A Minimally Supervised Approach Based on Variational Autoencoders for Anomaly Detection in Autonomous Robots*

Davide Azzalini**, Luca Bonali, and Francesco Amigoni

Politecnico di Milano, Milan, Italy
{davide.azzalini, francesco.amigoni}@polimi.it
luca.bonali@mail.polimi.it

Abstract. Detection of anomalies and faults is a crucial ability for fully autonomous robots. This paper proposes a new deep learning-based minimally supervised method for detecting anomalies in autonomous robots. We contribute a new Variational Auto-Encoder architecture able to model very long multivariate sensor logs exploiting a new incremental training method, which induces a progress-based latent space that can be used to detect anomalies both at runtime and offline. While most existing approaches are trained in a semi-supervised fashion and require big batches of nominal observations, our method is trained using unlabeled observations of a robot performing a task, containing both nominal and anomalous executions. Only a very little amount (even just one) of labeled nominal executions is then required to partition the learned latent space into nominal and anomalous regions. Experimental results show that our method outperforms state-of-the-art anomaly detectors commonly used in robotics both in terms of false positive rate and alert delay.

Keywords: Anomaly detection · Autonomous robots · Long-term autonomy.

1 Introduction

Robots are increasingly expected to operate for long periods of time without human supervision, toward the goal of achieving *Long-Term Autonomy* (LTA) [16]. One of the fundamental ingredients of LTA is *anomaly detection* [7], namely the ability of robots to autonomously detect their anomalies and faults. In fact, a fault which is not promptly detected and addressed may result in the robot damaging itself or, even worse, in harming surrounding people. Different data-driven methods for detecting anomalies in robot systems have been proposed, most of

* This paper is an extended version of [2] that provides further experimental results and discussions (Sections 4.5 and 4.6).

** Davide Azzalini is supported by the ABB-Politecnico di Milano Joint Research Center, which provides financial support.

them being semi-supervised and requiring big batches of observations labeled as nominal by human experts for their training (e.g., [3, 21–23]).

This paper proposes a new deep learning-based *minimally supervised* method for detecting anomalies in autonomous robots. Anomaly detection can be done at different levels of abstraction, in our case we are interested in identifying anomalies not at a component-level, but in the overall behavior of a system. In particular, we propose a new VAE (Variational Auto-Encoder) [14] architecture able to model very long multivariate sensor logs of a robot performing a task. We also introduce a new incremental method for training VAEs, which induces a progress-based latent space that can be used to detect anomalies both online (at runtime) and offline. An original feature of our approach is that, differently from most approaches for anomaly detection in robotics, it is trained with unlabeled observations, possibly including both nominal and anomalous executions. Only few (even just one) labeled nominal executions are then required to partition the learned latent space into nominal and anomalous regions. This minimally supervised approach provides a big advantage over semi-supervised approaches in practical settings, where collecting several nominal runs of a robot performing a task could be hard, since a human expert is usually required to supervise the system in order to label runs as nominal. This is true especially when using deep learning-based algorithms which notoriously demand big datasets to be trained effectively. Experimental results on datasets collected from real robots show that our method outperforms state-of-the-art methods for anomaly detection in robots both in terms of false positive rate and alert delay.

This paper thus presents a novel application of VAEs to anomaly detection in autonomous robots and provides the following main original contributions:

- A new VAE architecture and a new training method, which induces a progress-based latent space that is suitable to detect anomalies (Sections 3.3 and 3.4).
- A new online and a new offline anomaly detection algorithms that require as little as one execution labeled as nominal (Sections 3.5 and 3.6).
- An experimental evaluation on datasets collected from real robots involved in three applications requiring LTA (Section 4).

2 Related Work

2.1 Anomaly Detection

Anomaly detection approaches in robot systems can be divided into three broad categories: model-based, knowledge-based, and data-driven [12]. Model-based approaches require explicit analytical models (i.e., mathematical equations) of robot systems and therefore need expert knowledge to be built. Knowledge-based approaches typically associate each known fault to a detection rule which is triggered when a specific behavior is observed. Data-driven approaches are based on (usually probabilistic) descriptions of behaviors or faults that are automatically learnt from observations of the system. Their advantage is that they do not need

any explicit prior knowledge of the system nor of the faults. Since in this paper we propose a data-driven approach, the rest of this section surveys this category.

Online data-driven methods are typically used for autonomous robots. In their basic form, they generate probabilistic representations of robots' behaviors in real-time, from data streams, and use them to statistically differentiate potential faults from nominal behaviors. Some approaches (e.g., [9]) adopt supervised machine learning methods to classify data acquired in real-time from a robot. Supervised methods need fully labeled data for training, which are not always available; moreover, they assume to already know all possible kinds (i.e., classes) of anomalies that will ever occur. Hence, recent developments shift to unsupervised and semi-supervised learning. Unsupervised methods (e.g., [13]) do not require a labeled training set but have the drawback of relying on the assumption that anomalies are rarely occurring. Semi-supervised methods (e.g., [3, 21–23]) relax the assumption on the rarity of anomalies, but require labeled instances for the nominal class (which are usually easier to collect with respect to, w.r.t., anomalous ones) [7]. The method we propose in this paper is trained with unlabeled data, but needs at least one execution labeled as nominal in order to detect anomalies. In this sense, we call it minimally supervised.

Recently, deep learning models have been employed to re-address several spatio-temporal modeling tasks, including those relative to anomaly detection in robotics, providing significant improvements over classical state-of-the-art methods. We first introduce some background on these techniques and then survey the literature most relevant for our contribution.

2.2 Autoencoders and Variational Autoencoders

Autoencoders (AEs) [11] are particular kinds of artificial neural networks which are trained to reconstruct their input, in a self-supervised manner. An AE is composed of an *encoder* network and a *decoder* network. The encoder takes as input the training data $x \in \mathbb{R}^d$, where d is the dimension of the data, and compresses these data into a *latent space* $z \in \mathbb{R}^h$, where h is the dimension of the encoding, usually $h < d$. Then, the decoder tries to map back the latent internal representation z to the original input space $\hat{x} \in \mathbb{R}^d$, through reconstruction. The encoder structure can be considered as a bottleneck, in which data pass and are compressed to extract a meaningful encoded representation. The decoder does the opposite. The two networks are characterized by f_ϕ, the encoding function, and f_θ, the decoding function, where $f_\phi : \mathbb{R}^d \to \mathbb{R}^h$ and $f_\theta : \mathbb{R}^h \to \mathbb{R}^d$. Finding weights (parameters) ϕ and θ for the two functions can be done by backpropagation, minimizing the loss function $\mathcal{L}_{AE}(x, \hat{x}) = \| x - \hat{x} \|^2$, called *reconstruction error*, given input x and model output $\hat{x}$.

Variational Autoencoders (VAEs) [14] differ from plain AEs in the fact that they assume the existence of a probabilistic model parametrized by the latent variable $z \in \mathbb{R}^h$ that generates the observed input values $x \in \mathbb{R}^d$. Being z latent (i.e., hidden), we can infer its characteristics by computing the posterior $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$; unfortunately, computing the marginalization $p(x) = \int p(x|z)p(z)dz$

at the denominator is intractable when z is high-dimensional. *Variational inference* can be used to overcome this issue by approximating $p(z|x)$ with a distribution $q(z|x)$ which is tractable and by minimizing their *KL-divergence* $D_{KL}(q(z|x) \parallel p(z|x))$ to ensure that $q(z|x)$ is similar to $p(z|x)$. By replacing $p(z|x)$ in $D_{KL}(q(z|x) \parallel p(z|x))$ with $\frac{p(x,z)}{p(x)}$, the minimization problem becomes equivalent to the maximization of $\mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x) \parallel p(z))$, where the first term represents the reconstruction likelihood (analogous to the reconstruction error in AEs), while the second term ensures that the learned distribution $q(z|x)$ is similar to the true prior distribution $p(z)$ (with the effect of regularizing the latent variable z). The distributions $q(z|x)$ and $p(z|x)$ can be parametrized by means of two artificial neural networks which can be considered as encoder (with weights ϕ), and decoder (with weights θ), respectively. Finding weights parameters ϕ and θ can be done by backpropagation, minimizing the loss function $\mathcal{L}_{VAE}(x) = D_{KL}(q_\phi(z|x) \parallel p_\theta(z)) - \mathbb{E}_{q_\phi(z|x)}(\log p_\theta(x|z))$, which represents the variational lower bound of the data x according to the Jensen's inequality (see [14] for full details). The posterior $q_\phi(z|x)$ is usually assumed to be normally distributed with parameters $\mathcal{N}(\mu_z, \Sigma_z)$, while a common choice for the prior distribution $p_\theta(z)$ is an isotropic normal distribution $\mathcal{N}(0, I)$.

2.3 AEs and VAEs for Anomaly Detection

The main idea behind the current use of AEs for anomaly detection is to train them only on nominal data so that they will not be able to accurately reconstruct anomalous behaviors (that the AEs have never seen), which will thus produce high reconstruction errors. AEs have been widely used for anomaly detection on time series, and more rarely on data coming from robots. For example, authors of [19] propose an LSTM (Long Short-Term Memory) based encoder decoder (EncDec-AD) that learns to reconstruct nominal time series and thereafter uses reconstruction error of observed samples to detect anomalies. Similarly, [30] proposes a Multi-Scale Convolutional Recurrent Encoder-Decoder (MSCRED) to perform anomaly detection and diagnosis in multivariate time series data. One significant example of application of AEs to detect anomalies in robots is [20], where the authors propose to convert sensor logs into images and then use a convolutional AE to detect anomalous behaviors resulting from cyber-security attacks.

After the introduction of VAEs, a lot of interest has developed around this new framework due to the continuity of its latent space and to its ability of producing probabilistic anomaly scores (see below) which are in general more powerful than AEs' reconstruction error. In [22], the authors propose an LSTM-VAE-based detector using a reconstruction-based anomaly score and a state-based threshold to detect anomalies for robot-assisted feeding, while authors of [27] apply the STORN model [4] to anomaly detection by introducing a trending prior on the latent representation. The AE/VAE-based approaches described so far use some variants of the reconstruction error as anomaly score. However, after its adoption in [1], the *reconstruction probability* $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$ has

become a more popular anomaly score than the reconstruction error. In this case, for each sample from the encoder, the probabilistic decoder outputs the mean and variance parameters of the reconstruction instead of the reconstructed value itself. Then, the reconstruction probability is calculated using the output parameters of the encoder, given the original input as a sample. For instance, in [8], a sliding-window convolutional variational autoencoder (SWCVAE) is proposed, which can perform real-time anomaly detection on multivariate time series acquired from an industrial robot. Another example of the use of the reconstruction probability is in [24], which presents a variational recurrent autoencoder with attention.

All the above approaches, and many others, are trained on nominal data in a semi-supervised fashion. However, some AE/VAE-based approaches trained in an unsupervised fashion exist, such as [29], which proposes an unsupervised anomaly detection algorithm based on a VAE to detect anomalies on web servers usage time series. Also in this case, detection exploits the reconstruction probability. Another interesting approach has been proposed in [25], whose model is trained in a fully unsupervised fashion and applied to univariate healthcare time series in which anomalies are detected directly in the latent space by computing the Wasserstein distance between a test sample latent representation and other encoded samples in the test set.

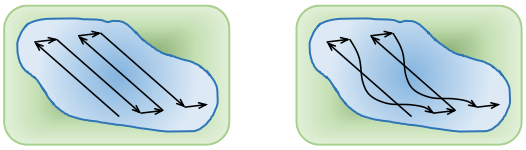
Our method employs VAEs but differs from semi-supervised approaches in requiring only minimal supervision, and from unsupervised ones in modelling multivariate time series and not assuming rarity of anomalies.

3 The Proposed Method

3.1 Problem Definition

We represent by $\boldsymbol{O} = [\boldsymbol{o}_1, ..., \boldsymbol{o}_n]$ a d-dimensional time series composed of n observations of a robot system performing a task, where each observation $\boldsymbol{o}_t$ is a d-dimensional vector representing the multivariate (multi-valued) observation of the robot at (discrete) time step t. Typically, $\boldsymbol{o}_t$ is extracted from sensors logs of the robot. We assume to know at least one nominal execution $\boldsymbol{O}^N = [\boldsymbol{o}^N_1, ..., \boldsymbol{o}^N_{n_N}]$ corresponding to the robot correctly performing its task. If the robot can display k different types of nominal behaviors when performing its task, we assume the availability of at least a nominal execution $\boldsymbol{O}^N$ for each of them. The observed behavior of the same robot along some time period is denoted by $\boldsymbol{O}^O = [\boldsymbol{o}^O_1, ..., \boldsymbol{o}^O_{n_O}]$. Informally, we consider an observed run $\boldsymbol{O}^O$ to be anomalous if there exists a time step $t_A \leq n_O$ from which on $\boldsymbol{O}^O$ starts to differ from one of the nominal executions $\{\boldsymbol{O}^N\}$. The details on the way in which an observation is classified as anomalous are discussed in Sections 3.5 and 3.6.

If we consider $\boldsymbol{O}^O$ as a (possibly infinite) data stream, *online anomaly detection* at time step t is the task of classifying the portion of the stream up to t as anomalous or non-anomalous w.r.t. $\{\boldsymbol{O}^N\}$.



(a) (b)

Fig. 1: Schematic representation of water drone nominal (a) and anomalous (b) behaviors, as they appear in the original data.

Given a finite time series of observations $\boldsymbol{O}^O$, *offline anomaly detection* is the task of classifying the whole behavior displayed by the robot in $\boldsymbol{O}^O$ as anomalous or non-anomalous w.r.t. $\{\boldsymbol{O}^N\}$.

3.2 Running Example

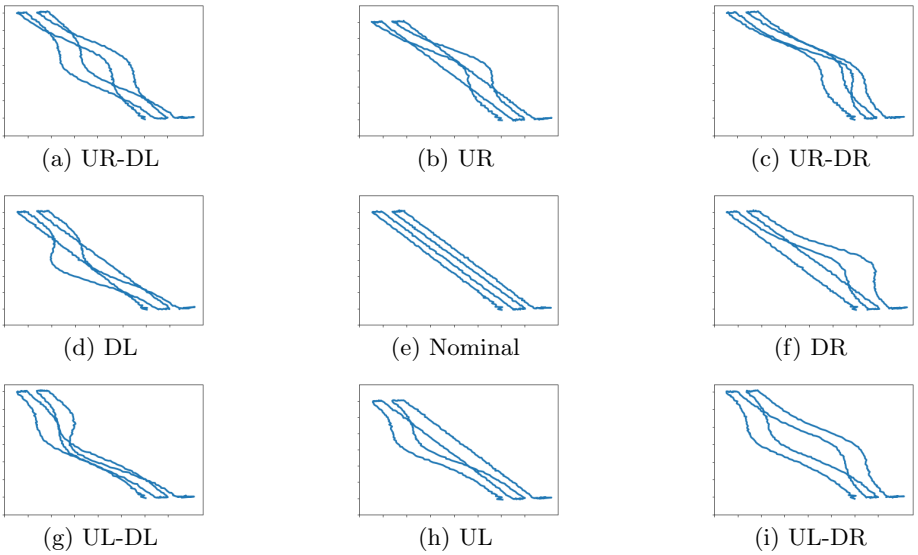
As a running example we consider a synthetic dataset derived from that used in [3] and collected from a water drone (called Platypus) while performing a coverage task (Fig. 1a) on a lake to collect water samples. The experimental setting takes inspiration from the H2020 EU project INTCATCH[1], devoted to develop user-friendly water monitoring strategies and systems for improving the quality of surface water in lakes and rivers [6]. Starting from some real runs, we generate 340 runs in which the following variables are recorded at each time step (1 Hz): heading, speed, acceleration, power signals to the left and right propellers, latitude, and longitude. From the anomalies observed in the real data (see [3]), which present a recurring curve leaning to the left in the descending traits (Fig. 1b), we incorporated in our simulated dataset 8 possibles nuances of similar anomalies (Fig. 2).

3.3 Network Architecture

We present a new VAE architecture (Fig. 3) in which we replace the typical feed-forward layers with 1D-convolutional (Conv-1D) and Bidirectional LSTM (Bi-LSTM) layers, which are better suited to represent the temporal dependency of multivariate time series collected from robots' sensors. Moreover, the use of Conv-1D and Bi-LSTM layers allows our network to model very long runs.

Before being passed to the network for training, runs in the training set are standardized, then a Gaussian noise n_σ is added to each non-padded value of the runs in the training set X_{train} (see next section for a detailed explanation) in order to perform training using the denoising principle [28]. The noise variance σ is set to 1. The noise-corrupted input $\widetilde{X}_{train}$ is passed to stacked pairs of Conv-1D and MaxPooling layers, then a Bi-LSTM takes the output of the previous levels of convolution and returns the concatenated final states $[\overrightarrow{h_f}, \overleftarrow{h_b}]$ (i.e., the contexts

[1] http://www.intcatch.eu/



(a) UR-DL (b) UR (c) UR-DR

(d) DL (e) Nominal (f) DR

(g) UL-DL (h) UL (i) UL-DR

Fig. 2: Simulated trajectories of a water drone. The nominal behavior of the robot is in the center (e): the robot starts from the bottom and goes up and down moving rigthward. (a) depicts the behavior of leaning to the right in the upward segments and to the left in the downward ones (UR-DL), (b) the behavior of leaning to the right in the upward segments (UR), (c) the behavior of leaning to the right in both the upward and the downward segments (UR-DR), (d) the behavior of leaning to the left in the downward segments (DL), (f) the behavior of leaning to the right in the downward segments (DR), (g) the behavior of leaning to the left in both the upward and the downward segments (UL-DL), (h) the behavior of leaning to the left in the upward segments (UL), (i) the behavior of leaning to the left in the upward segments and to the right in the downward ones (UL-DR).

in both directions). This concatenation is passed to 2 fully connected layers z_{mean} and $z_{log-var}$, which learn the parameters μ_z and σ_z^2 of the approximate posterior distribution $q_\phi(z|x)$, where ϕ is the matrix of the encoder's weights. These last two layers, together with the next one, represent the core of our VAE, in which a sample is extracted from a multivariate Gaussian distribution $\mathcal{N}(\mu_z, \sigma_z^2 I)$ by means of the reparametrization trick $z = \mu_z + \sigma_z \cdot \epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$, resulting in the value of the latent variable z of the layer $z_{sampled}$, that will be used by the decoder to reconstruct the input. The decoder network is composed of an initial fully-connected layer which reshapes $z_{sampled}$ in order to be compatible with the following layers. This is passed to the decoder Bi-LSTM that computes, for each time step of its input, a new value based on its context. Then, these sequences are passed to the stacked levels of Conv-1D and upsampling layers which expand the number of time steps of the sequences to match the original

length while reconstructing their values. The final output of the last upsampling layer is used to compute the loss function $\mathcal{L}_{VAE}$ (see Section 2.2) to update the network weights by backpropagation. According to the denoising criterion, the network is trained to output the reconstruction X_{rec}, as an approximation of the original non-corrupted input X_{train}. From now on, we will refer to our VAE architecture described above as *Incr-VAE* (code is publicly available[2]).

3.4 Incremental Training

Instead of training Incr-VAE on runs corresponding to complete task executions performed by the robot or on windowed slices of such executions, as it would be the norm, we originally build a training dataset that includes also *incomplete* task executions in an incremental manner. The detailed procedure is reported in Algorithm 1. Given a batch $\boldsymbol{\Omega}$ of B unlabeled task executions and chosen an increment τ, we represent by X_{train} the (initially empty) training set (line **1**). Each multivariate time series $\boldsymbol{O}$ in $\boldsymbol{\Omega}$ is inserted in X_{train} (lines **2-10**) at different stages of completion by progressively including τ additional time instants (lines **3-4**). Incomplete runs are zero-padded in oder to have the same length of complete ones (lines **5-7**). Complete runs are also added to X_{train} (line **11**). In our experiments, we use $\tau = 10$ and a maximum length of $T = 500$ (shorter complete runs are zero-padded).

When using our incremental training approach based on the augmented training set X_{train}, the VAE induces a *progress-based* latent space, where runs at different levels of completion are encoded in different regions of the space. Fig. 4 shows the first 3 principal components of the latent space of our running example extracted using Principal Component Analysis (PCA) [5]. We use the same PCA projection also for the following figures. In the rightmost part of Fig. 4 it can be noted how incomplete runs containing just the first few observations are all represented in the same spot of the latent space as they are all indistinguishable from each other. Then, as anomalous executions start to deviate from the expected rectilinear paths (Fig. 2), three different bundles start to emerge, which represent the behaviors showing the same attitude in the first upward segment (e.g., UL, UL-DR, and UL-DL all lean towards left). When the water drone

[2] https://github.com/lucabonali/Incr-VAE

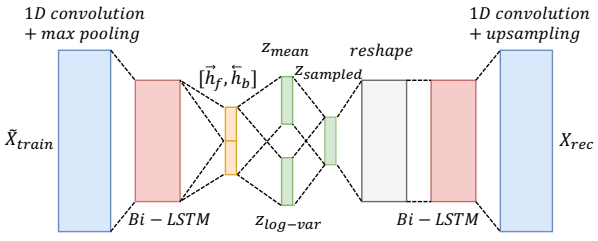


Fig. 3: Incr-VAE architecture.

Algorithm 1: Incremental Training

Input: $\boldsymbol{\Omega} = \{\boldsymbol{O}_1, ..., \boldsymbol{O}_B\}$, increment τ
Output: X_{train}

$X_{train} \leftarrow \{\ \}$
forall $\boldsymbol{O} = [\boldsymbol{o}_1, ..., \boldsymbol{o}_T] \in\ \boldsymbol{\Omega}$ **do**
  for $i = 1, ..., \lfloor T/\tau \rfloor$ **do**
    $X_{tmp} \leftarrow \{\boldsymbol{o}_1, ..., \boldsymbol{o}_{i*\tau}\}$
    for $j = (i * \tau) + 1, ..., T$ **do**
      $append(X_{tmp}, 0)$ `// zero-padding`
    end
    $X_{train} \leftarrow X_{train} \cup X_{tmp}$
  end
end
$X_{train} \leftarrow X_{train} \cup \boldsymbol{\Omega}$

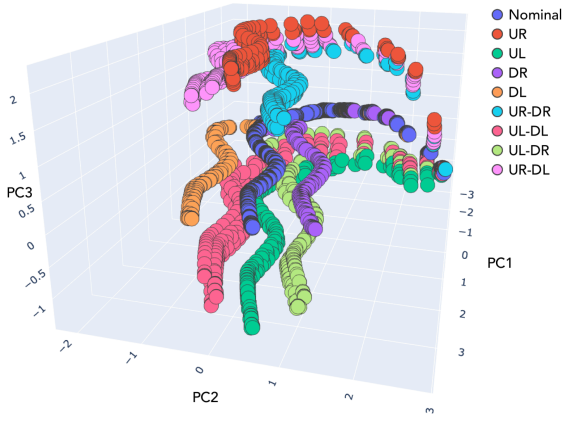
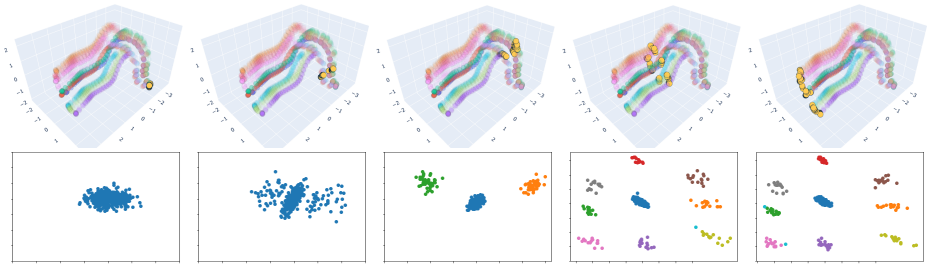


Fig. 4: Water monitoring robot latent space ($t = \tau$ on the right in the background, completed runs at $t = T$ in the foreground).

reaches the beginning of the first downward segment, the three bundles split and become nine as at this point all the different behaviors are distinct. Please note that in Fig. 4 (and in the following figures) the different behaviors have been depicted using different colors just for visualization clarity: the data used for training are unlabeled. From the figure, it appears clearly how our method leads to a structured latent space in which different behaviors are well separated.

As a consequence of the fact that our method induces a latent space which encodes both incomplete and complete executions, the same network trained only once can be used for both online and offline anomaly detection. Moreover, as a consequence of the fact that the incremental training involves some data augmentation, fewer runs are needed for training (as few as 6 runs, in our experiments) w.r.t. training VAEs in the standard way.



(a) t = 10 (b) t = 30 (c) t = 80 (d) t = 200 (e) t = 500

Fig. 5: Water monitoring robot latent space evolution (non-transparent encodings represent the slice to which the clustering refers to).

3.5 Online Anomaly Detection

Assuming an Incr-VAE trained as illustrated above and the availability of at least one nominal execution for each of the k different types of nominal behaviors for the robot performing its task, online anomaly detection is performed by partitioning the latent space into nominal and anomalous regions according to the provided nominal executions $\{\boldsymbol{O}^N\}$ and by testing, at runtime, to which region a new incoming partial run belongs to. Note that our method detects anomalies in the latent space, differently of other methods based on AEs and VAEs (e.g., [19, 20, 22]) that detect anomalies thresholding the reconstruction error or probability.

Given X^N, obtained by applying Algorithm 1 to $\{\boldsymbol{O}^N\}$, the latent space segmentation (Algorithm 2) is performed offline (after training) using the DBSCAN algorithm [18], a density-based clustering algorithm that relies on the assumption that clusters are contiguous regions of high point density, separated from other clusters by regions of low point density. Clustering (line **8**) is performed on each "slice" (line **2**) of the latent space $z_{sampled}$ (i.e., on each set of points corresponding to runs with the same progress (line **5**)) with the addition of the encodings of the nominal executions $\{\boldsymbol{O}^N\}$, zero-padded in order to represent the same level task completion (line **6-7**). Clusters containing points belonging to nominal executions are considered *nominal regions* (lines **10-11**), while clusters not containing points from nominal executions, outliers, and the rest of the latent space are considered as *anomalous regions*. Fig. 5 shows how clusters evolve at different slices for the water monitoring robot running example.

At runtime (Algorithm 3), an incoming incomplete run $\boldsymbol{O}$ that needs to be tested for abnormality is firstly standardized (w.r.t. the mean and standard deviation used for the standardization of the training set) and zero-padded (line **1-2**), then it is encoded into its latent representation $\hat{z}$ (line **5**). The cosine similarity between $\hat{z}$ and the encodings of all the runs in the same slice is computed and if $\hat{z}$ is within a distance of ϵ (i.e., DBSCAN's threshold on the maximum distance between two samples for being considered as neighbors of each other, $\epsilon = 0.5$ is the default value we use in our experiments) from an encoding belonging to

Algorithm 2: Latent Space Segmentation

Input: increment τ, training set X_{train}, X^N (output of Algorithm 1 on $\{\boldsymbol{O}^N\}$), VAE encoder f_ϕ
Output: $\mathfrak{R}^N$

1 $\mathfrak{R}^N \leftarrow \{\ \}$ `// nominal region`
2 **for** $i = 1, ..., \lfloor T/\tau \rfloor$ **do**
3 $\quad X^{(i)}_{train} \leftarrow x \in X_{train} \mid progress = i * \tau$
4 $\quad X^{(i)N} \leftarrow x \in X^N \mid progress = i * \tau$
5 $\quad Z^{(i)}_{train} \leftarrow f_\phi(X^{(i)}_{train})$
6 $\quad Z^{(i)N} \leftarrow f_\phi(X^{(i)N})$
7 $\quad Z^{(i)} \leftarrow Z^{(i)}_{train} \cup Z^{(i)N}$
8 $\quad C^{(i)} \leftarrow DBSCAN(Z^{(i)})$
9 $\quad$ **forall** $c \in C^{(i)}$ **do**
10 $\quad\quad$ **if** $(Z^{(i)N} \cap c) \neq \varnothing$ **then**
11 $\quad\quad\quad \mathfrak{R}^N \leftarrow \mathfrak{R}^N \cup c$
12 $\quad\quad$ **end**
13 $\quad$ **end**
14 **end**

a nominal region, the partial run is considered nominal, while an anomaly is detected otherwise (lines **9-11**). For test runs whose progress is not a multiple of τ (line **6**), the cosine similarity is computed w.r.t. the two slices immediately preceding and following (lines **14-20**). In our experiments, we use training datasets containing few hundreds of executions, hence we perform linear search at runtime; for larger datasets it may be worth considering nearest neighbor search algorithms with sub-linear time complexity, such as space partitioning (e.g., the K-D trees) or Locality-Sensitive Hashing (LSH) [26].

3.6 Offline Anomaly Detection

Assuming the availability of an Incr-VAE trained as discussed in Sections 3.3 and 3.4, offline anomaly detection is obtained by performing DBSCAN on the last slice (the one containing the encodings of complete executions) of the latent space with the addition of the encodings of the nominal executions and the encoding $\hat{z}$ of the run under scrutiny. As the DBSCAN algorithm either assigns each point to a cluster or treats it as an outlier, in case $\hat{z}$ belongs to a cluster containing the encoding of a nominal run, the behavior will be considered nominal, anomalous otherwise. In case a domain expert provides also runs labeled as anomalous and $\hat{z}$ belongs to a cluster containing one such anomalous run, it will also be possible to specify the nature of the anomaly. Outliers are considered as generic anomalies.

4 Experimental Results

In this section we present the results obtained by detecting anomalies in three different datasets collected from real robots.

Algorithm 3: Online Anomaly Detection

Input: increment τ, nominal region $\mathfrak{R}^N$, test run $\boldsymbol{O} \leftarrow [\boldsymbol{o}_1, ..., \boldsymbol{o}_t]$, threshold ϵ, VAE encoder f_ϕ
Output: $flag \in \{0, 1\}$

for $j = t+1, ..., T$ **do**
  $append(\boldsymbol{O}, 0)$ `// zero-padding`
end
$flag \leftarrow$ `True` `// anomaly flag`
$\hat{z} \leftarrow f_\phi(\boldsymbol{O})$
if $t\%\tau = 0$ **then**
  $i \leftarrow t/\tau$
  forall $z \in Z^{(i)}$ **do**
    if $Cosine(\hat{z}, z) \leq \epsilon \ \wedge \ z \in \mathfrak{R}^N$ **then**
      $flag \leftarrow$ `False`
    end
  end
else
  forall $i \in \{\lfloor t/\tau \rfloor, \lceil t/\tau \rceil\}$ **do**
    forall $z \in Z^{(i)}$ **do**
      if $Cosine(\hat{z}, z) \leq \epsilon \ \wedge \ z \in \mathfrak{R}^N$ **then**
        $flag \leftarrow$ `False`
      end
    end
  end
end

We use two common metrics in the field of anomaly detection, namely, *alert delay* and *false positive rate*. Given an anomaly occurring at time step t_A, the alert delay d_A is computed as $d_A = t - t_A$ where $t \geq t_A$ is the time step at which the occurrence of the anomaly is detected by a method. Given the set W containing the time steps at which a method reports an anomaly, the false positive rate (FPR) is computed as the fraction of the time steps preceding the actual occurring of an anomaly which have been identified as anomalous $FPR = \frac{|t \in W,\, t < t_A|}{t_A}$. The use of just the FPR without the TPR is meaningful since the case in which one could obtain a $FPR = 0$ by always saying that everything is nominal is prevented from the fact that in that case the *alert delay* would result to be substantially increased.

We compare our system against a baseline and four other methods proposed in the literature for online anomaly detection in robotics:

- A one-class support vector machine (OSVM) trained with a sliding window size $w = 10$.
- HMM-H [3], an HMM-based anomaly detector which detects anomalies by computing the Hellinger distance between the probability distribution of observations made in a sliding window and the corresponding nominal emission distribution. HMM parameters are chosen minimizing the BIC score, the window size is set to 10 and the 3σ-rule is used to select the detection threshold.

- ENC-DEC AD [19], the first work that proposed to employ LSTM-based AEs for anomaly detection on time series. ENC-DEC AD learns to reconstruct nominal time series and then uses the reconstruction error on unseen executions to detect anomalies. We optimize the detection threshold τ by maximizing F_β (a function of precision and recall), as suggested by the authors of the method.
- Conv-AE [20], based on transforming system logs into images, which are then used to train a convolutional (2D) AE. As for ENC-DEC AD, the reconstruction error on unseen executions is used to detect anomalies. We optimize the sensitivity parameter z for each dataset according to authors' suggestions (i.e., $z \in [0, 3]$).
- LSTM-VAE [22], a state-of-the-art LSTM-based VAE with a varying state-based threshold obtained by employing a progress-based prior and a support vector regressor (SVR) for threshold prediction. We optimize the sensitivity parameter c for each dataset.

While our method is trained in a minimally supervised fashion (i.e., knowing the nominal label for k executions, where k is the number of different types of nominal behaviors), all four competitors are trained in a semi-supervised fashion (i.e., assuming that all training data are nominal) on the same datasets. Given a dataset, training our method takes some minutes on a commercial laptop, while online and offline detection of anomalies takes few milliseconds.

4.1 Water Monitoring Robot Dataset

This is the same dataset introduced in the running example. Our Incr-VAE network is trained using the Adam optimizer with one level of convolution in the encoder and decoder, $h = 20$ as latent dimension, 10 filters for the convolution, and 10 as convolution window. A single nominal run is used to partition the latent space.

4.2 Patrolling Robot Dataset

This publicly available[3] dataset has been collected within the scope of the STRANDS[4] project [10], where an autonomous robot called SCITOS-G5 performs a patrolling task in a small office every 5 minutes. A complete description of the data collection process is provided in [15]. We consider a total of 463 different executions sub-sampled at 1 Hz. We restrict the set of available sensors to those that are intuitively useful for anomaly detection, namely, robot location (x and y) and robot and camera headings. After a visual inspection of the logs, only one type of nominal behavior ($k = 1$) has been assumed and a subset of the runs have been manually labeled as anomalous in order to be used for testing. Examples of anomalies are deviations from the predefined path and incorrect uses

[3] https://lcas.lincoln.ac.uk/nextcloud/shared/datasets/
[4] http://strands.acin.tuwien.ac.at/

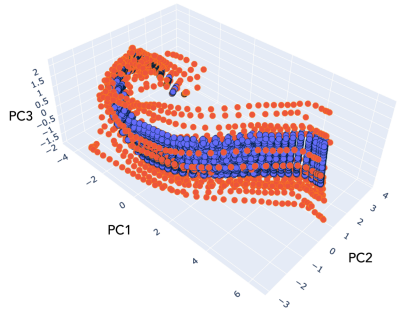


Fig. 6: Patrolling robot latent space ($t = \tau$ in the background, completed runs at $t = T$ in the foreground). Nominal runs in blue, anomalies in red.

of the RGB-D camera when checking for the presence of intruders. The latent space of our VAE induced by this dataset is shown in Fig. 6. Anomalies (in red) are clearly detached from the central nominal bundle (in blue). The network is trained using the Adam optimizer with 2 levels of convolution on both encoder and decoder, 10 as number of filters and window of convolution, $h = 20$ as latent dimension. A single nominal run is used to partition the latent space.

4.3 Assistive Robot Dataset

The third dataset has been collected within a project developing an innovative multi-actor platform, centered around an autonomous robot for supporting the independence of elderly people living alone at home [17]. The socially assistive autonomous mobile robot is a human-sized robot which moves in domestic environments, which represent a typical context for LTA. To localize the elder it has to assist, the robot starts from its charging base and visits in sequence different rooms of the house until the person is found. Data are collected in a 9-day experiment simulating the same number of interventions performed in a month of use of this social assistive robot, thus performing multiple interventions per day. The dataset contains 238 runs, each one composed of a sequence of observations collected at 1 Hz including: heading, speed, acceleration, position w.r.t. the x-axis, and position w.r.t. the y-axis. The dataset presents $k = 4$ different types of nominal behaviors (corresponding to reaching one of four different rooms) for each of which a domain expert provided a single nominal run. Out of the 238 runs, 12 have been classified by a domain expert as anomalies which are used for testing and correspond to the robot not being able to return back to its charging station because it remains stuck in forniture, the robot departing from its nominal trajectory, or the robot moving too fast. We use the same network hyperparameters as in the patrolling robot dataset. The latent space induced by this dataset is in Fig. 7. As said, a single nominal run for each one of the $k = 4$ types of nominal behaviors is used to partition the latent space.

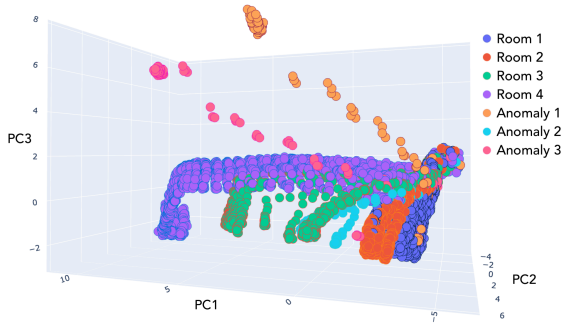


Fig. 7: Assistive robot latent space ($t = \tau$ on the right, completed runs at $t = T$ on the left).

4.4 Results

Alert delays and FPRs are reported in Table 1 and Table 2, respectively. Our method provides the best performance across all datasets despite using just 1 labeled nominal execution for the first two datasets and 4 for the third one.

As highlighted also in [22], the higher alert delay and FPR of ENC-DEC AD have to be attributed to the fact that sometimes the reconstruction error is high also in nominal situations. In fact, depending on the stage of the execution, the reconstruction quality may vary. An example is when spikes (i.e., impulses to make the boat turn) appear on the currents to the motors in the time series of the first dataset, which result in high anomaly scores also in nominal runs. Thanks to its varying state-based threshold, LSTM-VAE is able to overcome the above issue after we set a tolerance value to be added to the state-based threshold of the model to avoid the incorrect detection of the spikes, even though this comes at the cost of a slight worsening of the alert delay. Comparing the two AE-based methods, Conv-AE always outperforms ENC-DEC AD, probably due to the use of convolution, that is more stable and easier to train than recurrent layers. We also note that the VAE-based methods outperform the AE-based ones, as also pointed out in [22]. The high alert delay for HMM-H in the third dataset has to be ascribed to its inability to detect anomalies on the velocity of the robot due to the Markov assumption. To enable HMM-H to detect also such anomalies, velocity should be explicitly modeled as an additional dimension of the multivariate time series as done in [3]. OSVM's higher FPR and bad performance in general on the second dataset result from its inability to represent the portion of time series inside the window as an actual sequence instead of as a feature vector without any time dependence. Moreover, it is difficult to adjust OSVM's threshold after training as it coincides with the SVM decision boundary.

We finally remark that our method reaches an accuracy of 100% when performing offline anomaly detection on the three datasets.

4.5 Latent Space Analysis

As said, instead of training Incr-VAE on runs corresponding to complete task executions performed by the robot or on windowed slices of such executions, as it usually happens, we originally build a training dataset that includes also *incomplete* task executions in an incremental manner. Here, we investigate how the latent space would be learnt in the two usual cases just mentioned for the water monitoring robot dataset. We include in our comparison also the latent space induced by LTSM-VAE, the other method based on a variational autoencoder we consider in our experimental assessement. As a reference, the latent space of Incr-VAE when trained in the incremental way is reported in Fig. 4.

Complete Executions Fig. 8 depicts the latent space resulting from training Incr-VAE on complete executions only. As it can be seen, our architecture manages to encode very long sequences in a meaningful way (i.e., nominal runs and each anomaly are clearly separated). Note that, as one would expect, this arrangement coincides also with the last slice of Fig. 5 (i.e., the one encoding complete executions). Looking at Fig. 8 it can be noted a very interesting feature: not only each anomaly type has its own "cluster" clearly detached from the others, but they are arranged in a meaningful and intuitive way. Take for example UL and DL, if we start from the nominal area (in the center) and proceed along a line passing between UL and DL, we reach the area of latent space in which UL-DL is located, i.e., the anomalous behavior affected by both UL and DL anomalies (note that this desired feature is true also when Incr-VAE is trained in the incremental way). One drawback of training on complete runs is that in this case the online anomaly detection problem reduces to the offline one, as complete executions are required to be provided as input to the Incr-VAE. Another drawback is that, according to our experiments, in this way Incr-VAE is harder to train (almost five times more epochs compared to incremental training).

Sliding Window When trained using a sliding window, the latent space of Incr-VAE is not structured as in Fig. 4, where there is a clear concept of beginning and end of a run. Our method, when trained incrementally, can tell if a partial run is nominal or anomalous up to a given point; when using sliding windows, it

	Platypus	**SCITOS-G5**	**Assistive robot**
OSVM	1.375 (2.18)	60.5 (65.19)	5.0 (7.07)
HMM-H	3.48 (2.11)	8.38 (2.18)	46.17 (58.91)
ENC-DEC AD	19.23 (11.60)	10.66 (13.42)	6.83 (6.47)
Conv-AE	6.22 (5.16)	3.40 (3.65)	6.42 (7.48)
LSTM-VAE	3.68 (2.25)	4.88 (6.33)	5.42 (6.81)
Incr-VAE	**0.5** (2.18)	**3.13** (5.56)	**3.34** (4.71)

Table 1: Alert delay results.

	Platypus	SCITOS-G5	Assistive robot
OSVM	0.123 (0.200)	0.180 (0.204)	0.130 (0.130)
HMM-H	0.019 (0.053)	0.169 (0.120)	0.073 (0.103)
ENC-DEC AD	0.107 (0.151)	0.142 (0.245)	0.200 (0.200)
Conv-AE	0.049 (0.086)	0.109 (0.058)	0.170 (0.100)
LSTM-VAE	**0.0** (0.0)	0.191 (0.197)	0.084 (0.119)
Incr-VAE	**0.0** (0.0)	**0.065** (0.115)	**0.007** (0.009)

Table 2: FPR results.

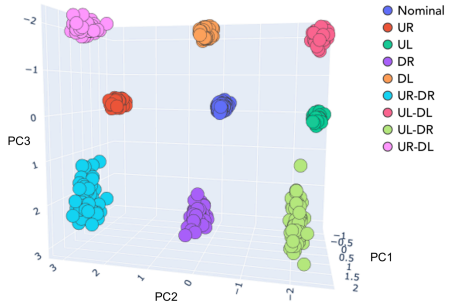


Fig. 8: Incr-VAE latent space when trained on complete runs.

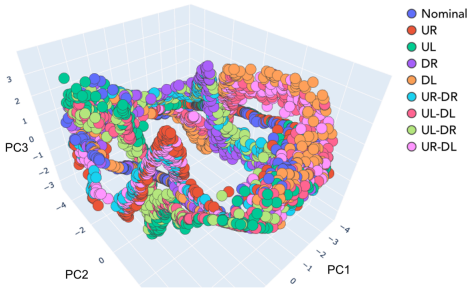
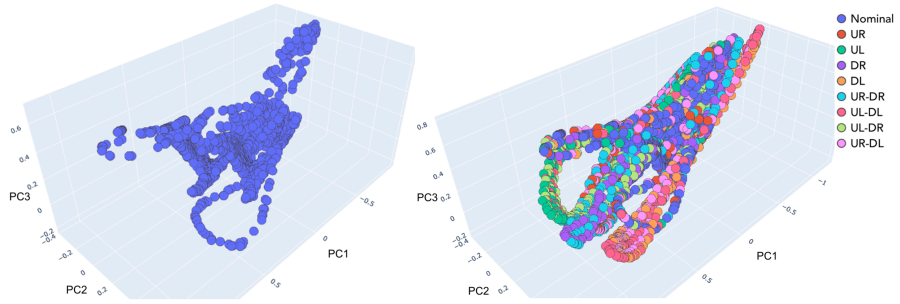


Fig. 9: Incr-VAE latent space when trained using a sliding window.

would tell only if a specific window is anomalous. For example, Fig. 9 depicts the latent space when Incr-VAE is trained using a sliding window of 10 time-steps. Some structure is still present (as reflected by the groupings of the colors), but most of the interpretability is lost. The temporal progression is also lost and consecutive instances of the same sub-task (e.g., the first and the second ascending traits in Fig. 1a) are now encoded in the same spot. Moreover, our incrementally-trained method can do both online and offline anomaly detection with a single network trained only once. When training using sliding windows, offline anomaly detection could not be performed.



(a) (b)

Fig. 10: LSTM-VAE latent space.

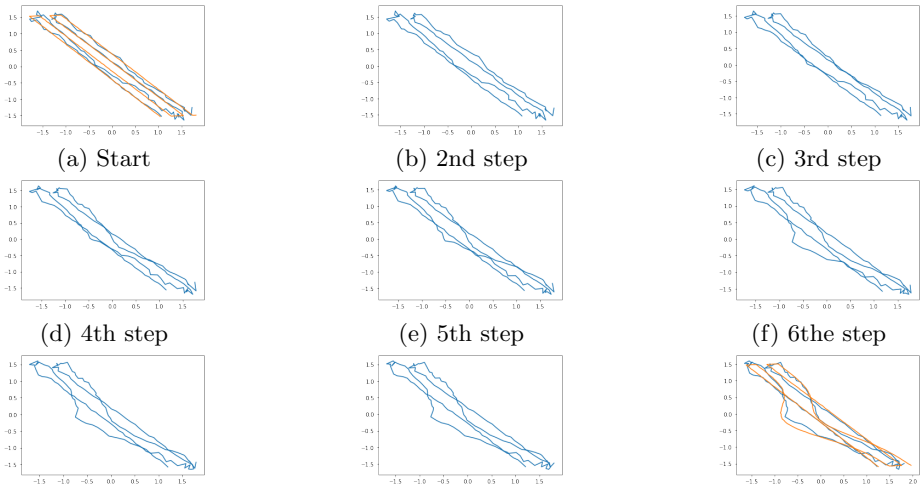
LSTM-VAE Fig. 10 depicts the latent space of LSTM-VAE when trained only with nominal executions of the water drone. Fig. 10a represents the encodings of the training set (i.e., only nominal runs). In order to make LSTM-VAE's latent space comparable to the other ones, Fig. 10b shows the encodings of the training set and of some anomalous runs, in different colors (the model is trained on nominal runs only, then, after training, some anomalous runs have been passed through the encoder). As it can be seen, some structure is present, which is enough for the model to learn good reconstructions and, as a consequence, detect correctly most of the anomalies (as our experimental results show). However, the latent space of LSTM-VAE is less separable, not very interpretable, and a progress-based structure is not present.

4.6 Latent Space Interpolation

One of the most important features of VAEs is the smoothness and continuity of their latent spaces, which means that, for example, by interpolating points between two encoded values that represent two different runs $\boldsymbol{O}_1$ and $\boldsymbol{O}_2$, and generating new runs using the decoder, the in-between generated runs will change smoothly from $\boldsymbol{O}_1$ to $\boldsymbol{O}_2$. In Fig. 11 it is shown the case in which starting from the embedding (i.e., latent representation) of a nominal run for the water monitoring robot dataset, and interpolating towards the embedding of an anomalous one (DL), the reconstructed runs becomes incrementally more anomalous when approaching the anomaly. This feature could be used to generate possible anomalies that could affect a robot system, in order to develop contingency strategies before the anomalies actually occur. Moreover, latent space interpolation could be useful to explain observed anomalies. Both directions are left as future work.

5 Conclusions

In this paper we have presented a new approach based on VAEs for detecting, both online and offline, anomalies in the behavior of autonomous robots.



(a) Start (b) 2nd step (c) 3rd step

(d) 4th step (e) 5th step (f) 6the step

(g) 7th step (h) 8th step (i) end

Fig. 11: Interpolation (reconstructions in blue, input runs in orange).

Our method outperforms other methods that have been recently proposed for anomaly detection in robotics and does so by requiring significantly less labeled data. We have shown how just even a single labeled nominal execution is sufficient for our method to partition a latent space (previously learned in an unsupervised fashion) in a meaningful way for detecting anomalies.

Future work includes employing a Gaussian mixture prior on z to better represent different types of nominal behaviors. Another interesting future direction is employing β-VAEs, as their ability of inducing a disentangled z could lead to an even more intuitive and interpretable latent space. Finally, we plan to apply the proposed approach to other robot applications involving the need of detecting anomalies in the context of LTA.

References

1. J. An and S. Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18, 2015.
2. D. Azzalini, L. Bonali, and F. Amigoni. A minimally supervised approach based on variational autoencoders for anomaly detection in autonomous robots. *IEEE RAL*, 2021. to appear.
3. D. Azzalini, A. Castellini, M. Luperto, A. Farinelli, and F. Amigoni. HMMs for anomaly detection in autonomous robots. In *Proc. AAMAS*, pages 105–113, 2020.
4. J. Bayer and C. Osendorfer. Learning stochastic recurrent networks. https://arxiv.org/abs/1411.7610, 2014.
5. C. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
6. A. Castellini, D. Bloisi, J. Blum, F. Masillo, and A. Farinelli. Multivariate sensor signals collected by aquatic drones involved in water monitoring: A complete dataset. *Data Brief*, page 105436, 2020.

7. V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput Surv*, 41(3):1–58, 2009.
8. T. Chen, X. Liu, B. Xia, W. Wang, and Y. Lai. Unsupervised anomaly detection of industrial robots using sliding-window convolutional variational autoencoder. *IEEE Access*, 8:47072–47081, 2020.
9. A. Christensen, R. OGrady, M. Birattari, and M. Dorigo. Fault detection in autonomous robots based on fault injection and learning. *Auton Robot*, 24(1):49–67, 2008.
10. N. Hawes, C. Burbridge, et al. The STRANDS project: Long-term autonomy in everyday environments. *IEEE RAM*, 24(3):146–156, 2017.
11. G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
12. E. Khalastchi and M. Kalech. On fault detection and diagnosis in robotic systems. *ACM Comput Surv*, 51(1):9, 2018.
13. E. Khalastchi, M. Kalech, G. Kaminka, and R. Lin. Online data-driven anomaly detection in autonomous robots. *Knowl Inf Syst*, 43(3):657–688, 2015.
14. D. Kingma and M. Welling. Auto-encoding variational Bayes. In *Proc. ICLR*, 2014.
15. T. Krajník, J. Fentanes, G. Cielniak, C. Dondrup, and T. Duckett. Spectral analysis for long-term robotic mapping. In *Proc. ICRA*, pages 3706–3711, 2014.
16. L. Kunze, N. Hawes, T. Duckett, M. Hanheide, and T. Krajník. Artificial intelligence for long-term robot autonomy: A survey. *IEEE RA-L*, 3(4):4023–4030, 2018.
17. M. Luperto, J. Monroy, J. Ruiz-Sarmiento, F.-A. Moreno, N. Basilico, J. Gonzalez-Jimenez, and N. A. Borghese. Towards long-term deployment of a mobile robot for at-home ambient assisted living of the elderly. In *Proc. ECMR*, pages 1–6, 2019.
18. M-Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. KDD*, pages 226–231, 1996.
19. P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. LSTM-based encoder-decoder for multi-sensor anomaly detection. In *Proc. ICML Anomaly Detection Workshop*, 2016.
20. M. Olivato, O. Cotugno, L. Brigato, D. Bloisi, A. Farinelli, and L. Iocchi. A comparative analysis on the use of autoencoders for robot security anomaly detection. In *Proc. IROS*, pages 984–989, 2019.
21. D. Park, Z. Erickson, T. Bhattacharjee, and C. Kemp. Multimodal execution monitoring for anomaly detection during robot manipulation. In *Proc. ICRA*, pages 407–414, 2016.
22. D. Park, Y. Hoshi, and C. Kemp. A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder. *IEEE RA-L*, 3(3):1544–1551, 2017.
23. D. Park, H. Kim, and C. Kemp. Multimodal anomaly detection for assistive robots. *Auton Robot*, 43(3):611–629, 2019.
24. J. Pereira and M. Silveira. Unsupervised anomaly detection in energy time series data using variational recurrent autoencoders with attention. In *Proc. ICMLA*, pages 1275–1282, 2018.
25. J. Pereira and M. Silveira. Learning representations from healthcare time series data for unsupervised anomaly detection. In *Proc. BigComp*, pages 1–7, 2019.
26. A. Rajaraman and J. Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

27. M. Soelch, J. Bayer, M. Ludersdorfer, and P. van der Smagt. Variational inference for on-line anomaly detection in high-dimensional time series. In *Proc. ICML Anomaly Detection Workshop*, 2016.
28. P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proc. ICML*, pages 1096–1103, 2008.
29. H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, and H. Qiao. Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications. In *Proc. WWW*, pages 187–196, 2018.
30. C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. Chawla. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. https://arxiv.org/abs/1811.08055, 2018.