Learning the Noise of Failure: Intelligent System Tests for Robots

Felix Sygulla and Daniel Rixen

Abstract—Roboticists usually test new control software in simulation environments before evaluating its functionality on real-world robots. Simulations reduce the risk of damaging the hardware and can significantly increase the development process's efficiency in the form of automated system tests. However, many flaws in the software remain undetected in simulation data, revealing their harmful effects on the system only in time-consuming experiments. In reality, such irregularities are often easily recognized solely by the robot's airborne noise during operation. We propose a simulated noise estimate for the detection of failures in automated system tests of robots. The classification of flaws uses classical machine learning — a support vector machine — to identify different failure classes from the scalar noise estimate.

The methodology is evaluated on simulation data from the humanoid robot LOLA. The approach yields high failure detection accuracy with a low false-positive rate, enabling its use for stricter automated system tests. Results indicate that a single trained model may work for different robots. The proposed technique is provided to the community in the form of the open-source tool NoisyTest, making it easy to test data from any robot. In a broader scope, the technique may empower real-world automated system tests without human evaluation of success or failure.

I. INTRODUCTION

Experimental testing of new control algorithms for robots is often time-consuming and sometimes a safety issue. However, it is common to reduce the time spent on experiments and run simulated system tests of the robot with virtual hardware and environment first. The robot can be tested safely by simulating its movement using full multi-body dynamics, environment model, and actuator/sensor characteristics. Failure evaluation can either be done by manual inspection of the simulation results or as part of automated system tests in combination with distinct conditions for failure or success. Based on the granularity of the used simulation models, the results can be quite close to real-world observations — although there will certainly be unmodeled effects, which require final experimental tests.

From our experience with testing algorithms on the biped robot platform LOLA (Fig. 1), we found that simulation data often contains clues on flaws in the robot’s behavior. Still, these remain undetected in an automated or manual inspection. In all these cases — one could term them soft failures — the robot reached its high-level goal, i.e., it did not fall or incline to an unexpected degree. In the corresponding experiments, however, one could immediately hear something is wrong — just from the machine’s noise. The reason was, for example, an oscillating feedback loop, or hard touchdown of the feet. We also found that clues for these failures are hidden in the simulation results. So the question we want to answer in this paper is: Can we use that information for system testing?

We propose a method to assess simulated system tests based on an estimated noise of the simulated robot. The noise signal is analyzed using classical machine learning approaches to distinguish between normal and failure-related noise. Typical fault detection and fault diagnosis approaches for robots focus on detecting failures during their use in a production environment — caused by faulty sensors or actuators. [1]. Common techniques for manipulators use the residual of a model-based observer on the system’s dynamics to detect and isolate actuator failures. [2]–[4]. To the authors’ knowledge, there is no work on the use of these techniques for the assessment of system tests in the development stage. Vibration signals are, however, commonly analyzed with machine learning approaches for fault diagnosis of rotating machinery, [5]. Typically, measured acceleration signals are transformed into the frequency domain, and the levels of individual subbands are then fed into neural networks or support vector machines, [6], [7]. Alternative approaches use wavelet transform coefficients [8] or a large set of features from classical signal analysis [9] as input data. In contrast, the work presented here uses a simulated (acoustic) noise — not a vibration (acceleration) signal — as input. The pre-processing pipeline uses a short-time Fourier transformation of the estimated noise signal and compresses the resulting spectrum instead of using individual frequency subbands.

This paper’s main contribution is a methodology to estimate the robot noise in simulation and detect soft failures.
based on features in this signal. The article is organized as follows: Section II describes the used simulation environment and generated noise data set. In Section III, the preprocessing pipeline and classification approach based on support vector machines are explained. Section IV describes the conducted hyper-parameter search and presents the accuracy results on a validation data set. Furthermore, we discuss the generalizability of the approach. Section V concludes the article and proposes ideas for future work and use cases.

II. SIMULATION ENVIRONMENT AND DATASET

The proposed failure detection method is applied to simulation data of the biped walking research platform LOLA. The robot is actuated by 24 electro-mechanical drives; see Fig. 1 for its kinematics structure. More details on the hardware design are described in [10], [11].

The custom simulation environment for LOLA includes models of the full rigid multi-body dynamics, the contacts with the environment, and the drive modules consisting of a decentralized joint controller, electrical system, motor, and gearbox. For details we refer the reader to [12], [13]. The simulation software also includes instances of the Walking Pattern Generator (WPG) [14] and the Stabilization and Inverse Kinematics (SIK) [15] modules to control the humanoid in its virtual environment. For a typical simulation run, high-level commands are sent to the WPG module to have the robot execute a certain movement (e.g. walk straight). In addition, obstacles (in this case unknown to the control software) are added to the environment. For a typical simulation run, high-level commands are sent to the WPG module to have the robot execute a certain movement (e.g. walk straight).

A. Noise Estimation

In this paper, we use an estimated scalar noise signal calculated from raw simulation data to detect and classify possible soft failures of the machine. We assume the robot is built of rotational joints only. To derive our approximation of the noise radiated by the virtual robot, let's first consider a rigid body \( i \) with surface \( S_i \), which is attached to a parent body \( i-1 \) via the rotational degree of freedom \( q_i \). The body \( i \) rotates around the point \( O_i \) of its parent. The radiated sound power of this body is then defined as [16]

\[
P_i(t) = \rho c \sigma_i S_i v_{S_i}^2(t),
\]

with the density of the air \( \rho \), the speed of sound \( c \), the radiation efficiency \( \sigma_i \), and the average velocity over all points \( P \) of the body's surface

\[
v_{S_i}(t) = \frac{1}{S_i} \int_{S_i} \|v_P\| \, dP.
\]

The radiation efficiency describes losses caused from the body to air transition and the fact that the average surface velocity is not on all parts of the body perpendicular to its surface. Using rigid body kinematics, the average surface velocity can be rewritten as

\[
v_{S_i}(t) = \frac{1}{S_i} \int_{S_i} \|v_O\| + \|\omega_{i-1} \times r_{iP}\| \, dP,
\]

with the velocity of the pivot point \( v_O \), the angular velocity of the parent body, \( \omega_{i-1} \), the relative angular velocity of the body, \( \omega_{i-1,i} \), which depends only on \( q_i \), and the position of a point \( P \) on the surface of the body, \( r_{iP} \). At this point, we scrutinize the exactness of the virtual noise signal for the sake of simplicity. The goal is to generate a signal, which does contain all information of the correctly calculated noise, but may deviate in scaling and amplitude. The terms \( A \) and \( B \) in (3) do not depend on the degree of freedom \( q_i \) of the considered body \( i \), but on the movement of all parent bodies of \( i \). We drop these terms and consider only the relative movement of every body in its surface velocity quantifier \( \hat{\nu}_{S_i} \).

Furthermore, term \( C \) can be simplified, because the rotation of a link is one-dimensional. This yields

\[
\hat{\nu}_{S_i}(t) = \frac{1}{S_i} \int_{S_i} \hat{q}_i l_P \, dP = \hat{q}_i \frac{1}{S_i} \int_{S_i} l_P \, dP,
\]

for body \( i \). The total sound power quantifier for all \( n \) joints can be written as

\[
\hat{P}(t) = ||\hat{q}||^2(t),
\]

with \( \hat{q} \in \mathbb{R}^n \). This already gives an estimation of the radiated sound power caused from the actual movement of the robot's joints. The scaling between the noise coming from the individual joints is lost with this approximation. We are, however, interested in time-local features of the noise, not the relative weighing between certain noise sources. In addition, we add the sound power related to the desired movement of the robot to get

\[
\hat{P}(t) = ||\hat{q}||^2(t) + ||\hat{q}^d||^2(t),
\]

with the desired joint velocities \( \hat{q}^d \in \mathbb{R}^n \). This ensures that features only visible in the desired movement — \( \hat{q}^d \) — will in general not follow the desired \( \hat{q}^d \) — also are part of the noise estimate. By assuming radial and even distribution of this sound power, the observed pressure at a large distance \( d \) to the bodies is [16]:

\[
p(t) = \sqrt{\hat{P}(t) \rho c}{4\pi d^2}.
\]
By setting all scaling constants to 1 and inserting (7), we obtain the final estimated noise pressure quantifier

$$
\hat{p}(t) = \sqrt{||\dot{q}||^2 + ||\ddot{q}||^2}.
$$

(9)

In the case of LOLA, we use $\mathbf{q} = [\mathbf{q}_f, \mathbf{q}_j]^T \in \mathbb{R}^n$, $\mathbf{q}^d = \mathbf{q}^d_j \in \mathbb{R}^m$ for the noise estimation with $n = 30$, $m = 24$.

B. Considered Scenarios and Failure Classes

We consider several test scenarios for the generation of training and validation data with the robot simulation. In all those scenarios, the robot does not fall; all these tests are considered as success by the implemented classical system tests. Still, the following undesired failure classes may occur in these simulations:

- **Hard impacts with the environment**: Faulty trajectories of the feet may lead to impacts on the ground - even for walking on level ground.
- **Unusually high joint accelerations**: Badly parametrized control systems without limitation of the actuating variable may lead to velocity jumps or otherwise high accelerations.
- **Oscillations**: Unstable control loops may lead to oscillations of the actuated bodies. The amplitude of these vibrations is usually small and does not cause the robot to fall — still it is an undesired behavior.

Based on the failure classes, we define 4 possible labels for a block of time-domain noise signal data: $\text{cls} = \{\text{OK}, \text{Impact}, \text{HighAcc}, \text{Oscillations}\}$. The noise signal data is generated from simulation runs for the scenarios defined in Table II. The scenarios were designed to generate noise data for the failure classes in multiple ways, while still having a relation to real-world problems. The data is manually annotated in the time-domain, i.e. every time block of simulation data is linked to one label and has a different block length — start and end time of a block is set manually based on the data. The total set of 83 time-data blocks corresponds to a total of 45.7 s annotated noise signal training data, and 27.6 s annotated noise signal validation data — both at a sample rate of $f_s = 10$ kHz. Note that in the scenarios 5-9, 20-21, and 22 the impact or high accelerations are actually expected and the simulations correspond to the best the control algorithms can do in these scenarios. Still, they are considered as failure training data, because their occurrence in other situations may indicate a fault. We consider it up to a higher-level system test software to interpret NoisyTest’s results depending on the context of the tested scenario.

### III. Classification Approach

This section describes the classification approach used to automatically learn the relations between the noise estimation signal and the annotated failure labels. In the following, several parameters for the preprocessing pipeline and classifier are introduced. Variables marked with a superscript * are subject to a later hyper-parameter search, see Subsection IV-A. The values of all other parameters are directly motivated within this section.

A. Data Preprocessing

The preprocessing step transforms the estimated noise signal in time-domain to a meaningful, dense representation with preferably high relation to the failure classes. The structure of the preprocessing pipeline is motivated by techniques used in speech recognition systems. In the following, all steps of the pipeline are described.

1) Framing: The incoming labeled noise signal blocks all have a different duration $l$ number of samples. By framing the data, all blocks are transformed to the same number of samples $n_f$. We expect failure patterns of the noise signal to fit in a time frame of $\Delta t_F = 0.4$ s; consequently, we set the frame size to $n_f = 4000$ samples. Given a data block with arbitrary size $n_c$, the following cases exist:

- (1) When $n_c \geq n_f$, we extract several overlapping frames of size $n_f$ using a stride length $s = 1000$ samples for a sliding window. This increases the total number of training frames and perturbates the time-domain data by time shifts. All extracted frames share the same label data of the original data block. Only windows which completely fit into the larger data block are extracted. The parameters $n_f$ and $s$ are set based on expert knowledge of possible data patterns and a coarse parameter search. They directly influence the size and composition of the data set and are considered fixed in the following.

- (2) When $n_c < n_f$, we pad the block with $n_f - n_c$ samples at its end by reflecting the samples at the edge value both in the direction of the time and amplitude axis, see Fig. 3. This way, the padded frame is continuous at the edge points and resembles the original data in a repeated manner. Padding is only required during training to avoid mixing data with different failures when these failures occur in a very short time. Blocks of the validation data set are never padded. Instead, always the size of one frame is used, which means the block is extended to cover more noise data than originally specified by manual annotation.

The output of this preprocessing step are $N_T = 291$ training data frames of $n_f$ time samples each, and $N_V = 191$ validation data frames. The failure class distribution of the data set is shown in Fig. 2.
2) Spectrogram Generation: In the second step, we decorrelate the time series data by transformation to the frequency domain. This step helps to improve the performance of downstream machine learning approaches, which perform best for uncorrelated input data. To preserve information on the noise signal changes over time, a short-time Fourier transformation is performed on every frame. We use a $n_{tr} = 1024$ points Fast Fourier transform (FFT) with Hann window, and a stride length $s_{tr}$, resulting in $n_{t}$ different transforms. The frame data is not padded at its end. We choose the FFT length based on the minimum resolvable frequency $f_{min} = \frac{n_{tr}}{T} \approx 9.8$ Hz — a value where typically the lowest mechanical eigenfrequencies of robots reside. \[17\]. The amplitudes resulting from the FFTs are used for the spectrogram; phase data is not used. The output of this preprocessing step is a $[N_{TF} \times n_{t} \times 513]$ training data tensor $/ [N_{TV} \times n_{t} \times 513]$ validation data tensor.

3) Frequency Domain Compression: The resolution of the Fourier transform in frequency domain is $\frac{f_{max}}{n_{tr}} \approx 9.8$ Hz, which leads to $\frac{n_{tr}}{2} + 1 = 513$ frequency bins for the frequency range $[0, \frac{f_{max}}{2}]$. Several tests of the whole classification architecture have shown this high resolution to be unnecessary for the classification task. Thus, we compress the frequency spectrum with a factor of 3 by binning the frequency samples into $\frac{n_{tr}}{2} + 1 = 171$ linear-spaced bins. This step has shown to significantly improve the accuracy on the validation data set, as it assumingly reduces overfitting the training data. The amplitudes of the spectrogram are log-scaled after compression. The output of this preprocessing step is a $[N_{TF} \times n_{t} \times 171]$ training data tensor $/ [N_{TV} \times n_{t} \times 171]$ validation data tensor.

4) Resolving Time-Locality: The spectrogram contains information on the evolution of the frequency spectrum over time. When directly using this data as input for a classical machine learning approach, the input data for two identical events, which occur at different points in time, are different. Due to this (time-) locality of the data, finding a valid model for all possible timings is difficult. Therefore, we apply a 1D type-II Discrete Cosine Transform (DCT) in orthogonal form on every frequency row of the spectrogram:

$$\text{dct} : \mathbb{R}^{n_{t}} \to \mathbb{R}^{n_{t}}$$

$$x_n \mapsto X_n, \text{ with}$$

$$X_0 = \frac{1}{\sqrt{n_{t}}} \sum_{n=0}^{n_{t}-1} x_n,$$

$$X_k = \sqrt{\frac{2}{n_{t}}} \sum_{n=0}^{n_{t}-1} x_n \cos \left[ \frac{\pi}{2} \left( n + \frac{1}{2} \right) k \right], \quad k = 1, \ldots, n_{t} - 1.$$  \[10\]

This way, time-locality is resolved and the temporal evolution is stored in form of DCT coefficients for every frequency bin. The final preprocessing step flattens the output to a $[N_{TF} \times 171 \cdot n_{t}]$ training data tensor $/ [N_{TV} \times 171 \cdot n_{t}]$ validation data tensor.

B. Support Vector Machine Classifier

For the classification, we apply a classical machine learning approach, a Support Vector Machine (SVM). SVMs have the advantage to be effective in high dimensional input spaces, even when the number of features in input data is higher than the size of the dataset — which is the case here. Furthermore, they are resource efficient and provide adequate training times for small datasets.

1) Two-Class SVM Formulation: We use the libsvm \[13\] implementation of the C Support Vector Classification (C-SVC) approach \[19\]. \[20\]. Given a data set of $n$-dimensional vectors $x_i \in \mathbb{R}^n, i = 1, \ldots, m$, and corresponding binary label data $y_i \in \{1, -1\}$, C-SVC looks for the
weights $w \in \mathbb{R}^l$ and bias $b$ by solving the following primal optimization problem [13]:

$$
\begin{align*}
\min_{w,b,\xi} & \quad \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i \\
\text{s.t.} & \quad y_i (w^T \phi(x_i) + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\
& \quad \xi_i \geq 0, \quad i = 1, \ldots, m,
\end{align*}
$$

where $\phi(x_i) : \mathbb{R}^n \to \mathbb{R}^l$ maps into the higher-dimensional space $l > n$, and $C > 0$ is a regularization parameter with the constraint error variables $\xi_i$. The value of $C$ is inversely proportional to the $l_1$ regularization of the weights $w$, which helps to avoid overfitting the training data. Regularization of $w$ tries to keep the weights minimal, i.e. use a minimal set of weights to represent a decision boundary.

For a high dimensional $w$, it is computationally expensive or even unfeasible to solve the problem in its primal form. Instead, the corresponding dual formulation is used:

$$
\begin{align*}
\max_{\alpha} & \quad -\frac{1}{2} \alpha^T Q \alpha + \sum_{i=1}^{m} \alpha_i \\
\text{s.t.} & \quad y^T \alpha = 0, \\
& \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m,
\end{align*}
$$

where $Q_{ij} = y_i y_j K(x_i, x_j) = y_i y_j \phi(x_i)^T \phi(x_j)$. With this formulation, an explicit calculation of the high-dimensional vector $\phi(x_i)$ is no longer necessary. Instead, the scalar kernel function $K(x_i, x_j)$ is directly calculated from its arguments without explicit mapping to the $l$-dimensional space. At the optimal solution, furthermore, only a few of the lagrangian multipliers $\alpha_i$ are non-zero — namely those, which correspond to the support vectors. This set of input data vectors $x_i$ defines the decision boundaries in the input data hyperplanes.

Common choices for the kernel function are the linear kernel $K_{\text{linear}}(x_i, x_j) = x_i^T x_j$, and the radial basis function (RBF) kernel $K_{\text{rbf}} = \exp(-\gamma^* ||x_i - x_j||^2)$. From the solution of the dual problem and the primal-dual relationship, the decision function for classification of an input vector $x$ results:

$$
\hat{y} = \text{sgn} \left( w^T \phi(x) + b \right) = \text{sgn} \left( \sum_{i=1}^{m} y_i \alpha_i K(x_i, x) + b \right).
$$

2) Multi-Class SVM Formulation: To extend the binary SVM classification to multiple classes in the data, the “one-vs-rest” approach is deployed. For each class $c$ in the dataset, a separate binary C-SVC is used to learn the appearance of the class in the data. The C-SVCs share the same parameters and kernel, except for the regularization

Fig. 3: Exemplary visualization of the 4 preprocessing steps. a) shows framed time data. The data is padded at its actual end ($t = 0.2$ s). b) and c) show the original and compressed spectrograms, respectively. This visualization uses $s_{nn}^* = 834$. The data in b) is additionally log-scaled to allow visualization. d) shows the final DCT coefficients (before flattening).
TABLE II: Optimal hyperparameter values and terms resulting from this selection.

| Parameter          | Linear kernel | RBF kernel |
|--------------------|---------------|------------|
| FFT stride length  | 401           | 834        |
| Inv. regularization| 1.0           | 1.1        |
| Kernel parameter   | 0.04          | 5.7e-04    |
| $\alpha$           | 8             | 4          |
| SVM input vector size | 1368      | 684        |

TABLE III: Achieved accuracies on the validation data set with the linear kernel

| OK       | Impact | HighAcc | Oscillations |
|----------|--------|---------|--------------|
| False negative errors | 0.0 %  | 1.0 %  | 5.8 %  | 4.7 % |
| False positive errors  | 5.8 %  | 5.8 %  | 0.0 %  | 0.0 % |
| Failure detection rate | 94.2 % |         |         |       |
| Subset accuracy        | 88.5 % |         |         |       |

TABLE IV: Achieved accuracies on the validation data set with the RBF kernel

| OK       | Impact | HighAcc | Oscillations |
|----------|--------|---------|--------------|
| False negative errors | 0.5 %  | 2.6 %  | 6.3 %  | 0.0 % |
| False positive errors  | 2.1 %  | 2.6 %  | 0.0 %  | 4.7 % |
| Failure detection rate | 97.9 % |         |         |       |
| Subset accuracy        | 90.6 % |         |         |       |

Fig. 4: Accuracy for a variation of one parameter from its optimal value for the RBF kernel.

parameters $C_{cls}$, which are set inversely proportional to the respective class frequency $\frac{N_{T,cls}}{N_T}$ of the training data set:

$$C_{cls} = \frac{1}{4} \frac{N_{T,cls}}{N_T} \hat{C}^*.$$  

The global regularization parameter $\hat{C}^*$ is subject to the hyperparameter search described in Section 4.

IV. RESULTS

NoisyTest is implemented in python3. It uses scikit-learn\(^1\) wrapper to libsvm\(^2\) for the support vector machine, and TensorFlow\(^3\) for data preprocessing. The code\([21]\) and used data set\([22]\) are available online.

A. Hyperparameter Search

The preprocessing pipeline and the SVM implementation have a set of hyperparameters, which need to be tuned to the data set and application for maximum performance. These parameters are the FFT window stride length $s_{f^2}$, the regularization parameter $\hat{C}^*$, and the kernel parameter $\gamma^*$ for radial basis function kernel. The best set of parameters maximizes the accuracy for all classes (subset accuracy) on the validation data set. It is found with a grid search by training the SVM for every parameter combination and evaluating the accuracy on the validation data set. For more details, refer to the implementation. The resulting optimal hyperparameters are shown in Table I. The sensitivity of the hyperparameters for the RBF kernel is shown in Fig. 4.

B. Accuracy and Discussion

The accuracy of NoisyTest is evaluated on the validation data set. Due to the relatively low sensitivity of the hyperparameters (see Fig. 4) and the small data set we are not using an extra test data set. The results for the linear kernel are summarized in Table III accuracies with the radial basis function kernel are shown in Table IV. Both approaches perform well in terms of the overall detection of failures (OK vs. rest). The RBF kernel performs best in terms of subset accuracy. However, it shows a non-zero false-negative rate for the OK class, i.e., a non-zero false-negative rate for the detection of a failure. We focus on the results of the RBF kernel variant in the following. From the shown accuracy data — and additional evaluations with reduced validation data set — we found that the undetected failures are caused by the Impact class of scenario 21. Furthermore, the HighAcc events in scenario 24 are falsely detected as Impact and Oscillations classes. The torque signal jumps induced in this scenario, however, are hard to classify as closely related to the effects of an impact and as they certainly also lead to subsequent oscillations. In reality, it is somehow likely that the classes sometimes only occur together or in close succession. Nevertheless, identifying the possible failure class (instead of just a single-class evaluation) may provide the user with additional hints on the failure cause. We conducted a test to evaluate if the SVM learns the concrete features of a failure independently of robot-specific noise characteristics. For this check, we generated a reduced noise estimate for LOLA based not on all joints, but only on the knee and ankle joints of both feet. The reduced number of involved joints results in a completely different time-domain signal. Evaluating the trained SVM with RBF kernel on a separated validation set with the reduced noise estimate still gives reasonable detection performance, see Table V. This result indicates the possibility of using a single trained model for different types of robots.

V. CONCLUSION

This paper presents the idea and implementation of NoisyTest, a tool for virtual system tests based on the estimated noise of a robot. We show that machine learning techniques on a scalar noise signal, generated from the simulated robot movement, detect different failures with

---

\(^1\)https://scikit-learn.org

\(^2\)https://tensorflow.org
high accuracy. The results show the generalizability of the approach to different robot types, even when the same trained model is used. *NoisyTest* is primarily intended for simulated system tests to help speed up the development of robot control software by finding failures in simulation data, which are hard to identify by manual inspection and are often only recognized during real-world experiments. It helps to generate automated system test pipelines with a more precise definition of success than conventional heuristic methods can offer.

Currently, we use this technique for a general test of simulation runs. However, several extensions are considered for the future. First, it might be possible to run *NoisyTest* on several noise estimates generated from intermediate signals in the control algorithm’s data flow to localize the faulty code. Second, it is interesting to test its performance on noise estimates generated from experimental data. A possible use case is a fully automatic system test of the actual robot system with a noise-based evaluation of its performance. Furthermore, the used classification technique is computationally efficient, presumably allowing online real-time diagnostics of the robot.

We strive for the extension of the failure classes and the application to more robots. We invite other researchers to use *NoisyTest* on their system and contribute to a shared tool for failure detection in robotic system development.

### References

[1] E. Khalastchi and M. Kalech, “On fault detection and diagnosis in robotic systems,” *ACM Computing Surveys*, vol. 51, no. 1, pp. 1–24, 2018.

[2] A. T. Vemuri, M. M. Polycarpou, and S. A. Diakoutris, “Neural network based fault detection in robotic manipulators,” *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 342–348, 1998.

[3] W. E. Dixon, I. D. Walker, D. M. Dawson, and J. P. Hartman, “Fault detection for robot manipulators with parametric uncertainty: A prediction-error-based approach,” *IEEE Transactions on Robotics and Automation*, vol. 16, no. 6, pp. 689–699, 2000.

[4] A. De Luca and R. Mattone, “An identification scheme for robot actuator faults,” *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 2613–2617, 2005.

[5] R. Liu, B. Yang, E. Zio, and X. Chen, “Artificial intelligence for fault diagnosis of rotating machinery: A review,” *Mechanical Systems and Signal Processing*, vol. 108, pp. 33–47, 2018.

[6] D. Martínez-Regó, O. Fontenla-Romero, and A. Alonso-Betanzos, “Power wind mill fault detection via one-class ν-SVM vibration signal analysis,” *Proceedings of the International Joint Conference on Neural Networks*, no. May 2016, pp. 511–518, 2011.

[7] D. Fernández-Francos, D. Martínez-Regó, O. Fontenla-Romero, and A. Alonso-Betanzos, “Automatic bearing fault diagnosis based on one-class m-SVM,” *Computers and Industrial Engineering*, vol. 64, no. 1, pp. 357–365, 2013.

[8] S. Guo, T. Yang, W. Gao, and C. Zhang, “A novel fault diagnosis method for rotating machinery based on a convolutional neural network,” *Sensors (Switzerland)*, vol. 15, no. 3, pp. 5627–5648, 2015.

[9] S. Lohmeier, *Design and Realization of a Humanoid Robot for Fast and Autonomous Bipedal Locomotion*. Dissertation. Technical University of Munich, 2010. [Online]. Available: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20101126-980754-1-4

[10] F. Sygulla, R. Wittmann, P. Seiwald, T. Berninger, A.-C. Hildebrandt, D. Wahrmann, and D. Rixen, “An EtherCAT-Based Real-Time Control System Architecture for Humanoid Robots,” in *2018 IEEE/RSJ International Conference on Automation Science and Engineering*, 2018.

[11] T. Buschmann, *Simulation and Control of Biped Walking Robots*, Dissertation, Technical University of Munich, 2010. [Online]. Available: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20101201-997204-1-6

[12] V. Favot, *Hierarchical Joint Control of Humanoid Robots*. Dissertation, Technical University of Munich, 2016. [Online]. Available: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20161206-1294180-1-9

[13] P. Seiwald, F. Sygulla, N. S. Staufenberg, and D. Rixen, “Quintic Spline Collocation for Real-Time Biped Walking-Pattern Generation with variable Torso Height,” in *IEEE/RSJ International Conference on Humanoid Robots*, 2019, pp. 56–63.

[14] F. Sygulla and D. Rixen, “A force-control scheme for biped robots to walk over uneven terrain including partial footholds,” *International Journal of Advanced Robotic Systems*, vol. 17, no. 1, pp. 1–14, 2020.

[15] L. Cremer, M. Heckl, and B. Petersson, *Structure-Borne Sound*. Springer-Verlag Berlin Heidelberg, 2005.

[16] T. F. C. Berninger, F. Sygulla, S. Fuderer, and D. J. Rixen, “Experimental Analysis of Structural Vibration Problems of a Biped Walking Robot,” in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 8726–8731.

[17] C. C. Chang and C. J. Lin, “LIBSVM: A Library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, 2011.

[18] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “Training algorithm for optimal margin classifiers,” in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, Association for Computing Machinery, 1992, pp. 144–152.

[19] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[20] F. Sygulla, “NoisyTest: Noise-Based System Test Tool,” 2020. [Online]. Available: https://github.com/am-lola/noisytest

[21] ———, “NoisyTest Dataset for Lola,” 2020. [Online]. Available: https://github.com/am-lola/noisytest-data-lola

---

**TABLE V: Achieved RBF-kernel SVM accuracies on the validation data set with reduced noise estimate. Only the ankle and knee joints of both feet are used.**

|                | OK     | Impact | HighAcc | Oscillations |
|----------------|--------|--------|---------|--------------|
| False negative errors | 1.6 %  | 4.2 %  | 6.3 %   | 7.3 %        |
| False positive errors  | 9.4 %  | 7.9 %  | 0.0 %   | 2.1 %        |
| Failure detection rate | 90.6 % |        |         |              |
| Subset accuracy       | 80.6 % |        |         |              |

---

**ACM Computing Surveys**, vol. 14, no. 2, pp. 342–348, 1998.

**IEEE Transactions on Robotics and Automation**, vol. 10, no. 6, pp. 689–699, 2000.

**Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory**. Association for Computing Machinery, 1992, pp. 144–152.

**Machine Learning**, vol. 20, no. 3, pp. 273–297, 1995.

**ACM Transactions on Intelligent Systems and Technology**, vol. 2, no. 3, 2011.

**Computers and Industrial Engineering**, vol. 64, no. 1, pp. 357–365, 2015.

**IEEE Transactions on Robotics and Automation**, vol. 10, no. 6, pp. 689–699, 2000.

**IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS**, pp. 2613–2617, 2005.

**Mechanical Systems and Signal Processing**, vol. 108, pp. 33–47, 2018.

**Proceedings of the International Joint Conference on Neural Networks**, no. May 2016, pp. 511–518, 2011.

**Computers and Industrial Engineering**, vol. 64, no. 1, pp. 357–365, 2013.

**Sensors (Switzerland)**, vol. 15, no. 3, pp. 5627–5648, 2015.

**2018 IEEE/RSJ International Conference on Automation Science and Engineering**, 2018.

**IEEE/RSJ International Conference on Humanoid Robots**, 2019, pp. 56–63.

**International Journal of Advanced Robotic Systems**, vol. 17, no. 1, pp. 1–14, 2020.

**Structure-Borne Sound**. Springer-Verlag Berlin Heidelberg, 2005.

**IEEE International Conference on Robotics and Automation (ICRA)**. IEEE, 2020, pp. 8726–8731.

**LIBSVM: A Library for support vector machines**. *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, 2011.

**Training algorithm for optimal margin classifiers**. in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*. Association for Computing Machinery, 1992, pp. 144–152.

**Support-vector networks**. *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

**NoisyTest: Noise-Based System Test Tool**. 2020. [Online]. Available: https://github.com/am-lola/noisytest

**NoisyTest Dataset for Lola**. 2020. [Online]. Available: https://github.com/am-lola/noisytest-data-lola