Intrusion detection on robot cameras using spatio-temporal autoencoders: A self-driving car application

Faouzi Amrouche, Sofiane Lagraa, Raphaël Frank, and Radu State
Interdisciplinary Centre for Security, Reliability and Trust (SnT)
University of Luxembourg, 29 Avenue J.F Kennedy, L-1855 Luxembourg
firstname.lastname@uni.lu

***Abstract*—Robot Operating System (ROS) is becoming more and more important and is used widely by developers and researchers in various domains. One of the most important fields where it is being used is the self-driving cars industry. However, this framework is far from being totally secure, and the existing security breaches do not have robust solutions. In this paper we focus on the camera vulnerabilities, as it is often the most important source for the environment discovery and the decision-making process. We propose an unsupervised anomaly detection tool for detecting suspicious frames incoming from camera flows. Our solution is based on spatio-temporal autoencoders used to truthfully reconstruct the camera frames and detect abnormal ones by measuring the difference with the input. We test our approach on a real-word dataset, i.e. flows coming from embedded cameras of self-driving cars. Our solution outperforms the existing works on different scenarios.**

I. Introduction

Most roboticists nowadays are using the Robot Operating System (ROS) [1] for the development of robot applications. It is used in a multitude of real world settings, including military projects and commercial robots [2]. It is also being used for self-driving cars development [3]. However, it is still vulnerable from the security aspect. Recent studies have shown many flaws in ROS [4] that can lead to irreversible damages if used with safety critical applications. A self-driving car is a good example for this, where an abrupt change in the trajectory could endanger the safety of the passengers and pedestrians.

A. Problem statement

ROS is based on a *publisher-subscriber* architecture. It is composed of nodes (e.g. camera, processing) that communicate through data publication into corresponding *topics* (e.g. images) by *publisher* nodes (e.g. camera). This data can be used by *subscriber* nodes (e.g. processing node). Any node can subscribe to any topic within an application in order to receive all the data that is published in it. Practically, there is no protection for nodes communication, which allows attackers to realize various scenarios; from learning confidential information, to injecting, modifying, and replaying packets. For instance, the attacker can play the Man-In-The-Middle role between the camera node and processing node, and is able to alter images or inject fake ones in order to induce a misrecognition of objects by the processing node.

Given a sequence of timestamped images, the problem is to detect, in real time, the abnormal images injected by an attacker over a streaming flow.

B. Related works

Autonomous cars with ROS: In [5], the authors addressed the autonomous cars application of ROS, and provided a security assessment. They presented an attack detection solution based on images similarity, which could detect some attack scenarios.

Autonomous systems with ROS : In [6], the authors assessed the security flaws of ROS regarding nodes communication for autonomous systems. They presented a solution based on cypher encryption that provides promising results. In [7], the authors discovered vulnerabilities and attacks against commercial humanoid robots running on ROS. They highlighted a relevant number of security flaws that can be used to take over the robot. In [8], the authors assessed the security of autonomous vehicules and presented several attacks that target the steering, braking, and acceleration of the vehicule, but also attacks against the computer vision system.

C. Contribution

To the best of our knowledge, very few works have targeted the vulnerabilities of ROS applied to self-driving cars, and the existing solutions do not offer optimal performances. Thus, we address in our paper the specific vulnerabilities related to the robot's camera, taking the case of an autonomous vehicle. We present a novel method for real-time anomaly detection on camera flows, which can be embedded directly on the car's system. Our solution is based on *autoencoders* used in an unsupervised way to reconstruct the camera flows. The experimental results are obtained on a previously published dataset [5]. We show that our solution outperforms existing works.

The paper is organized as follows. Section II describes adversarial models of attacks we consider when designing intrusion detection systems for cameras using ROS. Section III describes our intrusion detection system. Section IV describes the experiments and Section V concludes the paper.

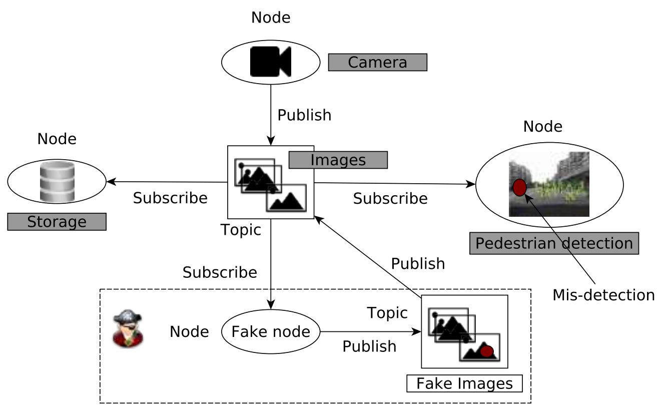


Fig. 1: Attack model on ROS camera node.

Scenario ID	Perturbation (attack)	Image
1	Insertion of a black image	6
2	Insertion of a white image	6
3	Blurred image	6
4	Image from the past	6
5	Successive same images	[4,8]
6	Flooding black images	[4,8]
7	Blurred image (a little blur)	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: Description of the attack scenarios with their corresponding images.

II. Adversarial models

Based on the vulnerabilities of ROS and the attacker model highlighted in [5] and [9], a number of attacks against the embedded camera of self-driving cars is defined in Fig. 1. The attacker creates a fake node in order to intercept, inject, or modify communications between the camera and the image processing nodes (ex : pedestrian detection node).

The proposed attack scenarios are described in Table I. Fig. 2 illustrates the different attacks applied on frames extracted from the embedded camera. Each line represents a series of frames over time, and contains perturbations following the attack scenarios proposed in Table I. In each scenario, the attack is represented by the 6th image, except for the 5th and 6th scenarios where the attacks are composed of several frames (from the 4th frame to the 8th frame).

The types of perturbed images (i.e. attacks) are:

- Insertion of an image: the attacker injects a fake image (white, black or an image from the past) in the image stream like in scenarios 1, 2, and 4.
- Blurred image: an image is altered by the attacker with an intense blur in scenario 3 and small one in scenario 7.
- Flooding images: an injection of multiple and successive images are injected in order to flood a processing node of images. Scenario 5 and 6 present flooding images, and flooding black images, respectively.
- Sophisticated attacks: they consist in adding small perturbations like forms on objects such as traffic lights. The goal of this attack is the misclassification or misrecognition of objects. The attacks can be: hiding a car (Fig. 3.a), hiding traffic lights (Fig. 3.b) or changing the color of the traffic lights by replacing the red light with a green one (Fig. 3.c).

From these scenarios, we show that the attacker can change data using low-cost techniques and thus cause misclassification. In the next section, we present our solution for detecting suspicious images in order to protect the camera system in a self-driving car.

III. Proposed solution

We propose, as illustrated in Fig.4, a solution to analyze video flows coming from ROS-enabled cars in order to detect attacks in real-time using autoencoders.

A. Autoencoders

Autoencoders [10] are a special type of Neural Networks used for reconstruction purposes. The principle is to train the model by taking an input x, and given an output objective y which is identical to the input (i.e : $x = y$), the goal is to reconstruct the input as similarly as possible by minimizing the difference between both. It is composed of two parts, an encoder, which takes the input x and encodes it to match a learned representation z, called *latent variable*, and a decoder that tries to decode the representation to come up with the original input.

The most important work done by an autoencoder, is to perform a dimensionality reduction, while keeping the most important information which allows a nearly lossless reconstruction, or at least, a minimal loss [11].

We use autoencoders for anomaly detection by feeding images as input to the model, so that it can learn the representation and reproduce them accurately. Then, if the input data is replaced by new data, different from the previous one, the model will map it differently in the latent space, and thus, reconstruct it differently.

B. Solution

Our solution is composed of two different parts: reconstruction and prediction. The reconstruction part is to reconstruct truthfully the images given in the input, taking into account the previous images. The prediction part consists on predicting the future image, given the present image, and the previous ones.

On a technical level, the proposed autoencoder is composed of two parts. The encoder part, which takes a sequence of preprocessed images, in form of time-series chunks, and encodes them in the latent space by passing them through several neural layers. Then, the decoder part takes the information from the bottleneck and proceeds to dimensionality augmentation to match the input's dimension.

We extract the spatial characteristics using two dimensional convolutions on single images, and in addition to that, we

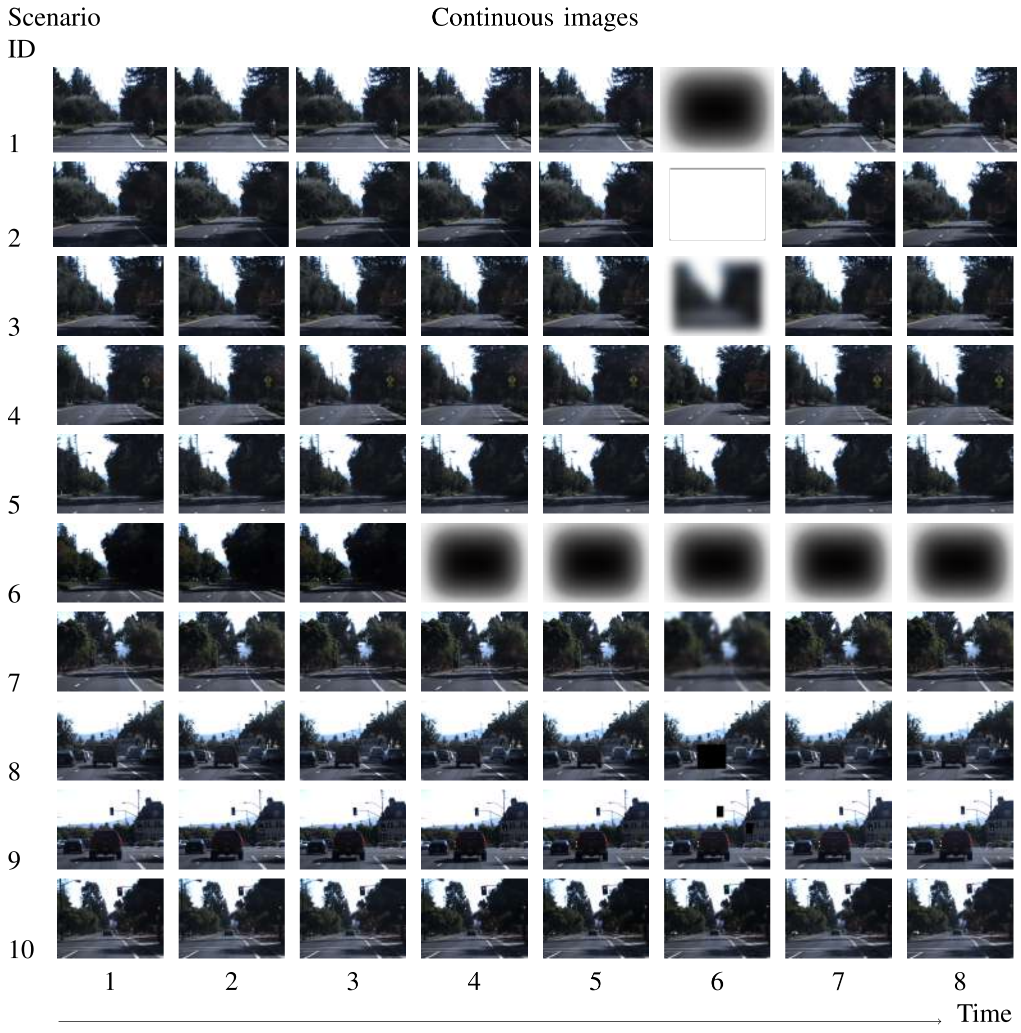
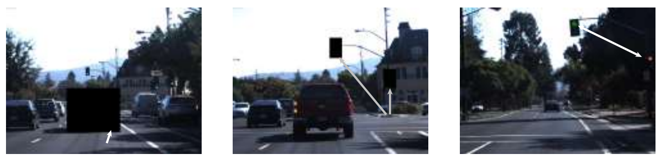


Fig. 2: Example of targeted attacks on a camera embedded on a self-driving car [5].



(a) A black rectangle hiding a car

(b) A black rectangle hiding the traffic lights

(c) A change of a color in the traffic lights (replacing red by green)

Fig. 3: Sophisticated perturbations on images [5].

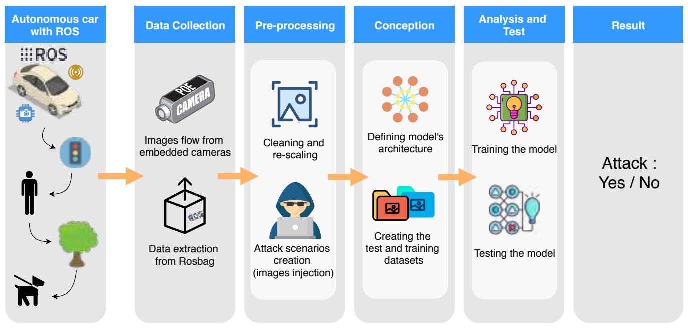

Autonomous car with ROS
ROS
Data Collection
Images flow from embedded cameras
Data extraction from Rosbag
Pre-processing
Cleaning and re-scaling
Attack scenarios creation (images injection)
Conception
Defining model's architecture
Creating the test and training datasets
Analysis and Test
Training the model
Testing the model
Result
Attack : Yes / No

Fig. 4: Schema of the proposed solution.

extract the temporal patterns using Recurrent Neural Network (RNN) cells. The type of RNN cells that we used is called Conv-LSTM [12]. In addition to the advantage offered by the standard LSTM over the RNN [13], in terms of solving the vanishing/exploding gradient problems and the long-term memory capacities, this type of cells is based on convolutions. Matrix multiplication is replaced by the Conv-LSTM with convolution operation at each gate in the LSTM cell. It captures spatial features by convolution operations in multiple-dimensional data.

C. Present image reconstruction

The principle of the reconstruction is to give the expected output as being the same as the input. So, when we start

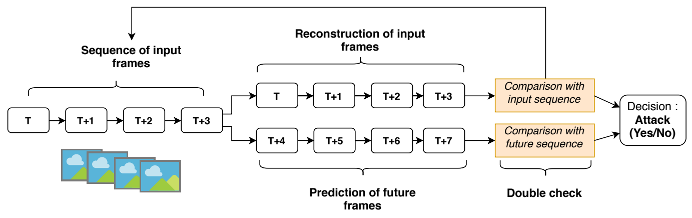

Sequence of input frames
Reconstruction of input frames
T
T+1
T+2
T+3
T+4
T+5
T+6
T+7
Comparison with input sequence
Comparison with future sequence
Decision : Attack (Yes/No)
Prediction of future frames
Double check

Fig. 5: Combined solution.

training the model, we define the target image as being the same as the one given in the input. Thus, after each epoch, the weights will be calculated through back propagation in order to produce an output that matches the input. On the side, the loss can be calculated with different methods, and we will present some of them in the experiment part.

D. Future image prediction

The architecture used for future frame prediction is similar to the reconstruction one. However, the training process is different; Instead of training the autoencoder to reconstruct the present image ($time = t$), it predicts the next incoming frame, given the present frame ($time = t$) as an input and the frame ($time = t + 1$) as a target.

E. Combined solution

The attack scenarios are various and have different characteristics. In order to maximize the detection, we decided to combine both the reconstruction and the prediction (Fig. 5).

In order to detect attacks, we use the *regularity scores*. They represent the result of the similarity measures between the input and output obtained after applying error functions that we will introduce in Section IV. By passing each sequence of frames through both models, we obtain two regularity scores for each scenario. We evaluate both of them and consider the existence of an attack if it is detected by either one of the two scores. We can consider it as two successive attack checkpoints. Some attacks can pass through one model without being detected, but passing through both models has a smaller probability to happen.

Some cases can easily be detected with reconstruction, if we take for example the case of frame injection such as black/white frames. Some others are more complicated and are harder to detect, like injecting images from the past. Applying this combination improves the regularity scores and gives better precision. We decided to separate the reconstruction and prediction models in order to have two independent latent spaces built and optimized for their respective purposes.

IV. Experimental results

A. Experimental setup

The experiments were conducted on machines running Ubuntu systems version "14.04.5 LTS : Trusty Tahr". It integrates an Intel(R) Xeon(R) CPU E5-2609 v3 with a base frequency of 1.90GHz. It has a RAM DDR memory of 128GB and two Nvidia Tesla K80 of 12GB each as a GPU. All the programming part was done using Python 3.6 with Tensorflow version 1.9.0 as a backend.

B. Dataset

We use a publicly available dataset of real-world driving data provided by Udacity [1]. The self-driving-car data from ROS are in ROSBAG file format. ROSBAG records all data incoming from different sensors. For our experiments, we extract videos from a camera.

The video contains more than 15000 frames. For the ground-truth, we construct different image perturbations. The images include both abnormal and normal images. Thus, the dataset contains 10 separate scenarios whose general characteristics were described in Table I.

C. Model Parameters

Error function and regularity score: We used both Mean Squared Error (MSE) and Structural Similarity Index (SSIM) [14] to train and evaluate the model performance. Based on Human Visual System (HVS), error visibility is correlated with loss of quality. However, MSE does not always have a good correlation with human perception. In other words, we can have cases where we can clearly see the difference between two images from a human vision perception, but MSE can give low error rates, indicating a high similarity. For those cases, SSIM proves to be a better technique.

Layers and dimensions: The number and types of layers of the proposed model have been fixed empirically. We have tested many combinations, varying the input dimension, the number of convolutions applied, the number of LSTM cells, and the number of Up/Down sampling, and according to the training and test results, fixed the architecture and parameters of the model. The goal being, finding the balance between reducing the dimension to extract useful information, and avoiding information loss due to too much reduction, while maintaining optimal reconstruction results. The input dimension has been fixed to 128x128x1 which means a resolution of 128x128 pixels with a unique channel. The reduction of dimension is applied twice, to obtain a dimension of 32x32x8 when reaching the bottleneck.

Optimizers: We have experimented four different optimizers: Adam, Adadelta, Adagrad, and RMSProp. We experimented each one with both the Mean squared error (MSE) loss and the SSIM (Structural similarity) loss.

D. Results

Table II shows the obtained scores for each case. It clearly depicts that both Adadelta and Adagrad are not well suited for our dataset as they are not optimized for time dependant data. However, the Adam optimizer has brought the lowest training losses, almost similar to the RMSProp.

In most cases, MSE gave better results. The regularity score is defined as being the same as the MSE score or the complement of SSIM to 1 (1-SSIM score).

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

Case	Optimizer	MSE score	SSIM score
Reconstruction	Adam	**0.0023**	**0.92**
	Adadelta	0.0037	0.91
	Adagrad	0.0036	0.90
	RMSProp	**0.0024**	**0.93**
Prediction	Adam	**0.0036**	**0.87**
	Adadelta	0.0047	0.86
	Adagrad	0.0044	0.84
	RMSProp	**0.0037**	**0.88**

TABLE II: Image similarity scores with different optimizers.

We have tested our model on the 10 different attack scenarios described in Section II. Fig. 6 shows the results of the regularity check with our reconstruction model of a partial video containing the first 6 scenarios. This result was obtained by using the mean squared error as it presented better results than SSIM with these scenarios.

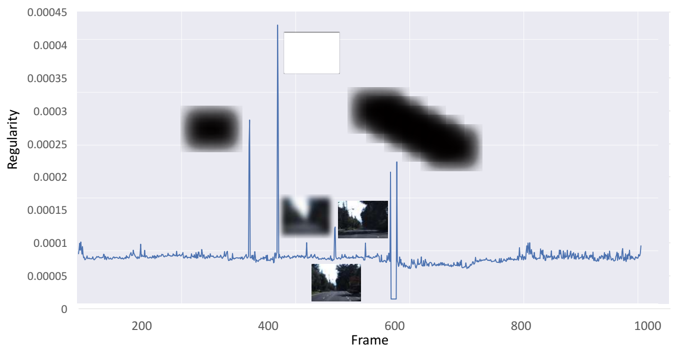


Fig. 6: Regularity scores (MSE) with the reconstruction model (6 attacks)

The outliers in the regularity score are clear and easy to spot with our model for these scenarios. Using simple thresh-hold methods, even static ones, we can easily spot the attacks as being anomalies, as they are separated from the normal sample distribution.

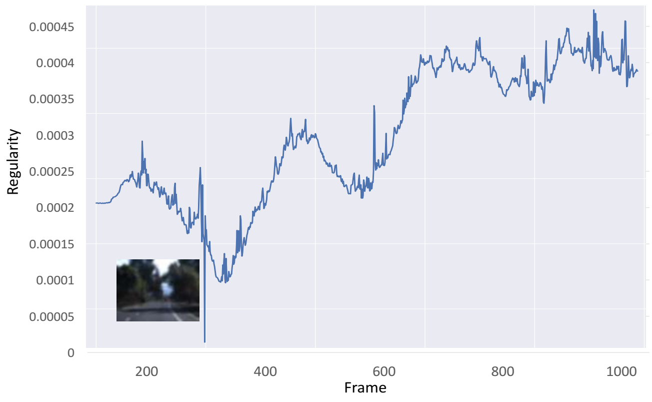


Fig. 7: Regularity scores (MSE) with the prediction model (advanced attacks)

For the remaining attack scenarios, the prediction model proved to be more effective. It is shown in Fig. 7 that scenario number 7 (Table I) presents a very low regularity score and is considered as anomalous. The sophisticated attacks require more advanced anomaly detection methods. A dynamic threshold is required to detect these scenarios.

V. Conclusion and Future Works

In this paper, we present a solution for detecting abnormal camera flows resulting from an attack on a ROS-enabled self-driving car. Our method is based on spatio-temporal autoencoders used to truthfully reconstruct the camera frames and detect abnormal images by measuring the difference with the input. The solution includes both Reconstruction and Prediction techniques, and combines them to get better results in anomalous image detection. Several attack scenarios are presented. The different attacks are perturbations on camera flows performed by an attacker in order to cause object misrecognition when using machine learning algorithms. Our experimental results on real data show the ability of our method to detect several attack scenarios.

References

[1] Ros. `http://wiki.ros.org`. Accessed: 2019-07-31.
[2] Jerry Towler and Matthew Bries. Ros-military: Progress and promise. In *Ground Vehicle Systems Engineering and Technology Symposium (GVSETS)*, 2018.
[3] Autoware. `https://www.autoware.ai`. Accessed: 2019-06-20.
[4] Jarrod Mcclean, Christopher Stull, Charles Farrar, and David Mascareñas. A preliminary cyber-physical security assessment of the robot operating system (ros). page 874110, 05 2013.
[5] Sofiane Lagraa, Maxime Cailac, Sean Rivera, Frédéric Beck, and Radu State. Real-time attack detection on robot cameras: A self-driving car application. In *IEEE International Conference on Robotic Computing (IRC)*, 2019.
[6] Francisco Javier Rodrıguez Lera, Jesús Balsa, Fernando Casado, Camino Fernández, Francisco Martın Rico, and Vicente Matellán. Cybersecurity in autonomous systems: Evaluating the performance of hardening ros. *Málaga, Spain*, 47, 2016.
[7] Alberto Giaretta, Michele De Donno, and Nicola Dragoni. Adding salt to pepper: A structured security assessment over a humanoid robot. *CoRR*, abs/1805.04101, 2018.
[8] Francisco Martín, Enrique Soriano, and José M. Cañas. Quantitative analysis of security in distributed robotic frameworks. *Robotics and Autonomous Systems*, 100:95 – 107, 2018.
[9] Sean Rivera, Sofiane Lagraa, and Radu State. Rosploit: Cybersecurity tool for ros. In *IEEE International Conference on Robotic Computing (IRC)*, 2019.
[10] Pierre Baldi. Autoencoders, unsupervised learning and deep architectures. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, pages 37–50, 2011.
[11] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, pages 3371–3408, 2010.
[12] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.
[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.
[14] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 13(4):600–612, 2004.