A Comparative Analysis on the use of Autoencoders for Robot Security Anomaly Detection

Conference Paper · November 2019

DOI: 10.1109/IROS40897.2019.8968105

6 authors, including:

Omar Cotugno
Descartes & Mauss
1 PUBLICATION **11** CITATIONS
SEE PROFILE

Lorenzo Brigato
Universität Bern
22 PUBLICATIONS **183** CITATIONS
SEE PROFILE

Domenico Daniele Bloisi
International University of Rome UNINT
95 PUBLICATIONS **1,956** CITATIONS
SEE PROFILE

Alessando Farinelli
University of Verona
252 PUBLICATIONS **3,827** CITATIONS
SEE PROFILE

A Comparative Analysis on the use of Autoencoders for Robot Security Anomaly Detection

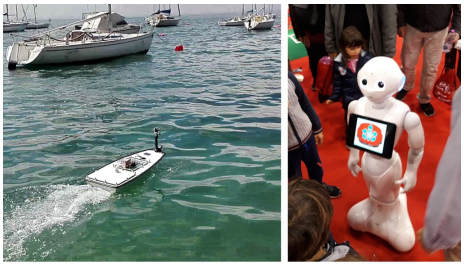
Matteo Olivato*[1], Omar Cotugno*[2], Lorenzo Brigato*[2], Domenico Bloisi[3], Alessandro Farinelli[4], Luca Iocchi[2]

***Abstract*— While robots are more and more deployed among people in public spaces, the impact of cyber-security attacks is significantly increasing. Most of consumer and professional robotic systems are affected by multiple vulnerabilities and the research in this field is just started. This paper addresses the problem of automatic detection of anomalous behaviors possibly coming from cyber-security attacks. The proposed solution is based on extracting system logs from a set of internal variables of a robotic system, on transforming such data into images, and on training different Autoencoder architectures to classify robot behaviors to detect anomalies. Experimental results in two different scenarios (autonomous boats and social robots) show effectiveness and general applicability of the proposed method.**

I. Introduction

The number and importance of decisions delegated to computers and electronic systems interconnected together is constantly growing. In this context, systems that control critical devices operating in the physical world can be compromised by cyber-attacks. As far as Cyber-Security for robots is concerned, there are three main groups of problems to be considered: 1) safety risks related to the physical actions of the robots; 2) theft of sensitive information; 3) damages to business or personal reputation. Cyber-attacks against robots operating in offices, restaurants, shopping centers, hospitals, etc. can produce business and personal damages, caused by hacked behaviors of the robots.

In this work, we consider two real autonomous robotic systems (see Fig. 1) as case studies: a robotic boat for water quality monitoring and a social robot. We propose a solution for identifying possible attacks by analyzing the robot's behavior. The proposed approach is based on the analysis of system logs and on the development of an analyzer module, based on autoencoders, to identify abnormal behaviors. In more detail, the proposed approach aims at detecting attacks to a cyber-physical system, in real-time, by processing system logs containing data related to the activities of the system (e.g., GPS position, heading, and velocity). Data are recorded at short time interval (e.g., every 200 ms) hence the logs provide a rich representation of the behavior of the system that is extremely valuable to identify faults and cyber(-physical) attacks. In particular, we propose a transformation process to convert the different types of variables included in the log records into a binary pixel array which resulted to be a sparser and more effective representation. This process allows to learn specific patterns of the log data related to normal behaviors and to detect possible abnormal situations.



(a) (b)

Fig. 1: Autonomous robots used as case studies. (a) Boat for water quality monitoring. (b) Pepper social robot for education.

In summary, the contribution of this work is two-fold: 1) the definition of a novel problem that focuses on detecting robot anomalous behaviors from system logs and 2) a novel method to analyze robot behaviors based on transforming system logs in images and on image reconstruction through autoencoders.

II. Related Work

Clark et al. [1] present a general robot cyber-security analysis concerning common vulnerabilities and countermeasures at different levels with an emphasis on the economic impact of cyber attacks on manufacturing and supply chains. An Hidden Markov Model (HMM) based approach is used by Park et al. [2] to identify anomalies in robot manipulation tasks. They show that multimodal anomaly detection outperforms unimodal detection. The same authors describe in [3] an improved version of their detector based on a long short-term memory variational autoencoder (LSTM-VAE). The detector estimates the negative log-likelihood of the multimodal input with respect to the distribution as an anomaly score. Harrou et al. [4] address the problem of anomaly detection in photovoltaic system arrays using One-class Support Vector Machine (SVM). They built a simulation model of the system to describe the normal behaviour and generate residuals for fault detection. Gautam et al. [5] propose Localized Multiple Kernel Learning for anomaly detection (LMKAD) as one-class classification problem. LMKAD provides a localized

* Equal contribution from M. Olivato, O. Cotugno and L. Brigato.
[1]Dept of Information Engineering, University of Brescia, Italy m.olivato@unibs.it
[2]Dept of Computer, Control and Management Engineering, Sapienza University of Rome, Italy iocchi@diag.uniroma1.it
[3]Dept of Mathematics, Computer Science, and Economics, University of Basilicata, Italy domenico.bloisi@unibas.it
[4]Dept of Computer Science, University of Verona, Italy alessandro.farinelli@univr.it

formulation for multi-kernel learning method by local assignment of weights to each kernel. Kirschgens et al. [6] analyze the consequences of insecure policies in robot development. They show alarming cases with the aim to create awareness about enforcing strong security in robotics.

One of the use cases described in this paper involves a SoftBank Pepper robot (see Fig. 1) that uses the NAOqi framework. About NAOqi security, Giaretta et al. [7] performed a set of different security assessments, both automated and manual, and discussed vulnerabilities that may enable an attacker to spoof user's credentials, steal data stored in the robot, hack other connected devices, etc.

While specific approaches to anomaly detection in robot security have been presented, to the best of our knowledge, we are proposing a novel general purpose domain independent approach using a machine learning method to identify attacks and faults based only on robot's internal system logs.

Our approach is inspired by the work of the winners of the Microsoft Malware Classification Challenge (BIG 2015)[1]. They converted malware binary assembly representations to greyscale images. This process results in a more dense features representation, which in turn avoid a degradation in the performance of the classification algorithm (they used XGBoost). Even if the idea of visualizing malware images for classification can be found in previous work [8], [9], the Microsoft competition evaluated the real effectiveness of this method on real-world data. In contrast with such previous work, in this paper we apply this concept to real-time data coming from robot's system logs. In addition, we consider an unsupervised anomaly detection problem and our experimental results show that sparse representations are more effective.

III. Problem Definition

Let us consider a general (robotic) system $\mathcal{S}$ in which it is possible to access a set of variables $\mathbf{V}$ characterizing its internal state over time. Let $\mathbf{r}_t$ be a record, i.e. a tuple of values of such variables collected at time t, and $\pi = \{\mathbf{r}_1, \ldots, \mathbf{r}_T\}$ a system log of a behavior containing the values of variables V recorded from time 1 to time T. Let us denote with Π the set of all possible logs π_i, with R being the set of all possible records $\mathbf{r}_i$. Let us consider a scenario in which such logs are taken when the system is performing a nominal (correct) behavior. The problem considered in this paper, which we call *one-class classification of system logs*, can be expressed as follows.

Given a dataset of system logs $D = \{(\pi_i, \text{normal})_{i=1}^m | \pi_i \in \Pi\}$*, capturing the nominal behavior of the system, generate a class model that will be able to classify new instances as normal or abnormal.*

Notice that, although the output of the system will be a binary classification (normal vs. abnormal), the training set contains only samples from the normal class. Practical importance of this definition of the problem in robot security is that, in this way, it is not necessary to predict and collect abnormal situations, which are often very difficult to predict and to reproduce.

The on-line procedure can be implemented by processing in real-time the logs captured in a time window that is calibrated with respect to the duration of the expected attacks/faults. For example, high frequency attacks, where the attacker changes the status of the robot at high frequency (e.g., $>$ 1 Hz), can be detected with a few seconds of log recording.

IV. Log-to-image Autoencoder Classification

We propose a solution to the one-class classification problem described in the previous section that is based on learning a class model for the single class *normal* for which we have samples in the dataset. Training and classification phases of our approach are described below.

Training phase. Given a one-class dataset of system logs for normal behaviors $D = \{(\pi_i, \textit{normal})_{i=1}^m | \pi_i \in \Pi\}$, learn a class model $\mathcal{M}$.

Classification phase. Given a new log $\pi' = \{\mathbf{r}'_0, \ldots, \mathbf{r}'_T\}$, determine using the learned model $\mathcal{M}$ if it is *normal* or *abnormal*.

The performance evaluation is obtained by using a test set T_S containing both normal and abnormal records. It is worth noticing that the records from security attacks or faults are used for performance evaluation only and they are not needed in the actual deployment of the system.

A key element in our approach is the transformation of each record $\mathbf{r} \in R$ into a squared image $I \in \mathcal{I}$ (denoting with $\mathcal{I}$ the set of images). To this end, we implemented a *log-to-image transformation function* $\sigma : R \to \mathcal{I}$ able to transform log records into images. More precisely, given a dataset D_R containing records $\mathbf{r}_t$ collected during normal behaviors of the robot, we define a new dataset of labeled images $D_I = \{(\sigma(\mathbf{r}_t), \textit{normal}) \mid \forall \mathbf{r}_t \in D_R\}$ computed from the system logs that will be used for generating the class model.

The remaining of this section describes the two main processes of our method: 1) the generation of images from a system log (i.e., the implementation of the *log-to-image* function σ), 2) the solution of the one-class classification problem.

A. Log-to-image Transformation

The log-to-image transformation function $\sigma : R \to \mathcal{I}$ is implemented in two steps, such that $\sigma(\mathbf{r}_t) \equiv \rho(\tau(\mathbf{r}_t, \mathbf{T}))$, where the τ transforms a record $\mathbf{r}_t$ into a transformed sequence k_t depending on the applied transformation $\mathbf{T}$. The function ρ transforms k_t into an image I_t.

In order to transform log records into binary images, we represent a binary image as a matrix of pixels, where each pixel is either 0 or 1. To obtain such an image, we proceed in two steps: 1) we encode all the values (of different types) in their standard binary representation, 2) we generate a binary image from such a sequence of binary chunks.

Binary record transformation. As previously mentioned, the first transformation represents each value as a binary

[1]Winning team was composed by J. Liu, X. Chen, X. Wang

Algorithm 1 Procedure τ to convert a log record $\mathbf{r}$ applying transformation $\mathbf{T}$

function $\tau(\mathbf{r}, \mathbf{T}) \rightarrow k$
  $k \leftarrow [\,]$
  if $\mathbf{T} = B$ **then** ▷ binary representation
    for $i \leftarrow 0$ **to** $|\mathbf{r}| - 1$ **do**
      $bits \leftarrow [\,]$
      $v \leftarrow \mathbf{r}[i]$
      if $type(v) = int \vee type(v) = double$ **then**
        $bits \leftarrow getBitRepresentation(v, 64)$
        $k.concat(bits)$
      else
        $\cdots$ ▷ possible extension to other types
      end if
    end for
  else if $\mathbf{T} = N$ **then** ▷ normalized representation
    $k \leftarrow Normalize(\mathbf{r}, Max, Min)$
  else
    $\cdots$ ▷ other representations
  end if
  return k
end function

Algorithm 2 Procedure ρ to convert a sequence k into a $W \times W$ image I

function $\rho(k, W) \rightarrow I$
  for $i \leftarrow |k|$ **to** $W^2 - 1$ **do** ▷ Padding if needed
    $k.append(Zero(\mathbf{T}))$
  end for
  $k' \leftarrow k[0 : W^2 - 1]$ ▷ Truncating if needed
  $I \leftarrow toSquaredMatrix(k', W)$
  return I
end function

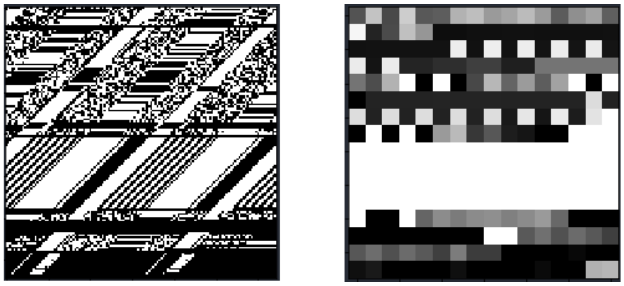


Fig. 2: Example of a binary (left) and grayscale (right) image.

array. In order to properly organize the record data, variables (and thus the corresponding values) are ordered in a specific way: 1) first, all the fixed-size variables are considered in lexicographical order; 2) then, all the variable-size variables (i.e., strings) are considered in lexicographical order. This order will produce a shallow semantic grouping of variables i.e. the variables referred to the same sensor use the same prefix so, after sorting, they will compare near each other. In the following, we assume that every record $\mathbf{r} \in R$ is a tuple of values ordered in this way. The encoding of each record into a *binary* sequence (τ function with $\mathbf{T} = B$) is obtained through the procedure described in Algorithm 1.

Normalized record transformation. We use a second data representation which corresponds to the normalization of the log records in the range $[0 - 1]$. In such manner, the images that will be generated are grayscale. This representation is obtained in Algorithm 1 by setting $\mathbf{T} = N$. Max and Min are vectors that contain maximum and minimum values for each feature in the training dataset.

Image generation. The second process transforms the sequences produced in the first process in a fixed-size square image. The adopted solution is based on calculating the largest log record and on defining the size of the image with respect to such a value. All samples with a lower size are properly padded, while data entries obtained at run-time that are larger of the selected size are truncated, as explained below. More specifically, given a log dataset D, the desired size $W \times W$ of the images is computed as:

$$W = \lceil \sqrt{\max_{\pi \in D} \{\texttt{length}(\tau(\mathbf{r}_t, \mathbf{T})) \mid \mathbf{r}_t \in \pi\}} \rceil$$

where `length` denotes the length of a sequence returned by the function τ.

With this choice and by properly padding all the chunks of smaller size, all the images in the dataset D_I have the same dimensions and convolutional autoencoders might be trained on them.

Once the size $W \times W$ of the images has been computed, the procedure to transform sequences into images (ρ function) can be implemented as described in Algorithm 2. The function $Zero(\cdot)$ executes a padding depending on the type of transformation that has been applied. Some resulting binary and grayscale images are shown in Fig. 2.

B. Class Modelling

Class modelling is performed through two different phases. First, one or multiple autoencoders are trained over the majority of the one-class dataset D_R to learn its latent representation. Then, the affiliation to the *normal* class follows a decision rule based on a threshold. It is computed through the loss function and estimated on a smaller sample of D_R. We decided to compare the previous class modeling method with a standard One-class SVM solution.

1) Architecture of the Autoencoders: We used three different autoencoders: a Convolutional Neural Network (CNN), a Shallow and a Deep Feed-Forward Neural Network. We will refer to them as *ConvEnc*, *Enc* and *DeepEnc*. The *ConvEnc* is composed by five convolutional layers with kernel size of dimensions $(3, 3)$. The first two are followed by Max-pooling layers whereas the central two layers with Up-sampling. The final convolutional layer is directly connected to the network output. The *Enc* has a single hidden layer of dimension 32 units, while the *DeepEnc* has in total five hidden layers with the first and central layers of dimension, respectively, 128 and 32. To train each network, Binary Cross-Entropy Loss and Adam optimizer have been used. We fixed the number of epochs to 30. We have used ReLUs for all hidden layers

and Sigmoids for the output layers. The *Enc* and *DeepEnc* were trained with records transformed in sequences (i.e., the output of τ function) whereas the *ConvEnc* in images.

2) Loss and Losses Variance based Classification: Since autoencoders are unsupervised models, a second step is needed to allow the detection of *abnormal* behaviours. The loss value is a straightforward metric to evaluate the one-class membership. Indeed, *abnormal* samples are expected to have a greater loss with respect to *normal* samples. Let us define a subset of logs from the *normal* dataset $D_{thr} \subset D_R$. The samples in D_{thr} are not used during the network training. Given D_{thr} and the loss function $\mathcal{L}$ of a trained network, we compute $\mathbf{l} = \mathcal{L}(D_{thr})$, with $\mathbf{l}$ being a vector containing the loss values for each record $\mathbf{r}_t \in D_{thr}$. It is then possible to compute the range of expected *normal* losses which might be calculated in different ways. We chose to set these bounds as $\delta_u,\ \delta_l = \mu(\mathbf{l}) \pm z \cdot \sigma(\mathbf{l})$, where μ is the mean and σ is the standard deviation of the values in $\mathbf{l}$, with the possibility to change the value of z in order to vary the interval. A testing sample is classified as *normal* if the value of its loss l_t is in between δ_u and δ_l.

This classification framework could be generalized to the case of multiple networks by using the variance of the prediction losses. The basic intuition is that the losses of easily detectable *normal* records have similar values, consequently a low variance, even though networks parameters are different. On the contrary, *abnormal* samples are expected to produce a higher loss variability. Let us assume to train n different networks. In this way, given the set D_{thr}, it is possible to compute n vectors of prediction losses $\mathbf{l}_n$ (where $\mathbf{l}_i$ corresponds to the losses of records as predicted by network i). Then, the losses variance is computed over the n network predictions, obtaining one value for each record belonging to D_{thr}. If we refer to this vector as $\mathbf{l}_{\sigma^2}$, we can compute the upper threshold as $\delta_u = \mu(\mathbf{l}_{\sigma^2}) + z \cdot \sigma(\mathbf{l}_{\sigma^2})$. Therefore, a testing record classified by n networks is considered *normal* if the losses variance $l_{\sigma^2,t}$ satisfies the inequality $0 = \delta_l < l_{\sigma^2,t} < \delta_u$. We keep $\delta_l = 0$ because the variance of the prediction losses is by construction always greater than 0 therefore, it is not worth to take into account a lower bound $\delta_l = \mu(\mathbf{l}_{\sigma^2}) - z \cdot \sigma(\mathbf{l}_{\sigma^2}) < 0$ for many values of z.

C. General Applicability and System-dependent Choices

Observe that the proposed method could be applied to several different situations and it is not thought to be specific of any system. It can apply to a large variety of robotic applications, where system logs may help in monitoring security conditions. However, when applied to a specific domain, the implementation of the proposed method requires to consider the following design choices.

1) Selecting the system variables suitable for characterizing the system and the task under analysis.
2) Collecting data during normal operation of the robot.
3) Optionally refine the configuration of the networks to get best performance (e.g., modify the hidden layers of *DeepEnc* and *Enc* depending on the input size).
4) Training the networks using the collected dataset and the transformation functions described above.

In the next section, we describe two different application scenarios in which the proposed method has been tested.

V. Case Studies and Results

In this section, we present experimental results[2] in two different domains: autonomous boats for environmental monitoring and social robots in public environments.

As described in the previous sections, we are interested in anomaly detection by using a classifier trained only on *normal* instances. The record datasets D_R used in these experiments are composed by a set of log files captured during normal operation of the robots in the two scenarios. All implemented models have been tested on both representations, *binary* and *normalized* sequences. Then we created a binary test set formed by normal and abnormal situations (simulating different kinds of security attacks) that is only used for performance evaluation. We collected a greater number of abnormal samples in order to effectively test the detection ability of the proposed models.

The results described below show a very good classification performance in both the scenarios, demonstrating the effectiveness and the generality of the proposed method in detecting behaviors affected by security issues.

A. Autonomous Surface Vehicles

Low-cost Autonomous surface vehicles (ASVs) are employed in the EU-funded project INTCATCH[3] to develop efficient and user-friendly monitoring strategies for facilitating sustainable water quality management by non-experts. INTCATCH boats can navigate autonomously using GPS data. Cyber-attacks and hacking are aspects clearly associated with the navigation of drones in general and autonomous boats in our reference application. Moreover, ASVs for water monitoring transmit the acquired data regarding water quality (e.g., Dissolved Oxygen, Ph level, Electrical Conductivity and so forth) using standard networks (a standard WiFi network and the 3G mobile network in our case) hence such data could be exposed to tampering.

For this case study, we have collected a training set D_R with $2,195$ *normal* records taken during the nominal behavior of the robot (659 of them were used to estimate the thresholds). On the other hand, the test set T_S contained $12,280$ records distributed among six categories as follows: 660 normal and the rest coming from two kinds of security attacks and faults. The latter were divided in: DoS with no payload 19%, DoS with *32 bytes* payload 27%, GPS down fault 27% and Vehicle got stuck fault 27%.

B. Social Robot

SoftBank Pepper social robot[4] is a robot designed for human-robot interaction and social robotics and it is com-

[2]Implementation and datasets will be published to reproduce the results and compare other methods.

[3]www.intcatch.eu

[4]www.softbankrobotics.com/emea/en/robots/pepper

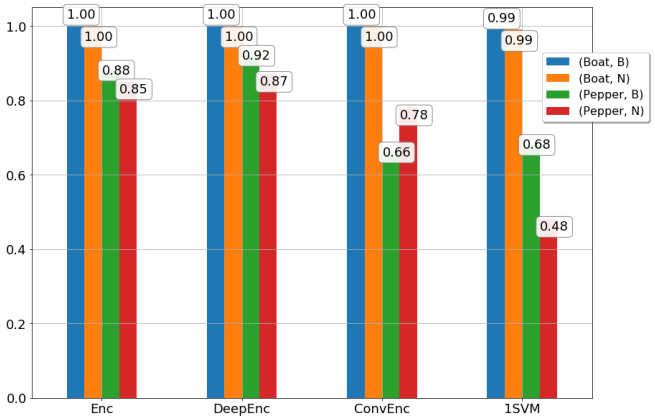


Fig. 3: Test accuracy of the best performing models (design parameters n and z) for each dataset and data representation.

monly used in public spaces to interact with people (including children). As mentioned in Section II, a number of security issues of this platform are still under investigation. As already mentioned, a dataset D_R with only *normal* records (taken from Pepper's built-in autonomous interactive behavior) has been collected to train the class model, while the test set T_S contains, in addition to some other normal records, the following simulated attacks: 1) *LedsControl*, where robot's leds are controlled by a remote program and activated in a pseudo-random mode; 2) *JointsControl*, where robot's joints are controlled by a remote program setting joints values to obtain special poses of the robot; 3) *WheelsControl*, where the robot movement is controlled by an external program connected to a remote joystick to navigate in the environment.

As state variables for the logs, we considered 256 real values from different sensors. We decided to not include in the logs data coming from microphones and video-cameras both for privacy concerns and for simplicity in the data processing. The dataset D_R used for training is composed by a total number of $14,144$ records, divided into $9,900$ samples for training and $4,244$ samples for thresholds evaluation, while the test set T_S contains $12,532$ samples, $8,288$ are abnormal samples, distributed as follow: *WheelsControl* 30%, *LedsControl* 35%, *JointControl* 3% and the remaining $4,244$ are normal.

C. Results

The results are mainly focused on the ability of the system to detect anomalous behaviors. Fig. 3 shows a summary of the results obtained in terms of accuracy in the test datasets for both the scenarios, comparing the proposed methods described in this paper, considering the resulted best design parameters n and z, with OneSVM. The latter has been trained using both *binary* and *normalized* log records and tuned on the D_{thr} validation dataset. From this picture, the advantage of using autoencoder architectures appears clear. Generally, a more detailed analysis is needed, as a good balance between recall and precision scores is expected from a security system, although there may be specific needs that prefer to maximize one or the other measure. Following the standard Anomaly Detection nomenclature, we consider *abnormal* samples as true positives and *normal* samples as true negatives. In order to properly fine-tune the parameters of a model we need a lot of abnormal data well logged. This type of data are rare and difficult to obtain in real conditions, so this could be an obstacle for the fine-tuning process. For this reason, in Tables I and II, we show the performance in terms of F1 score to picture the sensitivity of the tested anomaly detection models considering both the *binary* and *normalized* representations. The results have been obtained by varying the interval parameter z in the range of 0-3, while for the number of networks n we have considered four values, namely $\{1, 3, 5, 10\}$.

Table I shows the results for the autonomous surface vehicles. All models are able to reach a very high F1 scores with various configurations of the parameters z and n. The results of the social robot scenario, showed in Table II, denote a more challenging task with general lower performance, due to the higher data dimensionality (256 vs 32 original variables) and more realistic collected data. The gap between the *normal* and *abnormal* testing sample losses is certainly narrower in this case study. Indeed, keeping a large *normal* confidence interval, corresponding to greater z values, implies a high number of erroneously classified *abnormal* samples. The drop in performance caused by decreasing z is evident for all models but is much less severe when a single *Enc* or *DeepEnc* is trained with *binary* representations. The single *DeepEnc* reaches a highest F1 score of 0.94 with $z = 1$ and drops to 0.87 when $z = 3$ whereas its corresponding *normalized* network goes from 0.81 to 0.65. We suppose that such robustness is due to the higher sparsity of the *binary* representation that in turn regularizes the data reconstruction of feed-forward autoencoders. On average, a *binary* log record from this case study is $\sim 50\%$ sparse while its *normalized* counterpart is $\sim 10\%$.[5] The use of $n > 1$ networks and $z = 0$ improved the ability of recognizing *abnormal* samples when the *normalized* transformation is applied. The *ConvEnc* and *Enc* F1 scores rise from 0.77 and 0.75 to, respectively, 0.82 and 0.87 when using 5 networks. The *DeepEnc* from 0.81 to 0.89 when an ensemble of $n = 10$ autoencoders is used. The *ConvEnc* model performed better when input data where *normalized*. This result is probably caused by the nature of convolutions that are likely to find better correlations when input features are more dense. The *ConvEnc* was expected to perform worse than the feed-forward networks since convolutional layers rely on the assumption that the statistical structure of local patches of the input is invariant with regard to translation while in our case the image is artificially built. However, more complex log-to-image transformations such as 1) semantic representation of the log value by means of colors or 2) semantic spatial arrangement of fields values in the image or 3) both techniques, are expected to improve the classification results of convolutional autoencoders. Thus, results are promising

[5]Sparsity is computed as the number of zeros divided by the array size.

TABLE I: Autonomous surface vehicles F1 score results varying the design parameters n and z on two different log representation: B (binary) and N (normalized). Top-2 values for each model are reported in bold.

n \ z	0		1		2		3	
	B	**N**	**B**	**N**	**B**	**N**	**B**	**N**
Enc								
1	–	–	**0.99**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**
3	**0.99**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
5	0.98	**0.99**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
10	**0.99**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
DeepEnc								
1	–	–	**0.99**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**
3	**0.99**	0.42	**1.00**	0.42	**1.00**	0.42	**1.00**	0.42
5	**0.99**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
10	**0.99**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
ConvEnc								
1	–	–	**0.99**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**
3	**0.99**	0.03	**0.99**	0.03	**1.00**	0.03	**1.00**	0.03
5	**0.99**	0.42	**1.00**	0.42	**1.00**	0.42	**1.00**	0.42
10	**0.99**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

TABLE II: Social robot F1 score results varying the design parameters n and z on two different log representation: B (binary) and N (normalized). Top-5 values for each model are reported in bold.

n \ z	0		1		2		3	
	B	**N**	**B**	**N**	**B**	**N**	**B**	**N**
Enc								
1	–	–	**0.90**	0.75	**0.87**	0.65	**0.84**	0.58
3	**0.78**	**0.84**	0.50	0.71	0.41	0.68	0.37	0.63
5	**0.84**	**0.87**	0.48	0.72	0.39	0.66	0.35	0.62
10	**0.90**	**0.83**	0.46	0.62	0.32	0.60	0.23	0.57
DeepEnc								
1	–	–	**0.94**	0.81	**0.90**	0.74	0.87	0.65
3	**0.88**	0.87	0.78	0.69	0.74	0.63	0.70	0.60
5	**0.91**	0.86	0.76	0.74	0.67	0.67	0.56	0.60
10	**0.91**	**0.89**	0.72	0.77	0.66	0.70	0.62	0.67
ConvEnc								
1	–	–	0.66	**0.77**	0.47	0.63	0.37	0.54
3	0.50	**0.81**	0.23	0.47	0.12	0.31	0.09	0.24
5	0.68	**0.82**	0.53	**0.69**	0.36	0.54	0.14	0.43
10	0.49	**0.78**	0.14	0.65	0.03	0.54	0.01	0.49

but there is still room for improvements.

VI. Conclusions

In this paper, we have presented a method for detecting anomalous behaviors of a robot from its system log. We define the problem as a one-class classification for training and as binary classification for deployment and performance evaluation. The advantage in using one-class classification is that it does not require to collect any example of possible attack or fault of the robot (which may be very difficult to predict *a priori*). The proposed solution contributes to the research in cyber-security for robot applications by allowing a real-time on-line analysis of the robot behavior. The experimental results show a very good classification performance and a high degree of generalization demonstrated by the use of data coming from two different robotic case studies.

We believe that research in the field of cyber-security for robots, addressing both solutions to limit vulnerabilities and on-line analysis of robot behaviors, will have a significant impact on actual deployments of consumer and professional robots in public environments.

As future directions, we intend to study other data representations based on images, extend the data sources for the logs by including additional sensors (such as cameras and microphones), consider the temporal sequence of the records, and a continuous learning setting with the support of security operators.

References

[1] G. W. Clark, M. V. Doran, and T. R. Andel, "Cybersecurity issues in robotics," in *Cognitive and Computational Aspects of Situation Management (CogSIMA), 2017 IEEE Conference on*. IEEE, 2017, pp. 1–5.

[2] D. Park, Z. Erickson, T. Bhattacharjee, and C. C. Kemp, "Multimodal execution monitoring for anomaly detection during robot manipulation," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 407–414.

[3] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1544–1551, 2018.

[4] F. Harrou, A. Dairi, B. Taghezouit, and Y. Sun, "An unsupervised monitoring procedure for detecting anomalies in photovoltaic systems using a one-class support vector machine," *Solar Energy*, vol. 179, pp. 48–58, 2019.

[5] C. Gautam, R. Balaji, K. Sudharsan, A. Tiwari, and K. Ahuja, "Localized multiple kernel learning for anomaly detection: One-class classification," *Knowledge-Based Systems*, vol. 165, pp. 241–252, 2019.

[6] L. Alzola Kirschgens, I. Zamalloa Ugarte, E. Gil Uriarte, A. Muñiz Rosas, and V. Mayoral Vilches, "Robot hazards: from safety to security," *ArXiv e-prints*, June 2018.

[7] A. Giaretta, M. De Donno, and N. Dragoni, "Adding salt to pepper: A structured security assessment over a humanoid robot," *arXiv preprint arXiv:1805.04101*, 2018.

[8] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*. ACM, 2011, p. 4.

[9] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. ACM, 2011, pp. 21–30.