Received 24 May 2023, accepted 7 June 2023, date of publication 23 June 2023, date of current version 6 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3289068

RESEARCH ARTICLE

An Experimental Comparison of Anomaly Detection Methods for Collaborative Robot Manipulators

SØREN G. GRAABÆK[1,2], **EMIL VINCENT ANCKER**[3], **ANDREAS RUNE FUGL**[4], **AND ANDERS LYHNE CHRISTENSEN**[1], **(Senior Member, IEEE)**
[1]The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, 5230 Odense, Denmark
[2]Department of Innovation and Strategy, Universal Robots A/S, 5260 Odense, Denmark
[3]Dynatest A/S, 2750 Ballerup, Denmark
[4]Vitec Aloc A/S, 5000 Odense, Denmark

Corresponding author: Søren G. Graabæk (sggr@mmmi.sdu.dk)

This work was supported in part by the Independent Research Fund Denmark under Grant 0136-00251B, and in part by Universal Robots A/S.

ABSTRACT A large number of methods for anomaly detection in robotic manipulation have been proposed, but their applicability and performance in real-world scenarios are often not established. In this paper, we therefore perform an experimental comparison of a broad range of practically applicable methods to detect exogenous anomalies in pick-and-place tasks. We first collect a dataset, which has been made freely available, on a state-of-the-art collaborative robot. The extensive experimental campaign comprises 600 runs under normal operation, and 80 runs where exogenous perturbations are present, and data is collected on the robot's joints, force-torque readings, and the time spent in each program node. Using the data collected during normal operation, we train a set of anomaly detection methods whose computational complexity is low enough to run on resource-constrained robot hardware. We then evaluate the trained methods on the data collected in the perturbed runs and find that several methods can achieve a high anomaly detection performance. We show that exploiting knowledge of the robot's program tree can increase the performance for some types of anomalies. We also observe that performance, in general, deteriorates when the application includes: (i) rarely-visited program branches, (ii) physical contact with the environment, and (iii) stochastic trajectories.

INDEX TERMS Anomaly detection, collaborative robots, semi-supervised learning.

I. INTRODUCTION

With the current rate of technological improvement, collaborative robots (cobots) can be deployed in an increasing broader set of scenarios [1], including in environments with less structure than found in traditional robot manufacturing. Ideally, cobots can easily be reprogrammed, e.g. via drag-and-drop programming, to take on new tasks and collaborate seamlessly with humans [2]. However, when the operating environment is not highly controlled and frequently undergoes modifications, robot applications become vulnerable to exogenous anomalies. Exogenous anomalies can negatively impact a robot's efficiency, and severe anomalies can even cause robot breakdown. By detecting anomalies in a timely and reliable manner, predictive maintenance on the robot or its surroundings can be performed [3], which in turn can lead to improved overall system performance.

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior [4] and only require data from the system under normal operation, which mean that semi-supervised methods can be used to train detectors. It should be noted that more powerful methods exist for fault detection, such as knowledge-based methods and supervised learning-based methods, see for instance [5], [6]. However, such methods are often infeasible in real-world cobot scenarios since they require detailed a priori

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

knowledge of potential faults and perturbations. We therefore focus exclusively on semi-supervised methods.

To the best of our knowledge, a comprehensive comparison of practically applicable anomaly detection methods for cobots has not yet been reported in the scientific literature. In this study, we therefore conduct an experimental comparison of several existing and novel reconstruction-based anomaly detection methods and two instance-based methods. We collect data in a typical pick-and-place application for a general collaborative manipulator, a UR5e from Universal Robots, which has six Degrees of Freedom (DoF). The robot has a set of sensors that enable measurements of the robot's joint configurations, joint velocities, joint currents, and six-dimensional force-torque at the Tool Center Point (TCP). We use these features, combined with a feature describing the time spent in the robot's current program node, and train a total of 15 different anomaly detection models. The discriminatory power of the trained models is then evaluated on eight exogenous anomalies. The anomalies were carefully selected to cover potential issues in real-world collaborative robot applications, including a robot or its tool colliding with a human or an object, wrongly placed objects, changes in the robot's environment, and incorrect pose estimates. The complete dataset is available in [7].

The remainder of the paper is organized as follows: in Section II, we discuss previous studies on anomaly detection and fault detection applied to manufacturing robots. In Section III, we present the existing and novel anomaly detection models that we evaluate and compare. In Section IV, we detail our experimental setup, the robotic hardware used, and the exogenous anomalies considered. In Section V, we present and compare the anomaly detection performance achieved by the respective methods studied. In Section VI, we discuss the results and avenues for future work, and in Section VII, we provide concluding remarks.

II. RELATED WORK

Fault detection and diagnosis approaches are typically divided into three categories [8]: data-driven, model-based, and knowledge-based. Model-based approaches use mathematical models or logic to describe the system and its components. De Luca and Mattone [6] use a model of the robot's dynamic and a linear state observer to detect unknown actuator faults based on the residual between the physical system and the model. They achieve a good sensitivity, with no false positives, using a low-pass filter to remove data noise and a low detection threshold. Knowledge-based approaches typically associate recognized behaviors with predefined known faults and can thus be used for fault diagnoses. Fantuzzi et al. [5] use an expert system to detect and isolate a well-known set of faults using simple logic on the sensor data. A major issue with knowledge-based systems and model-based approaches is that they require a priori knowledge or a reasonably accurate analytical model, which may not be feasible to obtain. Data-driven approaches, on the other hand, are model-free and thus have the potential to detect unknown faults since a system model is learned and adapted through training. Data-driven methods are often computationally expensive to train, but due to a reduced cost of computational power, they are becoming increasingly popular [8]. Below, we review key data-driven methods and past studies on their application in robotics.

Anomaly detection can be used to pinpoint data instances potentially caused by faults in a robotic system. The data type and the nature of the given data determine which detection methods are appropriate and practical to use [4]; e.g., Mahalanobis distances could be used to detect anomalies in datasets composed of normally distributed real numbers, but may not be applicable in other cases. Sathish et al. [9] study the effect of the data source and type of training data on the detection performance for endogenous faults (gearbox, motor, brake, and computer unit) in industrial robots using Principal Component Analysis (PCA). The authors recorded time series containing mechanical measurements, e.g., TCP speed and joint torque, and non-mechanical measurements, e.g., CPU temperature and fan speed. The data was recorded on ten robots performing different tasks in the same application domain and the data was divided into datasets, each consisting of 20 days of recording. The authors' initial assumption was that raw data features are more representative of the current task, whereas data features based on absolute difference capture variation in data and are therefore more representative of the condition of the robot (i.e. *normal* or *faulty*). Their analysis showed that it is indeed possible to use a single reference robot to generate training data for testing multiple scenarios provided that the PCA model is trained on absolute difference data features.

Goncharov et al. [10] present a simple statistical approach for detecting exogenous anomalies in a pick-and-place application on a single robot. Specifically, the authors' aim was to detect a change in the robot's workload using the actuator currents as the only data feature. They calculated a detection threshold based on the summary statistics (minimum, maximum, and mean) for each robot operation. A limitation of the approach is that it can only detect external perturbations that excite the actuator currents above the extreme values.

Borgi et al. [11] built a linear regression model of the relation between features extracted from actuator current and the robot's accuracy, measured by external laser sensors. The authors recorded the actuator energy consumption from a robot moving between two configurations and calculated summary statistics (mean, standard deviation, skewness, and kurtosis). They then analyzed the Pearson correlation between the data features and the robot accuracy and found that the actuator energy and the mean and standard deviation of the actuator currents are strongly anti-correlated with the robot accuracy; a regression model was therefore trained on these features. The authors did not introduce any faults in their setup, but the predicted accuracy was highly correlated with the actual robot accuracy.

The actuator current can be preprocessed in many ways to train models or not processed at all. Sabry et al. [12] used Bode Equation Vector Fitting (BEVF) for fitting the model to the raw power measurements of a six DoF ABB IRB industrial manipulator as a function of the task, speed, load, temperature, and time: $P_t = f(task, speed, load, temp, time)$. The authors collected a time series dataset of the power consumption during two types of tasks using a sampling rate of up to 123 Hz. The first task was a pick-and-place task where the robot lifted a 200 g payload, and the second task was a polishing task. To detect anomalies, an analytical power model was calibrated to the individual joints of the robot. After the joint calibration, a BEVF was fitted for each of the tasks. More specifically, the power pattern was divided into intervals according to any change in the excitation of the joints and a detection threshold was set to $\pm 10\%$ of the reference/model values. The authors introduced three types of fault: (i) a spring was attached between the two links in the elbow joint, (ii) the electrical signal was removed from joint 2 and the drive belt was removed from joint 5, and (iii) an encoder fault on joint 2 was additionally introduced to (ii). The presented method achieved a good fault detection performance on the studied faults. However, the approach is not directly applicable to dynamic task environments since it relies on a fixed and accurate power consumption pattern.

In [13], Wu et al. proposed a generative Bayesian non-parametric Hidden Markov Model (HMM) for encoding spatial and temporal dependencies in robot data in a low-dimensional hidden-state space. The non-parametric method enabled the automatic inference of the number of latent states from recorded data [14]. The authors evaluated the approach on the right end-effector of a Baxter humanoid robot in a human-robot collaborative task. The human was tasked with feeding six common household objects to the robot consisting of box-like shapes and bottles. During the task, seven types of exogenous anomalies were introduced: human collision, tool collision, object slip, human collision with object, wall collision, no object, and others. The anomalies were detected using a six DoF force-torque data feature, a six DoF Cartesian velocity data feature, and two 56 DoF tactile data features, one for each of the left and right tactile sensors. To make the data features more generic, they were normalized within their category, i.e., the three DoF forces were normalized, the three DoF torques were normalized, and so on. Lastly, the normalized data features and the raw data features (except the raw tactile data features) were combined into a single data feature used for training the non-parametric HMM. The authors achieved high sensitivity and good detection performance across varying demonstrations and tasks by adjusting the anomaly detection threshold during the task. The dynamic threshold was a function of the maximum and minimum of the log-likelihood of observations for latent states.

Using statistical methods to obtain a representation of the latent space has also provided good anomaly detection results in other domains, e.g., aviation [15], and the methods are commonly used for feature engineering in computer vision applications [16]. Recent papers [17], [18] report on an attempt to incorporate the statistical latent space methods into reconstruction-based anomaly detection methods. Reconstruction-based methods assume that anomalous data instances lose more information than normal data instances when they are projected to a lower dimension space by an encoder, and hence that anomalous data instances are not properly reconstructed by the decoder. Reconstruction-based methods are often based on Autoencoders (AEs) [16].

Park et al. [17] used a PR2 humanoid robot for a robot-assisted feeding task. The authors developed a real-time anomaly detector to detect 12 types of exogenous anomalies: touch by user, aggressive eating, utensil collision by user, sound from user, face occlusion, utensil miss by user, unreachable location, environmental collision, environmental noise, utensil miss by system fault, utensil collision by system fault, and system freeze. The data consisted of 17 raw sensor features recorded from the robot: joint encoders, current sensors, force sensors, a microphone, and a camera. The presented AE method used variational inference [19], [20] to model the underlying latent space distribution of robot data and a Long Short-Term Memory (LSTM) structure [21] to model temporal dependencies. The AE structure was trained on the 17 raw sensor data using a denoising autoencoder criterion [22], [23] and it outperformed a likelihood-based classifier (a HMM) trained on four hand-engineered features.

In [18], Chen et al. present a recent anomaly detection study on a KUKA KR 6 R900 sixx doing a pick-and-place application. The approach embeds variational inference and a Convolutional Neural Network (CNN) [24] into the AE model instead of LSTM neurons, because the authors claim that the CNN is better at capturing data dependencies. The proposed method achieved promising results but had to be trained offline due to a very large number of trainable parameters. The method was trained on 12 standardized sensor signals sampled at 33.3 Hz. The data features consisted of the six joint positions and currents.

In the following section, we review the theory behind some of the key methods used in the studies discussed above along with anomaly detection methods commonly used in the broader anomaly detection literature [4], [25], [26], [27].

III. ANOMALY DETECTION METHODS COMPARED

In this study, we implement 15 different methods for anomaly detection and all the presented methods are trained semi-supervised. The training data consist of the 25 behavioral features listed in Table 1 (see section IV for elaboration) recorded on a robot during normal operation; and for some of the methods, the two contextual features are also used (see details below). The methods will be evaluated based on their discriminatory power.

TABLE 1. Summary of used data feature. B: Behavioral. C: Contextual.

Feature	Size	B	C	Description
Joint configuration	6	✓		Relates to robot movements.
Joint velocity	6	✓		Relates to robot movements.
Joint current	6	✓		Relates to external wrench and robot movements.
TCP F/T sensor	6	✓		Relates to external wrench.
Timestamp	1		✓	Establishes temporal context.
Program node ID	1		✓	Establishes robot program context.
Time_in_node	1	✓		Provides temporal behavioral information.

A. INSTANCE-BASED METHODS

Instance-based models are popular in the outlier analysis domain because of their simplicity, effectiveness, and intuitive nature [27], [28], [29]. The methods are trained by organizing the training data in memory, and the outlier score is calculated by computing the distance between the query data and the data stored in memory, based on a prespecified distance metric. The methods can be updated online by adding new data instances to the stored data. The outlier score is valid under the assumption that anomalous data instances have a significantly larger distance to their neighbors than normal data instances, measured under the specified distance metric. Thus, the performance of the method depends on the chosen distance metric. Basic instance-based methods have a space and computational complexity of $\mathcal{O}(N)$ [27], where N is the number of training instances. In our implementations, we use a k-dimensional index trees to lower the query and training computational complexity to $\mathcal{O}(\log N)$ [30], [31].

In this study, we evaluate a k-Nearest Neighbors (k-NN) [27] algorithm and a Local Outlier Factor (LOF) [32] algorithm as performance baselines due to their relative simplicity, high performance, and their ability to detect anomalies in local neighborhoods. Ensemble structures have been shown to increase performance and decrease parameter sensitivity [33], [34]. Thus, based on preliminary experiments, we implemented both methods using a parametric ensemble consisting of ten ensemble components randomly initialized with $k \in [5, 50]$. The remaining parameters for the methods were also tuned based on preliminary experiments. In the k-NN ensemble, we compute the average of all ensemble components' kth neighbor distances.

For the LOF ensemble, a data instance is considered an outlier when its score deviates significantly from 1 [35]. Hence, we calculate the outlier score as the average of the ensemble components' absolute difference from 1.

B. EXPLICIT GENERALIZATION MODELS

Explicit generalization models [27] are an alternative to instance-based methods that capture the normal behavior in a fixed size model, e.g., linear regression models, Boltzmann Machines, and Neural Networks. Generalization models often benefit from a low memory requirement and many possible extensions of the basic methods. Below, we provide details on our implementation and use of Principal Component Analysis, Autoencoder Structures, and Time Series Networks.

Principal Component Analysis: PCA can either be used as a standalone anomaly detection method or to extract data features for another method as part of a hybrid method [25]. In both cases, PCA generates a k-dimensional uncorrelated latent space (with $k < d$ principal components) using a linear combination of the original d data features. The k latent space features are then either fed to another detection method, or a reconstruction error is computed by re-projecting the features to the original data space:

$$s = \left\| \boldsymbol{x} - \sum_{j=1}^{k} \boldsymbol{e}_j^T \boldsymbol{x} \boldsymbol{e}_j \right\|, \tag{1}$$

where s is the reconstruction error, $\boldsymbol{e}_j$ is the jth eigenvector, and $\boldsymbol{x}$ is the d dimensional input feature. PCA can only describe linear relations between the features which motivates the usage of nonlinear explicit generalization methods, e.g., kernel-PCA and neural network-based AE structures. In our PCA implementation, we use the randomized Singular Value Decomposition (SVD) algorithm [36] for obtaining the principal components.

We evaluate three PCA variants: (i) the standard PCA as described above, (ii) the Probabilistic Principal Component Analysis (PPCA) [37], and (iii) the Incremental Principal Component Analysis (IPCA) [38]. PCA and IPCA differ only by how the model is fitted; IPCA is trained in small batches, whereas PCA is deterministic since all data instances are used at the same time. PPCA is trained like PCA, but it returns the probability of the query belonging to the learned model, instead of the Euclidean distance. We initialize each method as a parametric ensemble consisting of five ensemble components with $k \in [4, 12]$, we compute the outlier score by averaging the reconstruction error of each ensemble component. When training the IPCA, we use a batch size of 128. These parameters were found in preliminary experiments.

We also implement a hybrid model consisting of an IPCA model ($k = 6$) for feature extraction and a One-Class Support Vector Machine (OCSVM) as the detector. We initialize it as an ensemble consisting of 30 OCSVM components using the Radial Basis Functions (RBF) as the kernels. We use a six-dimensional latent space because it yielded the best results in preliminary experiments, and we observed no gain in performance with more than 30 ensemble components. Additionally, to mitigate the long training time resulting from OCSVM's computational complexity of $\mathcal{O}(N^2)$, we train it using $1,000$ randomly sampled data instances, as recommended in [27]. We combine the ensemble outputs by averaging the scores.

Autoencoder Structures: In an AE, the Neural Network (NN) can be thought of as two sub-networks: an encoder mapping the data features from the original space $\boldsymbol{x} \in \mathbb{R}^{d_x}$ to a latent space $\boldsymbol{z} \in \mathbb{R}^{d_z}, \boldsymbol{z} = e(\boldsymbol{x})$, and a decoder reconstructing the data features to the original space, $\hat{\boldsymbol{x}} = d(\boldsymbol{z})$; similar to PCA. To avoid the network simply passing the input to the output, the AE structure must be regularized e.g., by using

TABLE 2. The implemented UAE network structure.

Layer	Layer size	Activation fnc.
x	25	
$h_1 = Dense(x)$	22	tanh
$z \;= Dense(h_1)$	14	tanh
$h_2 = Dense(z)$	22	tanh
$\hat{x} \;= Dense(h_2)$	25	linear

TABLE 3. The implemented RandNet network structure.

Layer	Layer size	Activation fnc.
x	25	
$h_1 = Layer(x)$	18	tanh
$z \;= Layer(h_1)$	12	tanh
$h_2 = Layer(z)$	18	tanh
$\hat{x} \;= Layer(h_2)$	25	linear

an undercomplete network, or regularization on the loss function [22], e.g. using the L2 norm.

The performance of NN methods is highly sensitive to the choice of hyperparameters [39]. Therefore, an appropriate strategy for choosing hyperparameters must be considered when designing the autoencoders. We use Hyperband [40] from the keras-tuner module [41] to tune appropriate network parameters, e.g., layer size, activation function, and learning rate. We use Hyperband because of its focus on rapidly testing of configurations and fast elimination of non-promising configurations. We configure Hyperband to evaluate the models using an unsupervised loss function, i.e., reconstruction error. To prevent Hyperband from exploring very deep networks, we fix the structure of all autoencoders to consist of: (i) one encoding layer, (ii) a latent space layer, and (iii) one decoding layer. Hyperband tends to prematurely terminate models with a slow convergence rate, thus prioritizing models with a fast convergence rate even though they might converge to a suboptimal solution. We mitigated this issue by pinpointing multiple smaller search spaces using a random search tuner and then running Hyperband on the subspaces.

We use dense layers for all network layers and train all autoencoder variants using the Adam optimizer [42] for 30 epochs at a batch size of 128, unless stated otherwise. The parameters explicitly mentioned in the text are found using Hyperband, the remaining parameters are set to their default value given by scikit-learn (v. 0.23.2) [43] or TensorFlow (v. 2.4.0) [44]. In the following paragraphs, we introduce the networks compared in this paper.

1) UNDERCOMPLETE AUTOENCODER

The Undercomplete Autoencoder (UAE) is one of the simplest AE structures and it regulates the network using an hourglass structure $d_x > d_z$, where d_x is the dimension of the input and d_z is the dimension of the latent space. We train the UAE using Mean Squared Error (MSE) as the loss function and a learning rate of $\alpha = 0.001$. The network details are shown in Table 2.

2) DENOISING AUTOENCODER

In the Denoising Autoencoder (DAE) [22], the input data is intentionally corrupted during training, e.g., by adding isotropic Gaussian noise to all data instances, to make the network robust to noise:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \varepsilon \boldsymbol{I}). \tag{2}$$

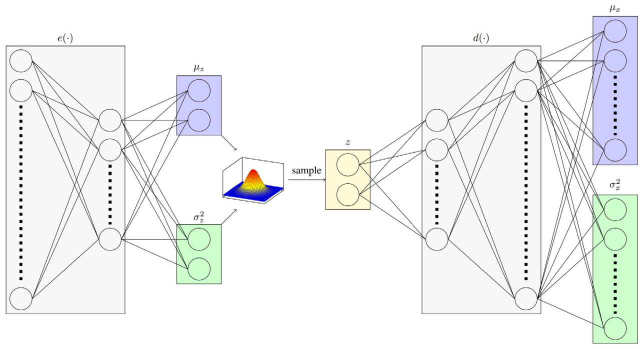


FIGURE 1. Visualization of the implemented VAE structure.

The loss is computed by calculating the MSE loss between the noisy data propagated through the network and the original noise-free data:

$$\mathcal{L}(\boldsymbol{x}, d(e(\tilde{\boldsymbol{x}}))). \tag{3}$$

In our implementation, we reuse the UAE network and add Gaussian noise, $\varepsilon = 0.1$, to all data features except the six TCP force-torque features. We do not add noise to the TCP force-torque features as they are subject to sensor noise and the artificial noise reduced the performance of the method.

3) AUTOENCODER ENSEMBLE

Chen et al. [34] introduced the Randomized Neural Network (RandNet) for Outlier Detection. RandNet is an autoencoder ensemble that has shown promising anomaly detection performance on some datasets from the UCI Machine Learning Repository.[1] The networks are randomized in the sense that some of the layer connections are randomly discarded to create diversity between the ensemble components.

We implement a RandNet consisting of five ensemble components instead of the proposed 100 ensemble components used by the authors because our network is shallower, and the additional components did thus not improve performance. We randomly remove connections as proposed by the authors. The network details are shown in Table 3.

4) VARIATIONAL AUTOENCODER

The Variational Autoencoder (VAE) is a generative model based on variational inference [19], and the latent space is thus a posterior distribution $q(z|\boldsymbol{x})$ instead of a feature

[1] https://archive.ics.uci.edu/

TABLE 4. The implemented VAE network structure.

Layer	Layer size	Activation fnc.
x	25	
$h_1 = Dense(x)$	26	ReLU
$\mu_z = Dense(h_1)$	24	linear
$\log(\sigma_z) = Dense(h_1)$	24	linear
$z \ = Sample(\mathcal{N}(\mu_z, \sigma_z^2 I))$	24	-
$h_2 = Dense(z)$	26	ReLU
$\mu_x = Dense(h_2)$	25	linear
$\sigma_x = Dense(h_2)$	25	softplus

vector. The encoder network is responsible for approximating the true posterior distribution of the latent features, $p(z|\boldsymbol{x})$. We assume that the latent features are distributed according to an isotropic Gaussian:

$$z \sim q(z|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z^2 \boldsymbol{I}). \quad (4)$$

We then use the reparameterization trick [19] to sample from the learned latent space posterior and feed it through the decoder network used to reconstruct the data instance into the original feature space:

$$\boldsymbol{z} = \boldsymbol{\mu}_z + \boldsymbol{\sigma}_z \cdot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I}). \quad (5)$$

When training the VAE, we minimize the reconstruction error and regularize the loss function using the Kullback-Leibler (KL) divergence between an isotropic Gaussian prior with unit variances, $p(z)$, and the approximated latent space posterior distribution, $q(z|\boldsymbol{x})$:

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{z\sim q}[\log p(\boldsymbol{x}|z)] - D_{KL}(q(z|\boldsymbol{x})||p(z)). \quad (6)$$

When using MSE as reconstruction error, it is equivalent to assuming that the original feature space posterior, $p(\boldsymbol{x}|z)$, has unit variance [45]. Due to our standardization method, presented later in eq. (13), we know this assumption does not hold. Thus, we expand the decoder network to learn an isotropic multivariate Gaussian as the output, as visualized in Fig. 1. With an appropriate choice of batch size (greater than 100), we can directly use the computed log-likelihood as the expected log-likelihood in the VAE's loss function [19]. For a detailed explanation of auto-encoding variational Bayes, we refer the reader to [19] and [46]. We train the network using a learning rate of $\alpha = 0.0005$, and Table 4 summarizes the VAE network.

5) CLUSTERING THE LATENT SPACE OF THE VARIATIONAL AUTOENCODER

In preliminary experiments with the VAE structure, we observed that instances sampled within the same program nodes (see Section IV) tend to cluster in the latent space. Fig. 2 illustrates this phenomenon with a 2-dimensional t-SNE plot [47]. We therefore construct a novel deep hybrid model using a Gaussian Mixture Model (GMM) [48] to cluster the latent space of the VAE.

In our implementation, we reuse the VAE network structure from Table 4 and we initialize the GMM with the same number of modes as the number of program nodes in the robot's program tree.

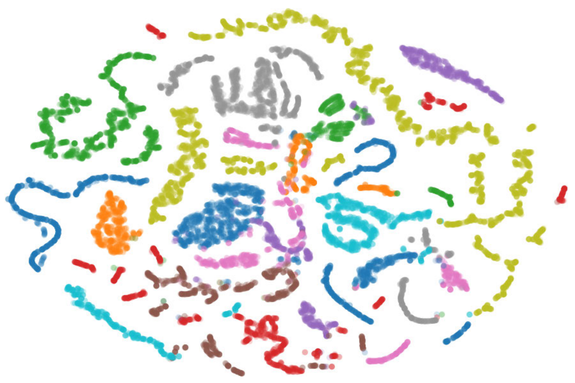


FIGURE 2. The 2D t-SNE representation of the latent space feature encoded by the VAE structure used on the behavioral features for data collected during the normal behavior of a robot program. We apply a color to each data instance based on the program node it is collected under; data instances collected under the same program nodes are assigned the same color.

TABLE 5. The implemented CVAE network structure.

Layer	Layer size	Activation fnc.
x	$25 + len(onehot)$	
$h_1 = Dense(x)$	26	ReLU
$\mu_z = Dense(h_1)$	8	linear
$\log(\sigma_z) = Dense(h_1)$	8	linear
$z \ = Sample(\mathcal{N}(\mu_z, \sigma_z^2 I))$	$8 + len(onehot)$	-
$h_2 = Dense(z)$	26	ReLU
$\mu_x = Dense(h_2)$	25	linear
$\sigma_x = Dense(h_2)$	25	softplus

6) CONDITIONAL VARIATIONAL AUTOENCODER

The Conditional Variational Autoencoder (CVAE) structure is an extension to the VAE structure in which a set of conditional (contextual) features are both concatenated with the input data to encoder and decoder network [45]. Providing contextual information to the network can potentially reduce the complexity of the feature mapping the method needs to learn.

We extend the use case of the network to a robotic setting and introduce the concept *program node ID*, which is further elaborated in Section IV. The *program node ID* is fed to the CVAE as a one-hot encoded categorical feature, since this transformation is commonly used when passing categorical data to neural networks [49]. The network is trained using a learning rate of $\alpha = 0.0005$ and the network details are listed in Table 5.

Time Series Networks: Using time series models, such as LSTM networks or CNN networks, for anomaly detection in time series data has yielded promising results in previous studies [17], [18], [50]. However, the structures can rapidly accumulate many trainable parameters and often have a high tuning complexity to achieve good performance (see [18] for an example).

TABLE 6. **The implemented DeepAnT [51] network structure.**

Layer	Layer size	Kernel size	Filters	Activation fnc.
x	$(10, 25)$			
$h = Conv1D(x)$	$(32, 23)$	3	32	ReLU
$h = MaxPool1D(h)$	$(32, 11)$	2		-
$h = Conv1D(h)$	$(32, 9)$	3	32	ReLU
$h = MaxPool1D(h)$	$(32, 4)$	2		-
$h = Flatten(h)$	128			-
$h = Dense(h)$	40			ReLU
$\hat{x} = Dense(h)$	25			linear

TABLE 7. **The implemented LSTM network structure.**

Layer	Layer size	Activation fnc.
x	$(10, 25)$	
$h_1 = LSTM(x)$	$(10, 24)$	tanh
$h_2 = LSTM(h_1)$	6	tanh
$z \ = RepeatVector(h_2)$	$(10, 6)$	-
$h_3 = LSTM(z)$	$(10, 6)$	tanh
$h_4 = LSTM(h_3)$	24	tanh
$\hat{x} \ = Dense(h_4)$	25	linear

We briefly explore some time series methods to investigate if the extra model complexity introduced by analyzing sequences of data instances leads to a considerable increase in performance. The methods are implemented as estimation model-based methods as defined in [26]; we use the data sequence of length, t_{len}, consisting of the past and present data points, $\{\boldsymbol{x}_{t-t_{len}}, \ldots, \boldsymbol{x}_t\}$, to calculate the expected value $\boldsymbol{x}_t$.

7) CONVOLUTIONAL NEURAL NETWORK

Munir et al. [51] designed a general CNN for anomaly detection called DeepAnT using 1D convolutional layers. We implement the network as it is presented in the paper except for the sequence length, which we set to ten during training. The network structure is shown in Table 6.

8) LONG SHORT-TERM MEMORY AUTOENCODER

We implement the LSTM with the network structure listed in Table 7, and we train the network using a learning rate of $\alpha = 0.001$. The network structure and sequence length were found using Hyperband.

9) BILINEAR AUTOENCODER

Tran et al. [52], [53] recently presented a neural network based on Bilinear Layers (BL) and found that their bilinear network could outperform a CNN and a LSTM, both in terms of computational complexity and precision, in stock price prediction. Furthermore, the concept of bilinear mapping has recently yielded promising results for time series anomaly detection in dynamic maps [54] among others. Therefore, we extend the method to the robotic setting.

The bilinear map is a function that combines elements of two vector spaces into a third vector space [52]. In our case, the function maps a multivariate time series, $X \in \mathcal{R}^{d_x \times T}$, into a latent space using $W_1 \in \mathcal{R}^{d_z \times d_x}$ and $W_2 \in \mathcal{R}^{T \times d_t}$:

$$z = F(X) = W_1 X W_2, \tag{7}$$

TABLE 8. **The implemented BL AE network structure.**

Layer	Layer size	Activation fnc.
x	$(5, 25)$	
$h = BL(x)$	$(2, 45)$	ReLU
$\hat{x} = BL(h)$	25	linear

where W_1 can be interpreted as mapping the d_x-dimensional representation of each time series to a d_z-dimensional subspace, while W_2 maps the contribution of the T times series into d_t times series using a weighted average approach.

We stack the Bilinear layer and train it as an autoencoder network. We use the MSE reconstruction error as the loss function and we regularize the network with a dropout rate of 0.1 and L_2 max-norm of 1.5 as recommended in [53]. We structure the network as shown in Table 8. We found that the billinear network generally is hard to use in a semi-supervised setting because the network tends to converge to local minima. Additionally, networks with low reconstruction error did not always achieve better performance than networks with a slightly higher reconstruction error, it is thus difficult to tune the network properly using Hyperband. Therefore, we resorted to manual tuning.

C. EVALUATION METRICS

The objective of the evaluation is to describe the method's discriminatory power, i.e., the ability to distinguish between normal data instances and anomalies. An often used metric within classification is accuracy:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{8}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives. However, as can be seen in eq. (8), accuracy depends on TN, and since anomaly detection tends to be an imbalanced problem where most data instances typically are normal, a high score is not necessarily indicative of high anomaly detection performance: simply classifying all data instances as normal would yield a high accuracy score.

Two often used metrics within anomaly detection for methods returning a continuous output score are the Receiver Operating Characteristic (ROC) curve [17], [27], [34], [45] and the Precision-Recall (PR) curve [13], [18], [27]. The ROC curve denotes the trade-off between the *true positive rate* (*TPR*), which is the percentage of anomalies correctly identified as anomalous, and the *false positive rate* (*FPR*), which is the percentage of normal instances wrongly classified anomalies, by plotting the *TPR* against the *FPR*.

$$TPR = \frac{TP}{TP + FN}, \tag{9}$$

$$FPR = \frac{FP}{FP + TN}. \tag{10}$$

The ROC curve has the benefit of its Area Under Curve (AUC) being interpretable as the detector's discriminatory power [55] since Receiver Operating Characteristic Area Under Curve (ROC-AUC) is interpretable as the probability of a detector assigning a higher score, e.g., greater distance or high reconstruction error, to an anomaly than a normal data instance [27]. However, ROC presents the performance as overly optimistic for imbalanced datasets, since a large change in FP can lead to a small change in *FPR*, when having a high number of normal instances and thereby possibly $TN \gg FP$ [56].

The *PR curve* denotes the trade-off between the *precision* and the *recall* of a detector, defined as follows:

$$\text{recall} = \frac{TP}{TP + FN}, \tag{11}$$

$$\text{precision} = \frac{TP}{FP + TP}. \tag{12}$$

The recall metric is the same as *TPR*, while precision is the number of identified anomalies actually being anomalous. Precision is not impacted by imbalanced data in the same sense as *FPR*, since it relates *TP* to *FP* instead of *TN* [56].

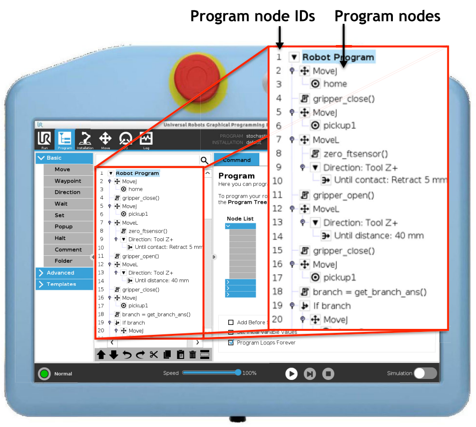
A ROC curve dominates another ROC curve, i.e., all parts of the other curve are beneath or equal, if and only if the corresponding PR curve for the method dominates as well [56]. In [56], it is furthermore shown that optimizing the ROC-AUC does not necessarily optimize the Precision Recall Area Under Curve (PR-AUC). Based on these findings, it is appropriate to choose methods optimizing the PR-AUC, while taking into account the trade-offs reflected by the ROC and PR curves.

IV. EXPERIMENTAL SETUP

We use a UR5e [57] manipulator from Universal Robots in this study. The UR5e has six joints with a wide working range of ±360° and a maximum joint speed of 180°/s for all joints. The robot's 850 mm reach and ±0.03 mm pose repeatability makes it suitable for many high-precision tasks. The *teach pendant* provides an drag-and-drop programming interface that creates a program tree consisting of multiple robot operations. The teach pendant and a snippet from a program tree with the program nodes and program node IDs are shown in Fig. 3.

To evaluate the presented methods, we design a pick-and-place application, because collaborative robots and robot manipulators in general are commonly used for such applications [10], [12], [18], [58]. The robot program consists of the main program and a rarely-visited branch. The robot uses contact detection, i.e. moving until the TCP measures a contact, to locate the objects and to determine when to release the object during place operations. Finally, the program also includes a non-deterministic place location to represent applications in which the exact location of the objects is not known in advance.

Figure 4 shows the robot in its home configuration and the workcell. The robot's task is to move the cylindrical objects



```
1  Robot Program
2    MoveJ
3      home
4    gripper_close()
5    MoveJ
6      pickup1
7    MoveL
8      zero_ftsensor()
9      Direction: Tool Z+
10       Until contact: Retract 5 mm
11   gripper_open()
12   MoveL
13     Direction: Tool Z+
14       Until distance: 40 mm
15   gripper_close()
16   MoveJ
17     pickup1
18   branch = get_branch_ans()
19   If branch
20     MoveJ
```

FIGURE 3. A screenshot of the *teach pendant* with a visualization of the program nodes and program node IDs.

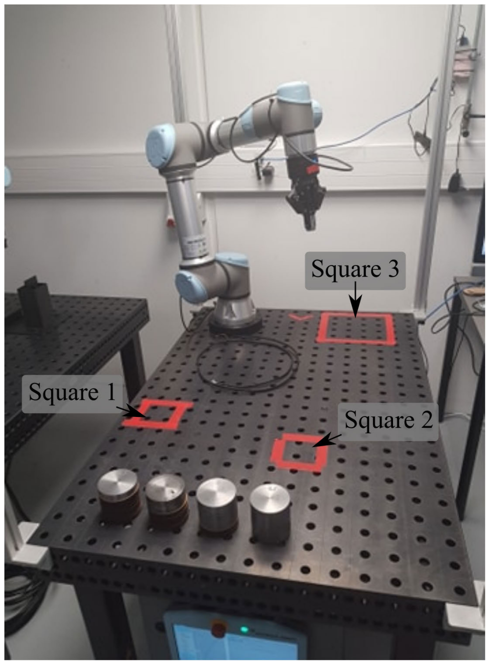


FIGURE 4. The robot workcell.

between the red squares marked on the table with the use of a Robotiq 2F-85 gripper. The four cylinders, from left to right, have a mass of 0.5 kg, 1.0 kg, 2.0 kg, and 2.5 kg, respectively. The cylinder of mass 2.0 kg is used as the default mass in our experiments.

The robot first picks up the cylinder at Square 1, and then it either moves to Square 3 or enters the rarely-visited program branch with a likelihood of 0.167, and moves to Square 2 before moving to Square 3. At Square 2, the cylinder is placed at the center of the marked square. The robot then waits for a random amount of time uniformly drawn from the interval four to ten seconds, before it picks up the cylinder. Afterwards, the robot moves to Square 3. The place location at Square 3 is randomly generated to be within the 20 cm × 20 cm red square sampled from a uniform distribution. The robot then moves the cylinder back to Square 1. A video of the robotic experiment can be found at the following link: https://youtu.be/a63tmL3yC_E.

To collect data from the robot, we connected a computer to the robot's Real-Time Data Exchange (RTDE) interface [59]. The RTDE interface can sample sensor data and information about the robot's state at a maximum frequency of 500 Hz. The sampled features are listed in Table 1 and they consist of 25 behavioral features and two contextual features; one being a custom designed feature, *Time_in_node*, derived from the *timestamp* and the *Program node ID* feature. We standardize the features by computing a shared mean and variance for each category, e.g., all current measurements share a common mean and variance:

$$x'_j = \frac{x_j - \hat{\mu}_{c(j)}}{\hat{\sigma}_{c(j)}}, \tag{13}$$

where $\hat{\mu}_{c(j)}$ is the shared mean for all features belonging to category $c(j)$ and $\hat{\sigma}_{c(j)}$ is the shared standard deviation for all features belonging to category $c(j)$.

Aside from collecting data during normal operation, we conduct experiments in which we introduce eight different anomalies:

1) *Push*: we push the robot at either joint 3 or joint 5 (see Fig. 5a) with a force less than 50 N.
2) *TCP wrench*: we apply a wrench to the robot's TCP (Fig. 5c).
3) *Raised object*: we introduce a foreign object at the place location which makes the robot place the cylinder at a location higher than in the normal behavior (Fig. 5b).
4) *Outside square*: we sample a location outside the usual 20 cm × 20 cm square (Fig. 5d).
5) *Change weight*: we change the default 2.0 kg cylinder with either the 0.5 kg, 1.0 kg, or 2.5 kg cylinder.
6) *Drop object*: the robot losses the cylinder (Fig. 6).
7) *Speed scale*: we reduce the robot's movement speed by 50%.
8) *Long wait*: we extend the four to ten seconds wait to either 12 s, 15 s, or 20 s.

We designed the anomalies to represent a board range of common exogenous faults in robot application, e.g., (i) the robot collides with a foreign object in its environment (*Push*, *TCP wrench*, *Drop object*), (ii) the machine feeding the robot fails to deliver an object or delivers a wrong object (*Change weight*, *Long wait*, and the result can be *Drop object*), (iii) the machine receiving objects from the robot fails to move it before a new object is placed (*Raised object*), (iv) a safety sensor is wrongly triggered (*Speed scale*), and (v) an external sensor provides an incorrect pose estimate (*Outside square*).

The collected data comprise a dataset representing normal behavior collected from 600 program runs, and eight datasets with anomalies present – one for each of the introduced anomalies collected over ten programs runs per anomaly, i.e., a total of 80 program runs with anomalies present.

TABLE 9. Summary of data sets collected during normal operation and with external anomalies.

Dataset name	Total samples	Anomalous samples	Contamination [%]
Normal	1530921	0	0.00
Push	31117	2687	8.64
TCP wrench	31062	4660	15.00
Raised object	28438	3992	14.04
Outside box	23499	7539	32.08
Change weight	28978	9656	33.32
Drop object	22274	5105	22.92
Speed scale	32804	7449	22.71
Long wait	47979	5600	11.67

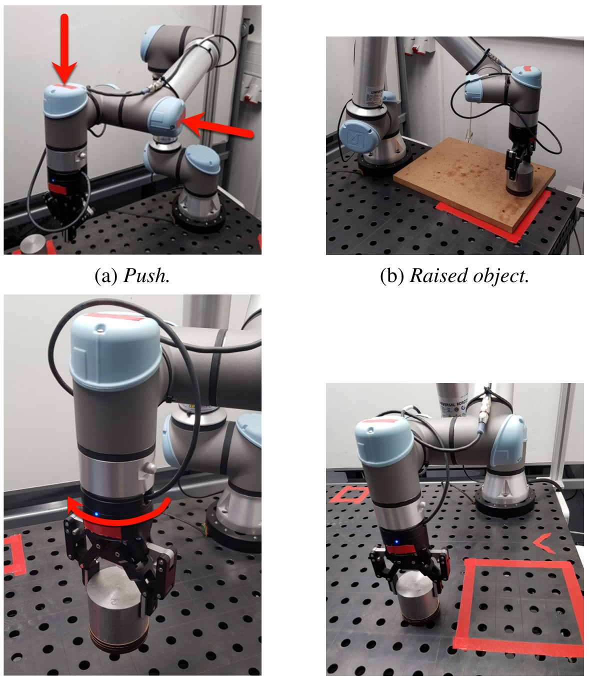

(a) *Push.* (b) *Raised object.* (c) *TCP wrench.* (d) *Outside box.*

FIGURE 5. Illustration of the introduced anomalies.

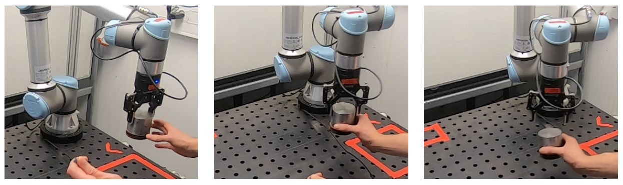


FIGURE 6. Illustration of the *Drop object* anomaly.

The datasets are available at [7]. The data is sampled at a frequency of 100 Hz, instead of the maximum 500 Hz, as it is sufficient to detect the classes of anomalies that we focus on in this study. Table 9 summarizes the datasets.

The implemented methods are all evaluated on a MacBook Pro with an Intel Core i7-9750H CPU. Even though many methods can benefit from running on a GPU they are all evaluated on a CPU. All software packages used for implementing the methods and their versions are listed in Table 10.

TABLE 10. **The software packages used to implement the experiments.**

Package	Version
Numpy [60]	1.19.5
scikit-learn [43]	0.23.2
TensorFlow [44]	2.4.0
TensorFlow Probability [44]	0.12.1

TABLE 11. **Training and prediction times for the evaluated methods. The times are reported as mean (± standard deviation) obtained by evaluating all methods 30 times on each test dataset (fitting to training data representing the robot application's normal behavior). Both measurements are obtained as process times, only counting time spent on the processing unit [63]. The $\mathcal{O}(\cdot)$ complexities of the training step and the prediction steps are shown for the non-NN-based methods. The computational complexities of the NN-based methods depend on their respective topologies, see Section III. *N*, *d* and *k* denote the number of data samples, the number of data features, and the number of latent dimensions, respectively.**

	Training time		Query time	
	$\mathcal{O}(\cdot)$	Time [s]	$\mathcal{O}(\cdot)$	Time [ms]
kNN [30], [31]	$Nk \log N$	116 (±6.1)	$\log N$	9.1 (±11)
LOF [32], [64]	N^2	4180 (±362)	$\log N$	15.8 (±14)
PCA [36]	$Nd \log k$	172 (±9.2)	dk	0.4 (±0.1)
PPCA [37]	Ndk	169 (±9.3)	dk	5.3 (±1.1)
IPCA [38]	$n^2 d$ *	184 (±4.7)	dk	0.4 (±0.1)
IPCA+ OCSVM [33]	N^2 **	38 (±1.6)	dk	2.8 (±0.4)
UAE	–	527 (±9.8)	–	1.2 (±0.2)
DAE	–	537 (±11.0)	–	1.2 (±0.2)
VAE	–	1078 (±34.4)	–	34.4 (±24)
VAE +GMM	–	5294 (±998)	–	1.6 (±0.4)
CVAE	–	990 (±33.0)	–	37.8 (±26)
DeepAnT	–	1534 (±62.1)	–	30.2 (±5.3)
LSTM AE	–	12690 (±378)	–	5021 (±1132)
BL AE	–	1155 (±173)	–	0.026 (±0.014)
RandNet	–	752 (±17.0)	–	31.0 (±2.8)

* update computational complexity for a batch size of n.
** only 1000 samples are used as stated in Section III.

V. EXPERIMENTAL RESULTS

Table 11 shows the training and query time for selected methods and Table 12 summarizes the discriminatory power of the 15 methods evaluated on the eight anomaly datasets individually. The three best performing methods are highlighted in bold for each anomaly. We compare the performance using the non-parametric Mann-Whitney U Test [61] by determining the degree of statistical significance of differences. We use Mann-Whitney since D'Agostino's test for normality [62] showed that some results are not normally distributed, i.e. $p < 0.05$.

From Table 12, we observe that the instance-based methods (k-NN and LOF) generally achieve a good performance despite their relative simplicity, while PCA-based reconstruction models (PCA, PPCA, and IPCA) generally have low performance; and implementing PCA as a hybrid method did not increase the performance on our datasets. Additionally, we see that non-linear models outperform PCA models in nearly all aspect. From the table, we also observe that hybrid methods (IPCA+OCSVM and VAE+GMM) yield a slightly higher performance than computing the score based on the MSE.

By comparing the UAE and DAE, we see that the incorporating the denoising criterion into the training phase increases the performance of the UAE; with the largest increase observed for the *Push* anomaly. Additionally, by comparing UAE with the statistical AE methods (VAE, CVAE, and VAE+GMM), we see that using variational inference increases the performance for the challenging anomalies: *Raised object*, *Outside box*, and *TCP wrench*. Specifically, we see that CVAE, which utilizes the robot program context, achieves the highest performance on these anomalies.

By comparing the time series methods, we see that the LSTM structure receives the highest score, and it ranks top three on half of the evaluated anomalies. Both DeepAnT and the bilinear network perform below average.

In order to gain insight into which aspects of robot applications complicate anomaly detection, we identified three hypotheses described below. To test the three hypotheses, we slightly modify the original pick-and-place task and collect a new training and evaluation dataset for each modification. In the new datasets, we only introduce the *Push* and *TCP wrench* anomaly and we only evaluated three anomaly detectors: LOF, VAE+GMM, and CVAE.

A. RARELY-VISITED PROGRAM BRANCHES

Hypothesis 1: Distinguishing between normal and anomalous data instances in the rarely-visited program branch is significantly harder than in the rest of the program.

To test Hypothesis 1, we remove the program branch that makes the robot move to Square 2, see Section IV, and we sample a new dataset, which we denote *no_branch*. The collected data is summarized in Table 13 and Table 14 lists the evaluation results for the methods.

We formulate the null hypothesis that the mean performance of the methods trained on the original data, μ_{orig}, is equal to the mean performance of the methods trained on the *no_branch* data, μ_{no_branch}:

$$H_0^{(1)} : \mu_{orig}^{\text{LOF}} = \mu_{no\_branch}^{\text{LOF}}, \tag{14}$$

$$H_0^{(2)} : \mu_{orig}^{\text{VAE+GMM}} = \mu_{no\_branch}^{\text{VAE+GMM}}, \tag{15}$$

$$H_0^{(3)} : \mu_{orig}^{\text{CVAE}} = \mu_{no\_branch}^{\text{CVAE}}. \tag{16}$$

We use the Mann-Whitney U Test to test for equality of means, which we reject for all three methods, i.e., $p < 0.05$. We thus reject the null hypothesis.

We then formulate an alternative hypothesis that μ_{no_branch} is greater than μ_{orig} and test it using the one-sided Mann-Whitney U Test:

$$H_A^{(1)} : \mu_{orig}^{\text{LOF}} < \mu_{no\_branch}^{\text{LOF}}, \tag{17}$$

$$H_A^{(2)} : \mu_{orig}^{\text{VAE+GMM}} < \mu_{no\_branch}^{\text{VAE+GMM}}, \tag{18}$$

TABLE 12. The results are reported as mean (± standard deviation) obtained by evaluating all methods 30 times on the test dataset (fitting to training data representing the robot application's normal behavior and evaluating on the datasets with anomalies 30 times). All standard deviations reported as (± 0.0000) indicate that the standard deviation is below 10^{-4}. The three best performing methods are highlighted for each dataset.

		All anomalies	Push	Raised object	Outside box	TCP wrench	Long wait	Drop object	Change weight	Speed scale
KNN	PR-AUC	**0.7926** (±0.0033)	**0.6248** (±0.0049)	0.3054 (±0.0062)	0.6972 (±0.0011)	0.7821 (±0.0046)	0.9737 (±0.0003)	**0.9526** (±0.0026)	0.6894 (±0.0012)	**0.8421** (±0.0041)
	ROC-AUC	0.9428 (±0.0014)	0.9429 (±0.0010)	0.8029 (±0.0061)	0.9213 (±0.0005)	0.8930 (±0.0020)	0.9948 (±0.0000)	0.9860 (±0.0006)	0.8805 (±0.0007)	0.9586 (±0.0011)
LOF	PR-AUC	**0.8255** (±0.0012)	0.5049 (±0.0004)	**0.6698** (±0.0089)	0.7559 (±0.0016)	0.8141 (±0.0004)	0.9955 (±0.0004)	**0.9748** (±0.0003)	0.7215 (±0.0015)	**0.7981** (±0.0006)
	ROC-AUC	0.9531 (±0.0002)	0.9322 (±0.0001)	0.9418 (±0.0010)	0.9300 (±0.0004)	0.9518 (±0.0001)	0.9991 (±0.0000)	0.9891 (±0.0001)	0.8788 (±0.0002)	0.9413 (±0.0004)
PCA	PR-AUC	0.3524 (±0.0264)	0.3465 (±0.9374)	0.1107 (±0.0049)	0.4892 (±0.0510)	0.4235 (±0.0305)	0.6854 (±0.2418)	0.3403 (±0.0266)	0.4129 (±0.0102)	0.2424 (±0.0089)
	ROC-AUC	0.6470 (±0.0146)	0.7952 (±0.0215)	0.3329 (±0.0140)	0.6971 (±0.0674)	0.7397 (±0.0251)	0.9354 (±0.0584)	0.6595 (±0.0322)	0.6040 (±0.0238)	0.5565 (±0.0186)
PPCA	PR-AUC	0.3882 (±0.0112)	0.3319 (±0.0282)	0.1144 (±0.0035)	0.4838 (±0.0591)	0.4203 (±0.0255)	0.9806 (±0.0113)	0.3134 (±0.0203)	0.4320 (±0.0077)	0.2275 (±0.0061)
	ROC-AUC	0.6489 (±0.0172)	0.8004 (±0.0154)	0.3422 (±0.0087)	0.6724 (±0.0822)	0.7845 (±0.0123)	0.9980 (±0.0012)	0.6259 (±0.0261)	0.6254 (±0.0089)	0.5200 (±0.0165)
IPCA	PR-AUC	0.3448 (±0.0382)	0.2920 (±0.0371)	0.1144 (±0.0103)	0.5046 (±0.0504)	0.3503 (±0.0241)	0.7243 (±0.2831)	0.3361 (±0.0406)	0.3906 (±0.0189)	0.2500 (±0.0121)
	ROC-AUC	0.6395 (±0.0209)	0.7630 (±0.0278)	0.3698 (±0.0226)	0.7361 (±0.0690)	0.7712 (±0.0351)	0.9445 (±0.0599)	0.6425 (±0.0463)	0.5647 (±0.0270)	0.5468 (±0.0210)
IPCA+ OCSVM	PR-AUC	0.4803 (±0.0023)	0.2528 (±0.0024)	0.1307 (±0.0011)	0.2279 (±0.0012)	0.4300 (±0.0051)	**0.9984** (±0.0004)	0.6593 (±0.0051)	0.6415 (±0.0031)	0.3384 (±0.0040)
	ROC-AUC	0.6153 (±0.0017)	0.6060 (±0.0039)	0.3966 (±0.0043)	0.2188 (±0.0034)	0.7240 (±0.0033)	0.9998 (±0.0000)	0.7463 (±0.0045)	0.7764 (±0.0029)	0.4966 (±0.0051)
UAE	PR-AUC	0.6489 (±0.0226)	0.3607 (±0.0278)	0.1443 (±0.0248)	0.7640 (±0.0607)	0.7505 (±0.0102)	0.9901 (±0.0105)	0.7176 (±0.0352)	0.6613 (±0.0635)	0.4929 (±0.0370)
	ROC-AUC	0.8083 (±0.0117)	0.7996 (±0.0161)	0.5011 (±0.0730)	0.9367 (±0.0154)	0.8301 (±0.0130)	0.9985 (±0.0016)	0.8328 (±0.0217)	0.7735 (±0.0274)	0.7179 (±0.0183)
DAE	PR-AUC	0.7014 (±0.0101)	**0.5246** (±0.0362)	0.1745 (±0.0115)	0.7169 (±0.0121)	0.7718 (±0.0225)	**0.9995** (±0.0004)	0.8169 (±0.0160)	**0.7646** (±0.0193)	0.5275 (±0.0237)
	ROC-AUC	0.8500 (±0.0067)	0.9022 (±0.0092)	0.5583 (±0.0310)	0.9286 (±0.0024)	0.8373 (±0.0200)	0.9999 (±0.0000)	0.9055 (±0.0140)	0.8386 (±0.0139)	0.7695 (±0.0156)
VAE	PR-AUC	0.7018 (±0.0306)	0.3732 (±0.0456)	0.3333 (±0.0981)	**0.8910** (±0.0571)	**0.8787** (±0.0384)	0.7143 (±0.1755)	0.7922 (±0.0198)	**0.7697** (±0.0375)	0.5586 (±0.0392)
	ROC-AUC	0.8572 (±0.0170)	0.8293 (±0.0125)	0.6625 (±0.0521)	0.9648 (±0.0173)	0.9465 (±0.0259)	0.8454 (±0.1182)	0.8855 (±0.0098)	0.8658 (±0.0158)	0.8147 (±0.0161)
VAE+ GMM	PR-AUC	**0.7925** (±0.0212)	0.4160 (±0.0619)	**0.6860** (±0.0761)	**0.7704** (±0.0522)	**0.8879** (±0.0469)	0.9626 (±0.0768)	0.8746 (±0.0236)	0.6836 (±0.0436)	0.7542 (±0.0305)
	ROC-AUC	0.9275 (±0.0078)	0.8461 (±0.0258)	0.9277 (±0.0200)	0.9388 (±0.0103)	0.9526 (±0.0368)	0.9841 (±0.0363)	0.9479 (±0.0126)	0.8422 (±0.0194)	0.9009 (±0.0124)
CVAE	PR-AUC	0.7208 (±0.0364)	0.4022 (±0.0316)	**0.8454** (±0.0423)	**0.8505** (±0.0379)	**0.9437** (±0.0168)	0.6318 (±0.2390)	0.8686 (±0.0275)	0.6099 (±0.0253)	0.5725 (±0.0586)
	ROC-AUC	0.8778 (±0.0259)	0.8405 (±0.0103)	0.9461 (±0.0166)	0.9499 (±0.0092)	0.9742 (±0.0083)	0.7409 (±0.1830)	0.9494 (±0.0121)	0.8353 (±0.0149)	0.8216 (±0.0246)
DeepAnT	PR-AUC	0.4169 (±0.0030)	0.2157 (±0.0076)	0.1695 (±0.0041)	0.5593 (±0.0183)	0.4248 (±0.0072)	0.9950 (±0.0001)	0.4113 (±0.0094)	0.4143 (±0.0062)	0.3331 (±0.0086)
	ROC-AUC	0.7200 (±0.0059)	0.7870 (±0.0065)	0.4761 (±0.0127)	0.7978 (±0.0236)	0.8647 (±0.0052)	0.9998 (±0.0000)	0.7674 (±0.0085)	0.6625 (±0.0079)	0.6820 (±0.0111)
LSTM AE	PR-AUC	0.7860 (±0.0108)	**0.6689** (±0.0238)	0.3328 (±0.0359)	0.7186 (±0.0225)	0.7409 (±0.0205)	0.9785 (±0.0073)	**0.8992** (±0.0213)	**0.7503** (±0.0342)	**0.7981** (±0.0158)
	ROC-AUC	0.9192 (±0.0049)	0.9457 (±0.0073)	0.8049 (±0.0294)	0.9265 (±0.0037)	0.8603 (±0.0189)	0.9970 (±0.0016)	0.9524 (±0.0111)	0.8697 (±0.0105)	0.9163 (±0.0066)
BL AE	PR-AUC	0.5827 (±0.0451)	0.4674 (±0.1135)	0.1564 (±0.0256)	0.7391 (±0.0430)	0.7218 (±0.0258)	0.9538 (±0.0301)	0.6273 (±0.2179)	0.5701 (±0.0806)	0.3587 (±0.0925)
	ROC-AUC	0.7861 (±0.0489)	0.8690 (±0.0480)	0.4934 (±0.1026)	0.9337 (±0.0086)	0.8320 (±0.0241)	0.9916 (±0.0040)	0.8059 (±0.1031)	0.7162 (±0.0469)	0.6705 (±0.0952)
RandNet	PR-AUC	0.3958 (±0.0101)	0.3246 (±0.0597)	0.1245 (±0.0075)	0.6891 (±0.0198)	0.4720 (±0.0632)	**0.9968** (±0.0033)	0.3824 (±0.0614)	0.4192 (±0.0510)	0.2541 (±0.0350)
	ROC-AUC	0.7681 (±0.0024)	0.7602 (±0.0411)	0.4200 (±0.0279)	0.9057 (±0.0139)	0.7787 (±0.0262)	0.9996 (±0.0004)	0.6866 (±0.0504)	0.6164 (±0.0466)	0.5867 (±0.0480)

$$H_A^{(3)} : \mu_{orig}^{\text{CVAE}} < \mu_{no\_branch}^{\text{CVAE}}. \tag{19}$$

For all methods, we also reject the alternative hypothesis, i.e., $p < 0.05$. Therefore, we conclude that the selected methods experience a significantly higher anomaly detection performance when scoring data instances in programs without rarely-visited program branches, which confirms Hypothesis 1.

TABLE 13. Number of data instances sampled for the normal case and all anomalous program runs without the program branch. The normal data is gathered from 200 program runs. The *Push* anomaly is introduced in ten program runs and the *TCP wrench* anomaly is introduced in ten program runs.

Dataset name	Total samples	Anomalous samples	Contamination [%]
Normal	422510	0	0.00
Push	21233	1391	6.55
TCP wrench	21171	2803	13.23

TABLE 14. Original results (with a rarely-visited program branch) from Table 12 and results of the new experiments without the rarely-visited program branch. Results are reported as mean (± standard deviation) which are obtained from 30 evaluations of each model.

Original results

		LOF	VAE+GMM	CVAE
Push	PR-AUC	0.5049 (±0.0004)	0.4160 (±0.0619)	0.4022 (±0.0316)
	ROC-AUC	0.9322 (±0.0001)	0.8461 (±0.0258)	0.8405 (±0.0103)
TCP wrench	PR-AUC	0.8141 (±0.0004)	0.8879 (±0.0469)	0.9437 (±0.0168)
	ROC-AUC	0.9518 (±0.0001)	0.9526 (±0.0368)	0.9742 (±0.0083)

Without rarely-visited program branch

		LOF	VAE+GMM	CVAE
Push	PR-AUC	0.9366 (±0.0007)	0.7350 (±0.0857)	0.8067 (±0.0228)
	ROC-AUC	0.9880 (±0.0003)	0.9558 (±0.0139)	0.9650 (±0.0030)
TCP wrench	PR-AUC	0.9220 (±0.0016)	0.9641 (±0.0105)	0.9668 (±0.0030)
	ROC-AUC	0.9910 (±0.0001)	0.9899 (±0.0018)	0.9905 (±0.0010)

B. NON-DETERMINISTIC TRAJECTORIES

Hypothesis 2: Distinguishing between normal and anomalous data instances in deterministic trajectories is easier than in non-deterministic trajectories.

To test Hypothesis 2, we remove the program branch that makes the robot move to Square 2, we replace the semi-randomly generated place location a Square 3 with a fixed location, and we sample a new dataset, which we denote *deterministic*. The collected data is summarized in Table 15 and Table 16 lists the evaluation results of the anomaly detection methods.

We formulate the null hypothesis that the mean performance of the methods trained on the *no_branch* data, μ_{no_branch}, data is equal to the performance of the methods trained on the *deterministic* data, $\mu_{deterministic}$:

$$H_0^{(1)} : \mu_{no\_branch}^{\text{LOF}} = \mu_{deterministic}^{\text{LOF}}, \tag{20}$$
$$H_0^{(2)} : \mu_{no\_branch}^{\text{VAE+GMM}} = \mu_{deterministic}^{\text{VAE+GMM}}, \tag{21}$$
$$H_0^{(3)} : \mu_{no\_branch}^{\text{CVAE}} = \mu_{deterministic}^{\text{CVAE}}. \tag{22}$$

TABLE 15. Number of data instances sampled for the normal case and all anomalous program runs without the program branch and with a fixed place location. The normal data is collected in 200 program runs. The *Push* anomaly is introduced in ten program runs and the *TCP wrench* anomaly is introduced in ten program runs.

Dataset name	Total samples	Anomalous samples	Contamination [%]
Normal	420886	0	0.00
Push	21192	1401	6.61
TCP wrench	21152	2805	13.26

TABLE 16. Fixed-place-location results. Results are reported as mean (± standard deviation) which are obtained from 30 evaluations of each model.

		LOF	VAE+GMM	CVAE
Push	PR-AUC	0.9814 (±0.0014)	0.8460 (±0.0594)	0.7718 (±0.0363)
	ROC-AUC	0.9962 (±0.0004)	0.9716 (±0.0080)	0.9598 (±0.0096)
TCP wrench	PR-AUC	0.9861 (±0.0010)	0.9847 (±0.0045)	0.9964 (±0.0009)
	ROC-AUC	0.9978 (±0.0002)	0.9963 (±0.0010)	0.9989 (±0.0004)

We use the Mann-Whitney U Test to test for equality of means, which we reject for all three methods, i.e., $p < 0.05$.

We then formulate an alternative hypothesis that $\mu_{deterministic}$ is greater than μ_{no_branch} and test it using the one-sided Mann-Whitney U Test:

$$H_A^{(1)} : \mu_{no\_branch}^{\text{LOF}} < \mu_{deterministic}^{\text{LOF}}, \tag{23}$$
$$H_A^{(2)} : \mu_{no\_branch}^{\text{VAE+GMM}} < \mu_{deterministic}^{\text{VAE+GMM}}, \tag{24}$$
$$H_A^{(3)} : \mu_{no\_branch}^{\text{CVAE}} < \mu_{deterministic}^{\text{CVAE}}. \tag{25}$$

We accept the alternative hypothesis for all methods and anomalies, i.e., $p < 0.05$, except for CVAE and the *Push* anomaly, i.e., $p > 0.05$. We conclude that two out of three methods significantly improve their anomaly detection performance when the robot program only consists of deterministic trajectories, which confirms that Hypothesis 2 holds for two out of three methods evaluated.

C. INTENTIONAL CONTACT WITH THE ENVIRONMENT

Hypothesis 3: Distinguishing between normal and anomalous data instances in robot programs with intentional contact with the environment is significantly harder than those without.

To test Hypothesis 3, we remove the program branch that makes the robot move to Square 2, we replace the contact-based pick-and-place operations with hard-coded locations, and we sample a new dataset, which we denote *no_contact*. The collected data is summarized in Table 17 and Table 18 lists the evaluation results for the methods.

We formulate the null hypothesis that the mean performance of the methods trained on the *no_branch* data, μ_{no_branch} is equal to the performance of the methods trained

TABLE 17. Number of data instances sampled for the normal case and all anomalous program runs without the program branch and without intentional contact with the environment. The normal data is gathered from 200 program runs. The *Push* anomaly is introduced in ten program runs and the *TCP wrench* anomaly is introduced in ten program runs.

Dataset name	Total samples	Anomalous samples	Contamination [%]
Normal	307782	0	0.00
Push	15391	1488	9.67
TCP wrench	15383	1647	10.71

TABLE 18. Without-intentional-contact results. The results are reported as mean (± standard deviation) which are obtained from 30 evaluations of the models.

		LOF	VAE+GMM	CVAE
Push	PR-AUC	0.9575 (±0.0024)	0.8951 (±0.0280)	0.8825 (±0.0183)
	ROC-AUC	0.9895 (±0.0005)	0.9767 (±0.0052)	0.9715 (±0.0038)
TCP wrench	PR-AUC	0.9401 (±0.0026)	0.9732 (±0.0116)	0.9839 (±0.0030)
	ROC-AUC	0.9932 (±0.0002)	0.9936 (±0.0018)	0.9959 (±0.0010)

on the *no_contact* data, $\mu_{no\_contact}$:

$$H_0^{(1)} : \mu_{no\_branch}^{\text{LOF}} = \mu_{no\_contact}^{\text{LOF}}, \quad (26)$$
$$H_0^{(2)} : \mu_{no\_branch}^{\text{VAE+GMM}} = \mu_{no\_contact}^{\text{VAE+GMM}}, \quad (27)$$
$$H_0^{(3)} : \mu_{no\_branch}^{\text{CVAE}} = \mu_{no\_contact}^{\text{CVAE}}. \quad (28)$$

We use the Mann-Whitney U Test to test for equality of means, which we reject for all three methods, i.e., $p < 0.05$.

We then formulate an alternative hypothesis that $\mu_{no\_contact}$ is greater than μ_{no_branch} and test it using the one-sided Mann-Whitney U Test:

$$H_A^{(1)} : \mu_{no\_branch}^{\text{LOF}} < \mu_{no\_contact}^{\text{LOF}}, \quad (29)$$
$$H_A^{(2)} : \mu_{no\_branch}^{\text{VAE+GMM}} < \mu_{no\_contact}^{\text{VAE+GMM}}, \quad (30)$$
$$H_A^{(3)} : \mu_{no\_branch}^{\text{CVAE}} < \mu_{no\_contact}^{\text{CVAE}}. \quad (31)$$

We accept the alternative hypothesis for all three methods, i.e., $p < 0.05$. Therefore, we conclude that all methods yield a significantly higher anomaly detection performance when scoring data instances in programs without intentional contact with the environment, which confirms Hypothesis 3.

VI. DISCUSSION

In our experiments, we found that the instance-based methods k-NN and LOF achieve a remarkably high detection performance for the exogenous anomalies introduced in the pick-and-place application, particularly considering their relative simplicity. For the reconstruction-based method, we found that PCA achieves poor performance, which indicates that our data features have nonlinear dependencies. Neural network-based autoencoders, the nonlinear equivalent to PCA, on the other hand, achieve high detection performance for some of the evaluated network structures: (i) DAE, (ii) VAE+GMM, (iii) CVAE, and (iv) LSTM. In particular, structures that use variational inference achieve good performance. This observation aligns with current research [17], [18] that indicates that combining variational inference with, e.g., times series methods, yields good performance. The nonlinear explicit generalization methods have several interesting characteristics that could be exploited in robot applications: (i) learning can be done incrementally, (ii) samples can be weighted differently when updating the model, and (iii) fast prediction times when models are kept relatively small. The studied generalization models can also easily be modified and combined with other methods to further enhance their performance.

We found three cobot program elements that can significantly decrease the performance of anomaly detection methods: (i) when the robot program contains rarely-visited program branches, (ii) when the program consists of non-deterministic trajectories, and (iii) when the robot's TCP intentionally makes contact with the environment. It would be interesting to explore which detectors are the least vulnerable to these situations, e.g., it might be possible to mitigate the issue of rarely-visited program branches by considering the imbalanced representation of the program nodes when training the AE structures.

The results of our study indicate that exploiting the robot program context directly in the CVAE structure generally leads to significantly better performance than for the (non-conditional) VAE structure. However, indirect exploitation as in the VAE+GMM model also yielded very promising performance. It would be interesting to further explore the incorporation contextual information from the robot program when performing anomaly detection in robot manipulators.

VII. CONCLUSION

We evaluated 15 anomaly detection methods in a common pick-and-place application for a robot manipulator. We trained models on data collected under normal operation and assessed their discriminatory power on eight different anomalies. The two instance-based methods, namely k-NN and LOF, achieved good performance despite their relative simplicity compared to nonlinear explicit generalization models based on autoencoders. However, the performance profile of the evaluated methods varied significantly across the different types of anomalies. To achieve the best performance, the choice of method should thus depend on the expected likelihood and severity of different types of anomalies in a given environment. Alternatively, two (e.g. k-NN and VAE+GMM) or more methods could potentially be combined in an ensemble to achieve better overall performance, but at the cost of increased complexity.

REFERENCES

[1] Y. Cohen, S. Shoval, and M. Faccio, "Strategic view on cobot deployment in assembly 4.0 systems," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 1519–1524, 2019.

[2] S. El Zaatari, M. Marei, W. Li, and Z. Usman, "Cobot programming for collaborative industrial tasks: An overview," *Robot. Auto. Syst.*, vol. 116, pp. 162–180, Jun. 2019.

[3] P. Zhao, M. Kurihara, J. Tanaka, T. Noda, S. Chikuma, and T. Suzuki, "Advanced correlation-based anomaly detection method for predictive maintenance," in *Proc. IEEE Int. Conf. Prognostics Health Manage. (ICPHM)*, Jun. 2017, pp. 78–83.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, 2009.

[5] C. Fantuzzi, C. Secchi, and A. Visioli, "On the fault detection and isolation of industrial robot manipulators," *IFAC Proc. Volumes*, vol. 36, no. 17, pp. 399–404, Sep. 2003.

[6] A. De Luca and R. Mattone, "An identification scheme for robot actuator faults," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2005, pp. 1127–1131.

[7] S. Graabæk, E. V. Ancker, A. R. Fugl, and A. L. Christensen. (2022). *Dataset for: An Experimental Comparison of Anomaly Detection Methods for Collaborative Robot Manipulators*. [Online]. Available: https://zenodo.org/record/5849300#.Y44M-3bMIuU

[8] E. Khalastchi and M. Kalech, "On fault detection and diagnosis in robotic systems," *ACM Comput. Surveys*, vol. 51, no. 1, pp. 1–24, Jan. 2019.

[9] V. Sathish, S. Ramaswamy, and S. Butail, "Training data selection criteria for detecting failures in industrial robots," *IFAC-PapersOnLine*, vol. 49, no. 1, pp. 385–390, 2016.

[10] A. Goncharov, A. Savelev, N. Krinitsyn, and S. Mikhalevich, "Automated anomalies detection in the work of industrial robots," in *Proc. IOP Conf. Mater. Sci. Eng.*, vol. 1019, no. 1. Bristol, U.K.: IOP Publishing, 2021, Art. no. 012095.

[11] T. Borgi, A. Hidri, B. Neef, and M. S. Naceur, "Data analytics for predictive maintenance of industrial robots," in *Proc. Int. Conf. Adv. Syst. Electric Technol. (IC_ASET)*, Jan. 2017, pp. 412–417.

[12] A. H. Sabry, F. H. Nordin, A. H. Sabry, and M. Z. A. Ab Kadir, "Fault detection and diagnosis of industrial robot based on power consumption modeling," *IEEE Trans. Ind. Electron.*, vol. 67, no. 9, pp. 7929–7940, Sep. 2020.

[13] H. Wu, Y. Guan, and J. Rojas, "A latent state-based multimodal execution monitor with anomaly detection and classification for robot introspection," *Appl. Sci.*, vol. 9, no. 6, p. 1072, Mar. 2019.

[14] J. V. Gael and Z. Ghahramani, *Nonparametric Hidden Markov Models*. Cambridge, U.K.: Cambridge Univ. Press, 2011, pp. 317–340.

[15] L. Basora, X. Olive, and T. Dubot, "Recent advances in anomaly detection methods applied to aviation," *Aerospace*, vol. 6, no. 11, p. 117, Oct. 2019.

[16] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[17] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1544–1551, Jul. 2018.

[18] 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*, vol. 8, pp. 47072–47081, 2020.

[19] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.

[20] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *J. Amer. Stat. Assoc.*, vol. 112, no. 518, pp. 859–877, Apr. 2017.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[22] Y. Zhou, D. Arpit, I. Nwogu, and V. Govindaraju, "Is joint training better for deep auto-encoders?" 2014, *arXiv:1405.1380*.

[23] D. Im, S. Ahn, R. Memisevic, and Y. Bengio, "Denoising criterion for variational auto-encoding framework," in *Proc. AAAI Conf. Artif. Intell.*, vol. 31, no. 1, 2017, pp. 1–7.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Jun. 1998.

[25] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," 2019, *arXiv:1901.03407*.

[26] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," 2020, *arXiv:2002.04236*.

[27] C. C. Aggarwal, *Outlier Analysis*, 2nd ed. Cham, Switzerland: Springer, 2016.

[28] E. Khalastchi, M. Kalech, G. A. Kaminka, and R. Lin, "Online data-driven anomaly detection in autonomous robots," *Knowl. Inf. Syst.*, vol. 43, no. 3, pp. 657–688, Jun. 2015.

[29] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLoS ONE*, vol. 11, no. 4, Apr. 2016, Art. no. e0152173.

[30] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[31] R. A. Brown, "Building a balanced k-d tree in O(kn log n) time," 2014, *arXiv:1410.5420*.

[32] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 93–104.

[33] C. C. Aggarwal and S. Sathe, *Outlier Ensembles: An Introduction*. Cham, Switzerland: Springer, 2017.

[34] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga, "Outlier detection with autoencoder ensembles," in *Proc. SIAM Int. Conf. Data Mining*. Philadelphia, PA, USA: SIAM, 2017, pp. 90–98.

[35] H.-P. Kriegel, P. Kroger, E. Schubert, and A. Zimek, "Interpreting and unifying outlier scores," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2011, pp. 13–24.

[36] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, Jan. 2011.

[37] M. E. Tipping and C. M. Bishop, "Mixtures of probabilistic principal component analyzers," *Neural Comput.*, vol. 11, no. 2, pp. 443–482, Feb. 1999.

[38] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *Int. J. Comput. Vis.*, vol. 77, nos. 1–3, pp. 125–141, May 2008.

[39] M. Claesen and B. De Moor, "Hyperparameter search in machine learning," 2015, *arXiv:1502.02127*.

[40] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6765–6816, 2017.

[41] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi. (2019). *Kerastuner*. [Online]. Available: https://github.com/keras-team/keras-tuner

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Jan. 2012.

[44] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and M. Kudlur, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[45] A. A. Pol, V. Berger, C. Germain, G. Cerminara, and M. Pierini, "Anomaly detection with conditional variational autoencoders," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 1651–1657.

[46] C. Doersch, "Tutorial on variational autoencoders," 2016, *arXiv:1606.05908*.

[47] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 1–27, 2008.

[48] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[49] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *J. Big Data*, vol. 7, no. 1, pp. 1–41, Dec. 2020.

[50] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 387–395.

[51] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed, "DeepAnT: A deep learning approach for unsupervised anomaly detection in time series," *IEEE Access*, vol. 7, pp. 1991–2005, 2019.

[52] D. T. Tran, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Tensor representation in high-frequency financial data for price change prediction," in *Proc. IEEE Symp. Comput. Intell. (SSCI)*, Nov. 2017, pp. 1–7.

[53] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj, "Temporal attention-augmented bilinear network for financial time-series data analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 5, pp. 1407–1418, May 2019.

[54] X. Teng, Y.-R. Lin, and X. Wen, "Anomaly detection in dynamic networks using multi-view time-series hypersphere learning," in *Proc. ACM Conf. Inf. Knowl. Manage.*, Nov. 2017, pp. 827–836.
[55] J. Fan, S. Upadhye, and A. Worster, "Understanding receiver operating characteristic (ROC) curves," *CJEM*, vol. 8, no. 1, pp. 19–20, Jan. 2006.
[56] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," in *Proc. 23rd Int. Conf. Mach. Learn. (ICML)*, 2006, pp. 233–240.
[57] (2018). *UR5e Technical Details*. Universal Robots. [Online]. Available: https://www.universal-robots.com/media/1802778/ur5e-32528_ur_technical_details_.pdf
[58] M. Stenmark, M. Haaae, and E. A. Topp, "Simplified programming of re-usable skills on a safe industrial robot–prototype and evaluation," in *Proc. 12th ACM/IEEE Int. Conf. Hum.-Robot Interact. (HRI*, Mar. 2017, pp. 463–472.
[59] U. Robots. (2021). *Real-Time Data Exchange (RTDE) Guide*. Accessed: Sep. 9, 2020. [Online]. Available: https://www.universal-robots.com/articles/ur/real-time-data-exchange-rtde-guide/
[60] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020, doi: 10.1038/s41586-020-2649-2.
[61] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, Mar. 1947.
[62] R. B. d'Agostino, "An omnibus test of normality for moderate and large size samples," *Biometrika*, vol. 58, no. 2, pp. 341–348, 1971.
[63] Python Software Foundation. (2021). *Time Access and Conversions*. Accessed: Mar. 23, 2021. [Online]. Available: https://docs.python.org/3/library/time.html#time.process_time
[64] O. Alghushairy, R. Alsini, T. Soule, and X. Ma, "A review of local outlier factor algorithms for outlier detection in big data streams," *Big Data Cognit. Comput.*, vol. 5, no. 1, p. 1, Dec. 2020.

SØREN G. GRAABÆK received the B.Sc. degree in robot systems and the M.Sc. degree in advanced robotics technology from the Faculty of Engineering, University of Southern Denmark, Odense, Denmark, in 2019 and 2021, respectively. He is currently pursuing an industrial Ph.D. degree in robotics with the University of Southern Denmark and Universal Robots A/S. He is an Industrial Host with Universal Robots A/S. His current research interest includes modeling and control of industrial robots.

EMIL VINCENT ANCKER received the B.Sc. degree in robot systems and the M.Sc. degree in advanced robotics technology from the Faculty of Engineering, University of Southern Denmark, Odense, Denmark, in 2019 and 2021, respectively. He is currently a Research and Development Manager with Dynatest A/S, Ballerup, Denmark. His current research interests include machine learning, robotics, and computer vision.

ANDREAS RUNE FUGL received the M.Sc. and Ph.D. degrees in robotics from the University of Southern Denmark, Odense, Denmark, in 2008 and 2013, respectively. He is currently with Vitec Aloc A/S, where he is developing high-performance investment management systems. His current research interests include solid mechanics, numerical algorithms, parallel computation, and robot control algorithms.

ANDERS LYHNE CHRISTENSEN (Senior Member, IEEE) received the Ph.D. degree from IRIDIA-CoDE, Université Libre de Bruxelles, Belgium, in 2008. He is currently a Full Professor with the University of Southern Denmark, Odense, Denmark. He is also the Founder and the former Head of the Bio-Inspired Computation and Intelligent Machines Laboratory, ISCTE-IUL, Lisbon, Portugal. His current research interests include bio-inspired intelligence, multi-robot systems, and fault tolerance.