Real-time attack detection on robot cameras: A self-driving car application

Sofiane Lagraa, Maxime Cailac, Sean Rivera, Frédéric Beck, Radu State

Real-time attack detection on robot cameras: A self-driving car application

Sofiane Lagraa[†], Maxime Cailac[‡], Sean Rivera[†], Frédéric Beck[‡], and Radu State[†]
[†] SnT, University of Luxembourg
firstname.lastname@uni.lu
[‡] Inria Nancy-Grand Est, 615 rue du Jardin Botanique, 54600 Villers-les-Nancy, France
firstname.lastname@inria.fr

***Abstract*—The Robot Operating System (ROS) are being deployed for multiple life critical activities such as self-driving cars, drones, and industries. However, the security has been persistently neglected, especially the image flows incoming from camera robots. In this paper, we perform a structured security assessment of robot cameras using ROS. We points out a relevant number of security flaws that can be used to take over the flows incoming from the robot cameras. Furthermore, we propose an intrusion detection system to detect abnormal flows. Our defense approach is based on images comparisons and unsupervised anomaly detection method. We experiment our approach on robot cameras embedded on a self-driving car.**

I. Introduction

The Robotic Operating System (ROS) is a framework for robotic system development which is popular in both academic and industrial contexts. In particular, ROS is the industry leading operating system for robotic systems, being used in a multitude of industries. An example of a product where ROS is used is to build a self-driving and autonomous vehicles [11]. Unfortunately, ROS does not provide any security features [14]. The communication paradigm behind ROS is that of a Publish-Subscribe model, where a master keeps track of the state of the system while applications called nodes can directly interact with each other, all communication being conducted through a middleware layer relying ultimately on an unsecured network. The only secure approach is to isolate the ROS nodes or the use of virtual private networks (VPN) for the communication of robots in a private network. Recent work [10] highlighted a number of security threats against ROS.

In this paper, we focus on the ROS camera node which is a vulnerable node. In fact, the attacker can be in the middle between the camera node and the processing node such as pedestrians recognition node. Thus, the attacker can modify flows i.e. images or inject fake images in order to misrecognize objects by processing nodes. The attacks are not necessary sophisticated. Unfortunately, no existing tool allowing to detect such attacks and protect camera flows in ROS. Thus, we focus on attack detection on ROS camera and take an intrusion prevention approach. Our solution addresses attacks from compromised camera nodes, and detects certain classes of attacks. Our specific contributions are:

- We propose several adversarial models capturing capabilities of attacks we consider when designing attack detection mechanism for ROS camera node. The different attacks are perturbations on images performed by an attacker in order to mis-recognize objects. For example the misrecognition of pedestrians or traffic lights.
- We propose an anomaly detection method for detecting abnormal images incoming from camera flows targeted and modified by an attacker.
- We provide experimental results on realistic environments including a self-driving vehicle running on ROS. Our experimental results demonstrate the effectiveness of the detection of attacks on different proposed scenarios.

The rest of the paper is organized as follows. Section II briefly presents the background on ROS, and related work on security of ROS. Section III describes several adversarial models capturing capabilities of attackers we consider when designing anomaly detection mechanisms for flows of cameras in ROS. Section IV describes our proposed approach for anomaly detection. We provide experiments in Section V. Finally, Section VII draws conclusions and discusses future work.

II. Background

In this section, we describe the background and existing works related in ROS and the security of ROS.

A. ROS

The Robot Operating System (ROS) [1] is a meta-operating system framework for developing robotic systems. In essence, it provides an application framework to applications consisting of independent computing processes called *nodes*, with the help of a master node acting as a global `namespace`, a parameter server acting as a repository of globally shared data, and a *middleware layer* providing a consistent set of interfaces for software development and hardware. They facilitate the communication between nodes based on two abstractions: topics and services. All processes run on top of a UNIX operating system. ROS provides tools and libraries for obtaining, building, writing, and running code across multiple computers. A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model.

ROS Master: The ROS Master provides name registration. The master node is the main communication hub that tracks all the offered topics and services and maintains the internal state representation of the communication system. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

ROS Nodes: ROS uses nodes as a high-level abstraction for defining individual processes running within its framework. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls the wheel motors, one node performs localization, and so on.

ROS Topics: Topics are the primary form of communication for the ROS system. Messages are routed via a transport system with publish/subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message.

A node that is interested in a specific kind of data will subscribe to the appropriate topic. Thus, a single topic may have multiple concurrent publishers and subscribers, and a single node may publish and/or subscribe to multiple topics.

ROS Messages. Nodes communicate with each other by passing messages. Messages are simply a data structure that contains the typed field, which can hold a set of data and that can be sent to another node. There are standard primitive types such as integer, floating point, Boolean. The developer can also build its own message types using these standard types.

ROS Services. The request/reply interactions is done via services, which are defined by a pair of message structures: one for the request and one for the reply.

For example, as shown in Figure 1, the ”Camera” node sends messages to the ”Images” topic. The messages in the topic are received by the ”Storage” node and the ”Processing” node. The ”Storage” node depends on the underlying Linux file system to provide access to the storage location. The publisher-subscription model is designed to be modular at a fine-grained scale and is suitable for distributed systems.

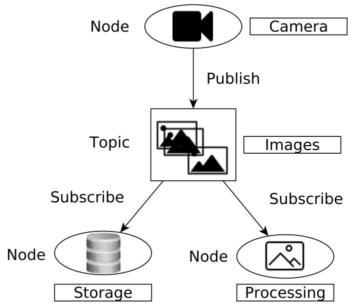


Fig. 1: Example of ROS

B. Security of ROS

The ROS master node plays an important and critical role in ROS. Without the master, nodes would not be able to find each other, exchange messages, or invoke services. In the event that the master node is compromised, it is functionally identical to the whole system being compromised.

For instance, a ROS node may publish data for an arbitrary topic without any prior authorization. A node may be misused to inject data into an application in order to disturb its process or operation. For example, an attack may inject a fake movement command causing unpredictable damages [4].

In addition, any node in ROS may subscribe to any topic within the application. After that, it will receive any data that is published for that topic. In fact, there is no protection between node communications, allowing an attacker to create numerous problems from learning confidential information, to injecting, modifying, replaying packets. A recent effort into addressing some of the security concerns for ROS is SROS, an experimental security suite designed to harden ROS systems against several common classes of attacks [2]. However, SROS still has several limitations. It also does not protect against compromised nodes and the system cannot react to compromised nodes and that at any level of system complexity. Additionally, the SROS system would not be able to adequately react to the malicious node. Attacks shown include: Remote control without authentication [7], Unauthorized publish/subscription, message modification, and publish message rejection [14], Denial of service [14]. In [7], the authors highlighted real-world attacks against commercial humanoid robots running on ROS. They highlighted a relevant number of security flaws that can be used to take over and command the robot. Several recommendations have been provided by the authors on how these issues could be fixed. In [14], it was shown how an attacker could stop the components that controls the robot such as legs, sensors, camera, in order to immobilize the robot. In this paper, we focus on the vulnerability and attacks that could be performed on a camera embedded in a robot using ROS. We describe several adversarial models capturing capabilities of attackers we consider when designing defense mechanisms for a camera flow in ROS.

Our work differs from the existing works. In fact, we propose an overlay for a processing node to detect and block adversarial examples. Our solution is more generic for any type of processing. The processing node can use any kind of processing, deep learning, reinforcement learning algorithm. We propose a cybersecurity solution like a firewall for adversarial models capturing attacks for ROS node image processing nodes. In our case, we assume that the processing nodes in ROS is robust against attacks. The processing nodes can use deep learning algorithms, machine learning,...etc. Our goal is to protect the processing node against attacks by detecting them.

III. Adversarial model

In this section, we describe several adversarial models capturing capabilities of attacks we consider when designing defense mechanisms for ROS camera nodes.

A. Assumptions

For different attacks performed on ROS camera nodes, we enumerate the following assumptions:

- The master node is assumed to be like a certificate authority (CA) and is the root of trust for the system. It is assumed that the master node is not compromised.
- The parameter server is assumed to have protection mechanisms from malicious alteration for the parameters stored on the parameter server.
- We assume that the ROS middleware is secure from exploits.
- All ROS systems run on top of the Linux operating system. We assume that the underlying Linux system is secure and that best practices are taken in the design of the system such that a compromised ROS camera node cannot compromise the Linux system.
- We assume that the attacker shares a network. Thus, they can conduct any of the attacks.

B. ROS camera node - Attacker Model

Remember that in ROS all communications are unprotected and as no communication is encrypted. Thus, the attacker may perform the following models:

- Remote supervision: The attacker observes remotely the camera flow for spying the people. In fact, the attacker may create a fake node in order to intercept the published flow from the camera.
- Fake flow modification: After the creation of the fake node, the attacker may alter the images by modifying the original ones. Thus, the attacker creates fake images in order to disrupt the processing nodes. For example, given a machine learning algorithm within a processing node, then a machine learning algorithm misclassifies a fake image.
- Fake flow injection: Without a fake node creation, the attacker may create a subscription to a processing node by injecting a fake images.

Figure 2 illustrates an example of an attack model on ROS camera node. The attack model allows an attacker to observe, inject, intercept, or modify communications between ROS nodes specially between ROS camera node and a fake node created by an attacker. We see that the attacker can replace the existing path from the camera to the processing node by a fake node and producing an alteration and modification of images.

In this paper, we focus on the different type of injections, alterations of flow images incoming from ROS camera node.

Figure 3 shows an example of a sample of images extracted from the camera embedded in a self-driving car and perturbed with the attack scenarios proposed in Table I. In Figure 3, each line of continuous images over time is a scenario of an attack and the perturbed image is the 6^{th} image, except the scenario 5 and 6 where the perturbed images are from the 4^{th} image to the 8^{th} image.

Scenario ID	Type of attacks (Perturbation)	Image
1	New image insertion (black image)	6
2	New image insertion (white image)	6
3	Blurry image	6
4	Image from the past	6
5	Successive same images	[4,8]
6	Flooding black images	[4,8]
7	Blurry image	6
8	Modified image (Adding a black rectangle)	6
9	Modified image (Adding a black rectangle on the traffic lights)	6
10	Modified image (replacing a red color by a green color in the traffic lights)	6

TABLE I: Scenario of attacks on images from a camera embedded on a self-driving car.

Table I shows the type of attacks or perturbations on images. The attacks are:

- New image insertion: the attacker injects a fake image (black, white or an image not related to current flow) in the streaming images like scenario 1 and 2. In addition, the attacker can take an image fro the past and inject it the top of the streaming like a scenario 4.
- Blurred image: the attacker alters an image by blurring it. We can find a high blurring in scenario 3 and small blurring in scenario 7.
- Flooding attack: the attacker can inject successive same images by flooding a processing node. Scenario 5 and 6 present flooding same images, and flooding black images, respectively.
- Image perturbation: the attacker can modify images by performing a small and sophisticated perturbations causing classification errors. The perturbation consist in adding black stickers on objects such as a car, or traffic lights in a image. The objective of this attack is to hide objects in a image.

We show that the attacker can physically modify objects using low-cost techniques to reliably cause classification errors in ROS processing node. For example, in Figure 4c, the attack causes a classifier to interpret a subtly-modified physical red color in the traffic lights as a green color. Specifically, our final form of perturbation is a set of black stickers attached to a physical objects such as a car (Figure 4a) or traffic lights (Figure 4b).

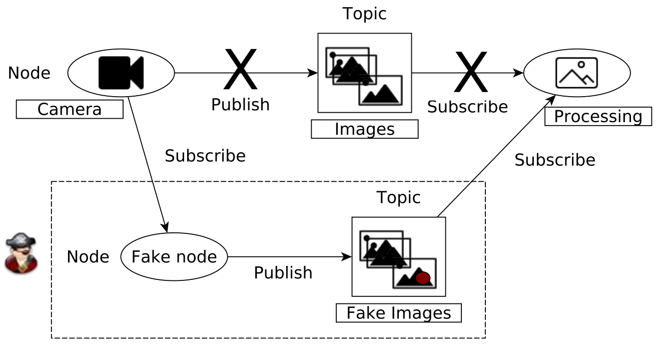


Fig. 2: Attack model on ROS camera node

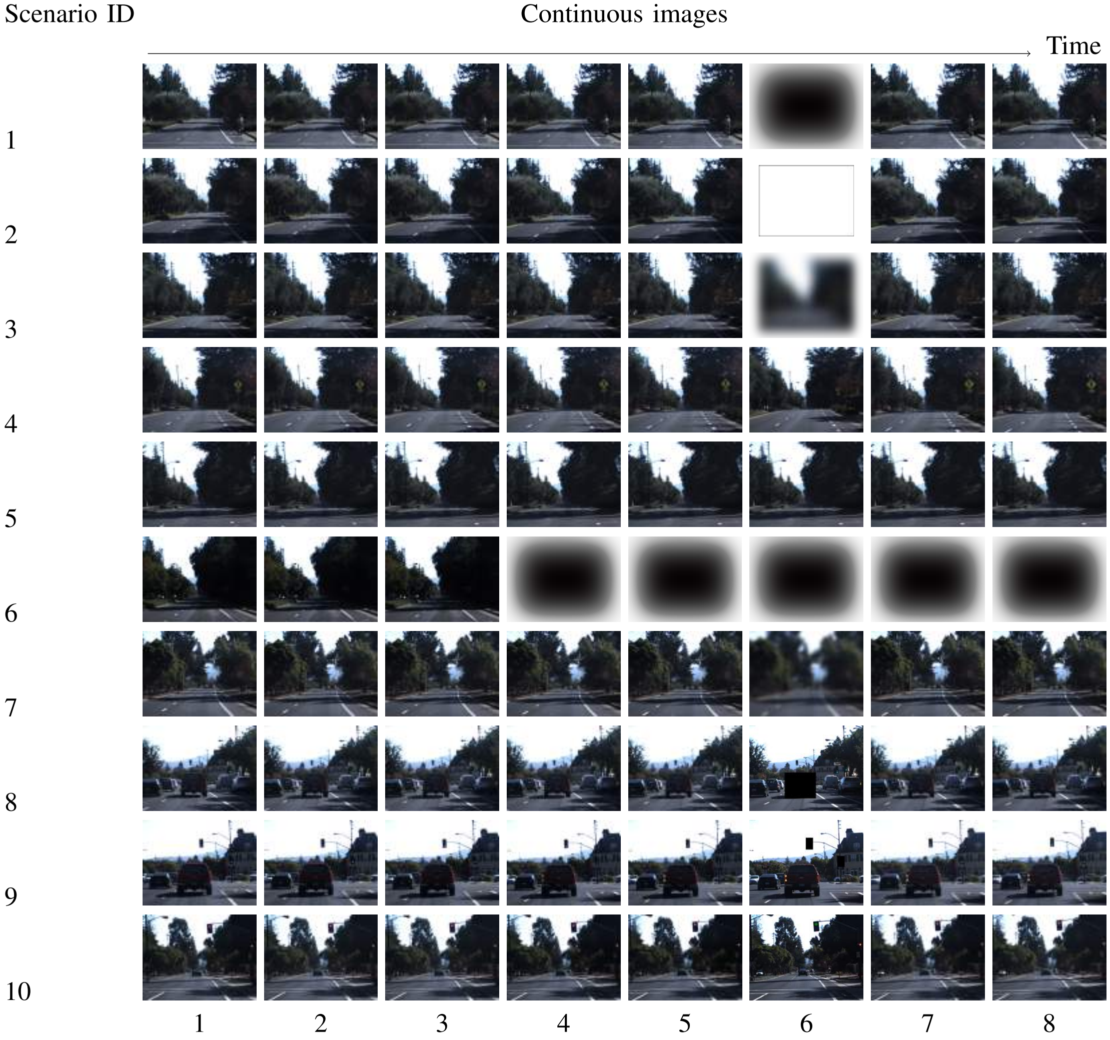


Fig. 3: Example of targeted attacks on a camera embedded on a self-driving car.

IV. Proposed approach

Figure 5 represents an architecture for detecting adversarial attacks on streaming images from a camera with a use case for Robot Operating System (ROS). The architecture is composed of two part components: ROS component providing streaming of images from a camera in ROS and defense tool for detecting abnormal images on steaming images comparisons.

A. Streaming images comparison

Suppose that the camera C is affected by a perturbation p on frame image I by a software. In this case, the output image O of the camera $O = C^p(I)$ may differ from the original image I.

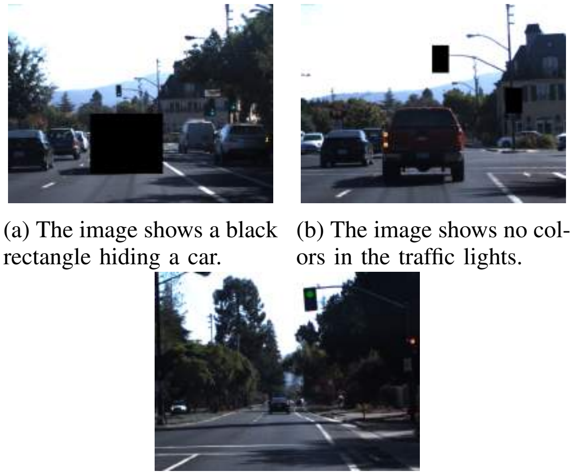
Definition 1 (Frame sequence): Let $F = \{f_1, f_2, \ldots, f_n\}$ be the set of all frames incoming from a camera. A frame sequence is a stream ordered sequence of frames ordered according to their timestamp $\{(f_1, t_1), (f_2, t_2), \ldots, (f_n, t_m)\}$ from camera C, where frame $f_i \in F, i \in \{1 \ldots n\}$ is a frame at time $t_j, j \in \{1 \ldots m\}$ and $t_i < t_{i+1}$.

For example, Figure 6 represents a sequence of 5 frames according to the time.

The task of successive frames comparison is then to identify a shift in the frame sequence over the time. This can be a challenging task since the key difficulty is to detect intrinsic unknown perturbations of frames. In order to express the property of changes between successive frames over time, we define a function to compare two successive frames over time called *time series of similarities* .

Definition 2 (Time series of similarities): Time series of similarities is a totally order collection of similarities values between two successive frames. Time series of similarities TS_sim of frame sequence FS is a triple $(F, \leq, Sim)$ where F is a set of frames, $\leq$ is a total order relation on F i.e $x \leq y$ or $y \leq x$ for all $x, y \in F$. Sim is a function that computes the difference between each successive frames $(f_i, t_i) \rightarrow (f_{i+1}, t_{i+1})$, it returns the similarity $Sim(f_i, f_{i+1})$ between the frame f_i and f_{i+1}.

Let the similarity between frames calculated by the similarity measure $Sim(f_{i-1}, f_i)$. Suppose that imaging systems



(a) The image shows a black rectangle hiding a car.

(b) The image shows no colors in the traffic lights.

(c) The image shows the change of a color in the traffic lights by replacing the red color by the green one.

Fig. 4: Sophisticated perturbations on images

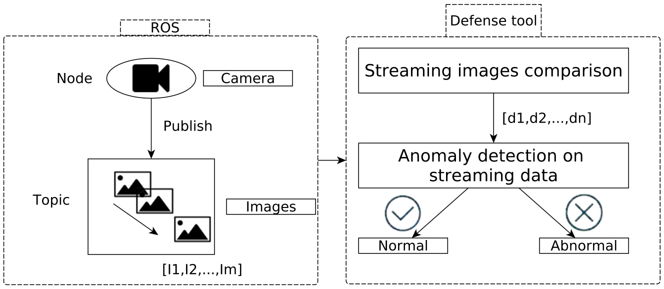


Fig. 5: Architecture for adversarial examples attacks detection on streaming images from a camera: Robot Operating System (ROS) use case

is affected by a perturbation. In this case, the current frame f_i may differ from the previous one f_{i-1}. The objective of similarity computation is to decide whether f_i is close enough to f_{i-1}. Several metrics to calculate the distance between two images exist. Since each metrics has advantages and disadvantages, we use and experiment six distance similarity metrics. The comparison of streaming frames allows to distinguish between the normal and abnormal frame. In Section IV-B, we provide more details how to detect abnormal images from steaming images.

1) Structural Similarity Image Metric (SSIM): The Structural Similarity (SSIM) measures loss of structure in the image as opposed to just any deviation with respect to reference. Loss of image structure measured locally through luminance similarity, contrast similarity, structural similarity. It performs average of local measure across the image. The SSIM index is based on the computation of three terms, namely the *luminance comparison* $l(x, y)$, the *contrast comparison* $c(x, y)$ and the *structural comparison* $s(x, y)$. The overall index is a multiplicative combination of the three comparisons of two images [19].

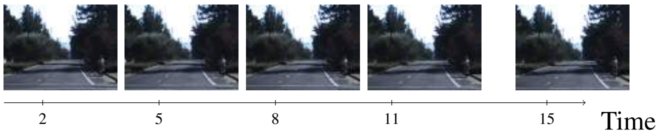


Fig. 6: A stream ordered frame sequence

$$SSIM(f_{i-1}, f_i) = (l(f_{i-1}, f_i))^{\alpha} \cdot (c(f_{i-1}, f_i))^{\beta} \cdot (s(f_{i-1}, f_i))^{\gamma} \tag{1}$$

where α, β, and γ are weight factors. The values $l(f_{i-1}, f_i)$, $c(f_{i-1}, f_i)$ and $s(f_{i-1}, f_i)$ are defined based on the luminance (represented by mean intensity).

$$l(f_{i-1}, f_i) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2}$$

$$c(f_{i-1}, f_i) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{3}$$

$$s(f_{i-1}, f_i) = \frac{\sigma_{xy} + C_3}{\sigma_x \cdot \sigma_y + C_3} \tag{4}$$

where C_1, C_2 and C_3 are constants. In practice, SSIM may be calculated for each color component separately, or an image may be divided into different blocs and SSIM can be calculated on these blocs and averaged. SSIM is a symmetric similarity i.e. $SSIM(f_{i-1}, f_i) = SSIM(f_i, f_{i-1})$ and bounded between 0 and 1. $SSIM(f_{i-1}, f_i) = 1$ if and only if images are identical.

2) DSSIM: In [13], the authors proposed a dissimilarity measure for computing the difference between images. The difference has been measured using structural dissimilarity (DSSIM) [23] using structural dissimilarity. DSSIM is a distance measure derived from structural similarity (SSIM). It keeps the advantages of SSIM, but is more similar to distance measures (nonnegative, reflexive, symmetric) [13]:

$$DSSIM(f_{i-1}, f_i) = \frac{1}{SSIM(f_{i-1}, f_i)} - 1 \tag{5}$$

DSSIM, like SSIM is defined for grayscale images only. The greater values of SSIM and DSSIM refer to greater similarity between images.

3) Mean Squared Error (MSE): MSE is the simplest, and most widely used full-reference image quality measurement. This metric measures quantify the difference between images through computing the average of errors. Is describes the degree of similarity. It stands for the mean squared difference between the images and can be defined as follows [18]:

$$MSE(f_{i-1}, f_i) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2 \tag{6}$$

Where x_i and y_i are the values of the *i*th pixels in f_{i-1}, and f_i, respectively of the compared images (f_{i-1}, f_i).

The ability of MSE is to capture perceptually relevant differences, such as high texture detail. MSE is simple and inexpensive to compute [18]. It satisfies the interpretation of similarity, i.e., non-negative, identity, symmetry,,and triangular inequality [8], [18]. Lower value of MSE means a good value, i.e., higher similarity of the reference image and distorted image. MSE words satisfactorily when distortion is mainly caused by contamination of additive noise [18].

4) Histogram-based image comparisons: Histogram comparison called sim_hist is the most commonly used technique in measuring color similarity of images. An image's color histogram is constructed by counting the number of pixels for each color in the image. More formally, the color histogram is defined by: the tuple $h(A, B, C)$ represent the dimensions of the 3 dimensional color spaces, either R, G, B or H, S, V. An image can be represented by a color histogram. In order to express the similarity of two histograms, a measure distance is employed to compute the similarity between histograms. The Euclidean distance between two color histograms h and g is given by:

$$d(h, g) = \sqrt{\sum_{A}\sum_{B}\sum_{C}(h(a, b, c) - g(a, b, c))^2} \qquad (7)$$

Where h, and g are the color histogram of the image frames f_{i-1} and f_i. a, b, c are color components in RGB color space for the histograms $a = r$, $b = g$, $c = b$.

In this formula there is only comparison between the identical bins in the respective histograms.

Histograms are good solutions for the problem of a large number of images storage which takes a large amount of space [6]. Using histograms the amount of memory required to store an image can be reduced.

5) Vectors-based image comparisons: We propose vectors-based similarity for image comparisons called sim_vect. Images pixels are treated as vectors, Given f_{i-1} and f_i two images and their two vectors of pixels $V(f_{i-1}) = [f_{i-1_1}, ..., f_{i-1_n}]$ and $V(f_i) = [f_{i_1}, ..., f_{i_n}]$, respectively. The vectors $V(f_{i-1})$ and $V(f_i)$ are normalized between 0 and 1 into $V'(f_{i-1}) = [f'_{i-1_1}, ..., f'_{i-1_n}]$ and $V'(f_i) = [f'_{i_1}, ..., f'_{i_n}]$. We use the dot product for measuring the similarity between images:

$$sim\_vect = V'(f_{i-1}) \cdot V'(f_i) = \sum_{i=1}^{n} f'_{i-1_j} \cdot f'_{i_j} \qquad (8)$$

Thus, the result is a float between 0 and 1 that indicated the percent similarity of the two images.

6) Grayscale and hashing-based image comparisons: We use the similarity developed by the authors in [20] which combine grayscale and hashing methods. In this paper, we call the similarity sim_gray. In the first step, the similarity method convert the image to grayscale. It means removing the color information and retain the brightness information. In the second step, it calculates the mean brightness of each block for scaling the image into the grid of blocks. Thus, the result is one pixel per block. In the third step, it determines the median value of the previously calculated mean values. In the fourth step, it sets the hash bit for each block according to whether its mean value is above the median or not. In the final step, the Hamming distance is employed, which counts the number of non-matching bits.

B. Anomaly detection

Each similarity metric computes its own similarity of successive images, in this section, we want to discover the abnormal values from each similarity metric. We say a frame i is normal if the similarity with the previous frame $i-1$ is close and no drift from the rest of similarity values. Otherwise, there is a suspect and a potential perturbation on the current frame. In order to detect such frames from the similarity values, we use the tool developed in [16]. In [16], the authors proposed an online outlier detection algorithm based on extreme value theory for univariate numerical streaming time series data. They proposed a library called *libspot* [1] to detect anomalies in stationary data streams as well as drifting data streams considering concept drift and to find distribution-independent bounds on the rate of extreme (large) values. The goal of the extreme value theory is to find the law of extreme events. For example the law of the daily maximum of temperature, or the law of the monthly maximal tide height. The idea of *libspot* library is to provide dynamic threshold with a probabilistic meaning based on standard model solution. *libspot* has two inputs: The main parameter is q which is the probability of abnormal events (between 0 and 1) and the second parameter is the number of initial observations to perform calibration on the threshold. It has an alert as an output.

The advantage of *libspot* is that that does not require to hand-set thresholds and makes no assumption on the distribution. The drawback is a parameter the probability of potential outliers it means controlling the number of false positives. In addition, the size of the initial batch to be used for the distribution must be large in order to avoid failing because of a lack of peaks to perform.

V. Experimental results

A. Experimental setup

We present the computational experiments and the performance conducted by running of each image comparisons combined with anomaly detection algorithm. We evaluate the accuracy over different types of scenarios of attacks. The experiments are performed on a 2.20 GHz Intel(R) Core(TM) i7-2720QM CPU 64bits laptop with 4 GB of RAM running Linux. All programs were implemented in Python. For image comparisons, we use PIL and *numpy* libraries. The Python Imaging Library (PIL): it supports many file formats, and provides powerful image processing and graphics capabilities. *numpy*: it supports large and multi-dimensional arrays and matrices.

[1] https://github.com/asiffer/libspot

For anomaly detection, we use *libspot* [2] library proposed in [16] to detect anomalies in stationary data streams. We measure the impact of the two input parameters of *libspot* on the detection of anomalies; the probability of abnormal events q, and the number of initial observations.

The goal of the experiments is to find a compromise between the image comparisons algorithms and the anomaly detection algorithm. We want to compare the performance of image comparisons algorithms in order to find the best one and discover which algorithm is well adapted to a specific scenarios of attacks.

B. Dataset

Different videos recorded by cameras embedded on a self driving car are used for evaluating the anomaly detection algorithms, and the videos are taken on different daytime scenes, including highway, urban common road, urban narrow road, multiple vehicles,...etc.

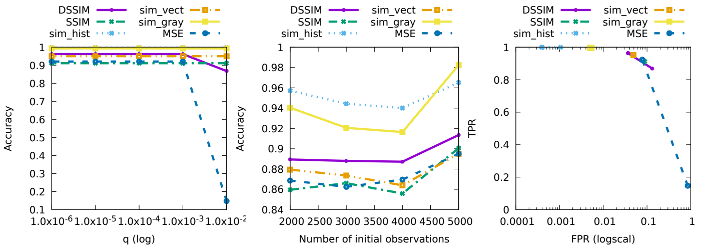
We use a driving data which is publicly dataset of real world publicly available and provided by Udacity [3]. The duration of the video is 10 minutes and 9 seconds which contains more than 15000 frames. We construct different images perturbations. The images records include both normal images, and attacked images with different perturbations. The dataset contains 10 separate scenarios whose general characteristics are described in Table I. The entire dataset is already classified normal or abnormal based on the scenarios in Table I. Figure 3 shows an example of a sample of images extracted from the camera in the self-driving car and perturbed with the scenarios. We provide both a ground-truth and an evaluation software in order to evaluate other images similarity and anomaly detection methods against the current data.

C. Results

The performance of each anomaly detection method is calculated based on the outliers detection accuracy, sensitivity, specificity, precision, and processing time. The outliers detection accuracy, true positive rate (TPR) and false positive rate (FPR) are calculated by using the following formula: $Accuracy = \dfrac{TP+TN}{TP+TN+FP+FN}$, $FPR = \dfrac{TP}{TP+FN}$, $FPR = \dfrac{FP}{FP+TN}$. Where:

- True Positive (TP): Observations where the actual and detected images were attack.
- True Negative (TN): Observations where the actual and detected images weren't attack.
- False Positive (FP): Observations where the actual images weren't attack but detected to be attack.
- False Negative (FN): Observations where the actual images were attack but weren't detected to be attack.

Figure 7a show the accuracy according to the variation of the main parameter q as a false-positive regulator. The number of initial observations is set to 5000 observation similarity values incoming from each similarity metric. We see that the accuracy of abnormal value of similarity metrics is very high using DSSIM, SSIM, sim_vect, sim_hist, and sim_gray expect MSE when the probability to have an abnormal value is high i.e $q = 10^{-2}$. The accuracy of different similarity metrics is between 0.85 and 0.96. Thus, the main parameter q has no influence on the accuracy of detection. In contract, Figure 7b shows the accuracy of detection according to the number of initial observation used for training. In this figure, we set $q = 10^{-3}$. We see that the number of initial observations useful for the detection of abnormal similarity values varies in each number of initial observations. Globally, when the number of initial observations grows then the accuracy grows too.



(a) Accurcy Vs. q (b) Accurcy Vs. init (c) TPR Vs. FPR

Fig. 7: Comparaison results

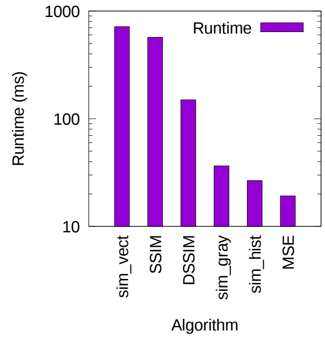


Fig. 8: Runtime of similarity metrics.

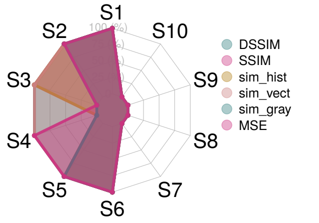


Fig. 9: Accuracy of detection per scenario S_i of Table I.

We study the impact of the main parameter q on different similarity metrics. On Figure 7c, the curve shows the effect of q on the False Positive rate (FPR). Values of q between 10^{-2} and 10^{-6} allow to have a high True Positive rate (TPR) while keeping a low FPR. Almost of similarity metrics provide high TPR and low FPR expect MSE when the probability $q = 10^{-2}$.

From these experiments, we conclude that the best similarity metric combined with the anomaly detection values (*libspot*) is sim_hist. Figure 8 shows the runtime of the similarity metrics. Thus, sim_hist is in *top-2* of the best algorithms which provides us better compromise between the runtime and the accuracy. Figure 9 shows the accuracy of detection per scenario described in Table I. We see that all the scenarios from S_1 to S_6 are detected by *libspot* on almost similarity metrics but the scenarios from S_7 to S_{10} are not detected due

[2] https://github.com/asiffer/libspot

[3] https://github.com/udacity/self-driving-car

to the small changes on a figure i.e. changing the traffic lights or hiding a car. Thus, the detection of scenarios from S_7 to S_{10} is considered as a future work of this paper.

VI. Related work

Adversarial Examples

For the image classification problem, in [17], the authors generated small perturbations on the images and fooled deep neural networks. These misclassified samples were named as *Adversarial Examples*. For instance, an adversary can construct physical adversarial examples and confuse autonomous vehicles by manipulating the stop sign in a traffic sign recognition system [12], [5] or removing the segmentation of pedestrians in an object recognition system [21].

Adversarial Examples have been proposed on deep learning algorithms due to their vulnerability to the inputs. Sometimes, adversarial examples are imperceptible to human and can easily fool deep neural networks in the testing or/and deploying stage. The vulnerabilities to adversarial examples become one of the major risks for applying and deploying deep neural networks in safety-critical environments such as a self-driving vehicle. In [22], the authors highlight two types of countermeasures of defense strategies for adversarial examples on deep learning: 1) *reactive*: detect adversarial examples after deep neural networks are built; 2) *proactive*: make deep neural networks more robust before adversaries generate adversarial examples. All the existing works develop more robust deep learning algorithms in order to detect adversarial examples using different techniques. In our work, we propose to protect the input flows of machine learning and deep neural networks algorithms. It allows the machine learnings tools to focus on their primary tasks i.e. classification, clustering,...etc.

Anomaly Detection. Anomaly Detection field has been extensively studied and many overviews can be found in [3] [9]. Recent work on anomaly detection has focused on statistical properties of ”normal” data to identify these anomalies, such as works in [15], which uses Benford’s Law to identify anomalies in social networks, and [16], which uses Extreme Value Theory to detect anomalies. But the problem of anomaly detection from steaming data without any supervision or input parameter settings remains a challenge.

VII. Conclusion and Future work

In this paper, we have proposed an anomaly detection method for detecting abnormal camera flows targeted and disturbed by an attacker in ROS. We also propose several adversarial models capturing capabilities of attacks on ROS camera node. The different attacks are perturbations on camera flow performed by an attacker in order to mis-recognize objects when using machine learning algorithms. Our experimental results, over real data highlight ability of our method to detect abnormal image frames incoming from camera flows. In the future, we plan to extend the intrusion detection system on different ROS sensors stream anomaly detection and support for machine learning.

References

[1] Ros technical overview. http://wiki.ros.org/ROS/TechnicalOverview. Accessed: 2018-09-11.

[2] Sros. http://wiki.ros.org/SROS. Accessed: 2018-09-11.

[3] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[4] B. Dieber, B. Breiling, S. Taurer, S. Kacianka, S. Rass, and P. Schartner. Security for the robot operating system. *Robot. Auton. Syst.*, 98:192–203, Dec. 2017.

[5] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song. Robust physical-world attacks on machine learning models. *CoRR*, abs/1707.08945, 2017.

[6] S. N. Ferdaus, A. Vardy, G. Mann, and R. Gosine. Comparing global measures of image similarity for use in topological localization of mobile robots. In *2008 Canadian Conference on Electrical and Computer Engineering*, 2008.

[7] A. Giaretta, M. D. Donno, and N. Dragoni. Adding salt to pepper: A structured security assessment over a humanoid robot. *CoRR*, abs/1805.04101, 2018.

[8] K. Gu, G. Zhai, X. Yang, and W. Zhang. An improved full-reference image quality metric based on structure compensation. In *Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference*, pages 1–6, 2012.

[9] V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, Oct 2004.

[10] S.-Y. Jeong, I.-J. Choi, Y.-J. Kim, Y.-M. Shin, J.-H. Han, G.-H. Jung, and K.-G. Kim. A study on ros vulnerabilities and countermeasure. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, HRI ’17, pages 147–148, New York, NY, USA, 2017. ACM.

[11] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296, 2018.

[12] A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.

[13] A. Loza, L. Mihaylova, N. Canagarajah, and D. Bull. Structural similarity-based object tracking in video sequences. In *2006 9th International Conference on Information Fusion*, pages 1–6, 2006.

[14] F. Martn, E. Soriano, and J. M. Caas. Quantitative analysis of security in distributed robotic frameworks. *Robotics and Autonomous Systems*, 100:95 – 107, 2018.

[15] S. Maurus and C. Plant. Let’s see your digits: Anomalous-state detection using benford’s law. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 977–986, 2017.

[16] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’17, pages 1067–1075, 2017.

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

[18] Z. Wang and A. C. Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE Signal Processing Magazine*, 26(1):98–117, 2009.

[19] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[20] C. Winter, M. Steinebach, and Y. Yannikos. Fast indexing strategies for robust image hashes. *Digital Investigation*, 11:S27 – S35, 2014. Proceedings of the First Annual DFRWS Europe.

[21] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. L. Yuille. Adversarial examples for semantic segmentation and object detection. *CoRR*, abs/1703.08603, 2017.

[22] X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li. Adversarial examples: Attacks and defenses for deep learning. *CoRR*, abs/1712.07107, 2017.