J. S. Ren, L. Xu, Q. Yan, and W. Sun, "Shepard convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 901–909.

[29] W. Ren, J. Zhang, J. Pan, S. Liu, J. S. Ren, J. Du, X. Cao, and M.-H. Yang, "Deblurring dynamic scenes via spatially varying recurrent neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 8, pp. 3974–3987, Aug. 2022.

[30] S. Sankaranarayanan, A. Jain, R. Chellappa, and S. N. Lim, "Regularizing deep networks using efficient layerwise adversarial training," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, 2018, pp. 4008–4015.

[31] H. Shin, Y. Son, Y. Park, Y. Kwon, and Y. Kim, "Sampling race: Bypassing timing-based analog active sensor spoofing detection on analog-digital systems," in *Proc. 10th USENIX Workshop Offensive Technol. (WOOT)*, 2016, pp. 1–11.

[32] Y. Shoukry, P. Martin, Y. Yona, S. Diggavi, and M. Srivastava, "PyCRA: Physical challenge-response authentication for active sensors under spoofing attacks," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 1004–1015.
[33] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in *Proc. 24th USENIX Secur. Symp. (USENIX Security)*, 2015, pp. 881–896.
[34] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Trans. Evol. Comput.*, vol. 23, no. 5, pp. 828–841, Oct. 2019.
[35] J. Sun, W. Cao, Z. Xu, and J. Ponce, "Learning a convolutional neural network for non-uniform motion blur removal," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 769–777.
[36] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*.
[37] K. S. Tharayil, B. Farshteindiker, S. Eyal, N. Hasidim, R. Hershkovitz, S. Houri, I. Yoffe, M. Oren, and Y. Oren, "Sensor defense in-software (SDI): Practical software based detection of spoofing attacks on position sensors," *Eng. Appl. Artif. Intell.*, vol. 95, Oct. 2020, Art. no. 103904.
[38] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, "WALNUT: Waging doubt on the integrity of MEMS accelerometers with acoustic injection attacks," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroSP)*, Apr. 2017, pp. 3–18.
[39] Y. Tu, Z. Lin, I. Lee, and X. Hei, "Injected and delivered: Fabricating implicit control over actuation systems by spoofing inertial sensors," in *Proc. 27th USENIX Secur. Symp. (USENIX Security)*, 2018, pp. 1545–1562.
[40] Z. Wang, K. Wang, B. Yang, S. Li, and A. Pan, "Sonic gun to smart devices: Your devices lose control under ultrasound/sound," in *Proc. Black Hat USA*, 2017, pp. 1–50.
[41] W. Xu, D. Evans, and Y. Qi, "Feature squeezing mitigates and detects Carlini/Wagner adversarial examples," 2017, *arXiv:1705.10686*.
[42] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," 2017, *arXiv:1704.01155*.
[43] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "BDD100K: A diverse driving dataset for heterogeneous multitask learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2633–2642.
[44] S. Zheng, Y. Song, T. Leung, and I. Goodfellow, "Improving the robustness of deep neural networks via stability training," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4480–4488.

JAEHWAN PARK is currently pursuing the bachelor's degree with the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, South Korea. His research interests include information security and applied cryptography.

CHANGHEE HAHN received the B.S. and M.S. degrees in computer science from Chung-Ang University, Seoul, South Korea, in 2014 and 2016, respectively, and the Ph.D. degree from the Department of Computer Science and Engineering, College of Informatics, Korea University, South Korea, in 2020. He was with Korea University as a Postdoctoral Researcher, from 2020 to 2021. He is currently an Assistant Professor with the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul. His research interests include information security and cloud computing security.