An algorithm for sparse piece-wise polynomial residual generation for anomalies detection in industrial manipulator robots.

Mazen Alamir[a,*], Sacha Clavel[b,a]

[a] *Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, 38000 Grenoble, France*
[b] *Staubli company. Auvergne-Rhône-Alpes, 74210, Faverges, France.*

Abstract

This paper addresses the characterization of normality in industrial robots which is a crucial step in anomaly detection without an a priori knowledge of the set of faulty behaviors. This is done using piece-wise multi-variate sparse polynomials involved in the identification of implicit relationships between sensors measurements. These relationships are then used to design tight residual generators for complex dynamical systems such as multi-link robot manipulators. An algorithm is provided and tested using real data collected on a 4-link `Staübli` robot. The precision of the residual is also compared to the one that might be obtained using the state-of-the art Long Short-Term Memory (LSTM) recurrent neural networks (RNNs) showing higher precision with far less complexity and drastically lower computation time. Moreover, some examples of use of the resulting residual generators in anomaly detection are provided showing promising preliminary results.

Keywords: Piece-wise polynomial relationships; Sparse Models; Anomaly Detection; Industrial Manipulator Robots.

1. INTRODUCTION

Although the mathematical *Euler-Lagrange* equations of the nominal behavior of robots are quite well mastered to derive "white-box" dynamic models, the use of neural networks (NN) to yield even more precise and complex models is now a well established practice with rather remarkable success [5, 4, 17]. The need for such data-driven models stem from the fact that the accuracy of historically used, purely white-box models, is limited by the difficulty in capturing complex non-linear dynamic phenomena such as friction, flexibility under rapid movements and backlash to cite but few ones. This justifies the use of data-driven models either in a fully "black-box" form by completely replacing the physical equations [6, 10, 20], or in a "gray-box" form where data-driven sub-models are used to capture specific parameters [16, 8, 9] or to reproduce the error of the mathematical model [7, 18, 19, 21].

Within the model learning literature, beside Neural Networks, several algorithms have already demonstrated their ability to help modeling the dynamics of industrial robots, including Locally Linear Regressors, Support Vector Regressors [11, 3], Gaussian Process Regressors [12, 2]. Nevertheless, the most promising approach seems to be the use of Recurrent Neural Networks [17, 18]. Thanks to their design, recurrent neural networks are perfectly suited to exploit temporal correlations, and thus to handle various tasks involving time-series. Well known recurrent networks such as Long Short-Term Memory (LSTM) networks or Gated Recurrent Units (GRU) networks are specifically built to take the most of temporal dependencies, even at a long term, and overcome the limitations traditionally faced when mining long time-series, such as the vanishing or exploding gradient problems. The use of LSTMs GRUs for dynamic modeling of robots enables reaching accuracies that outperform those achieved with more conventional methods [18, 19, 17].

While a small increase in the model's precision might not be significant for the control task which is precisely designed to compensate for small errors via the feedback mechanism, the relevance of having a tight modeling becomes even more justified when it comes to detect small anomalies inducing small signals that show up in the measurements stream. Indeed, the tighter the modeling error (also called the residual) around the predicted value, the smaller are the amplitudes of the anomalies that can be detected with confidence (see Fig.1).

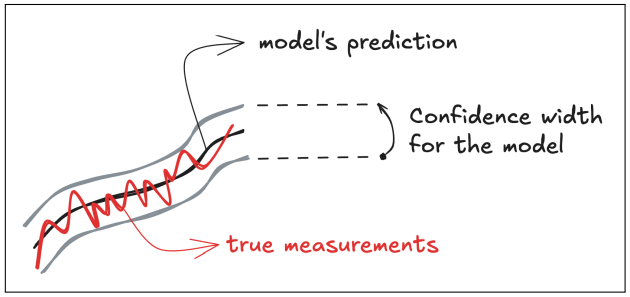
In its simplest form, the typical residual is based on a prediction error of a targeted label, denoted hereafter by y, and the following condition is used to fire an anomaly detection:

$$\texttt{if } \Big(\mathcal{M}[y(\cdot)|_{t-T}^{t} - \hat{y}(\cdot)|_{t-T}^{t}] > \eta\Big) \rightarrow \texttt{alarm} \tag{1}$$

where $y(\cdot)|_{t-T}^{t}$ is the measurements profile of y acquired over the time interval $[t-T, t]$, $\hat{y}(\cdot)|_{t-T}^{t}$ is the prediction of the same y, based on some prediction model, over the same time window, $\mathcal{M}[e(\cdot)|_{t-T}^{t}]$ is some measure of the error's profile while η is some *alarm-firing* threshold that is determined based on the statistics of the error produced on training data. For instance, polynomial thresholds [15] can be used once an acceptable false alarm's level `FA` is chosen, say 5%, together with a moment's

*Corresponding author
Email addresses: `mazen.alamir@grenoble-inp.fr` (Mazen Alamir), `s.clavel@staubli.com` (Sacha Clavel)

(a)



(b)

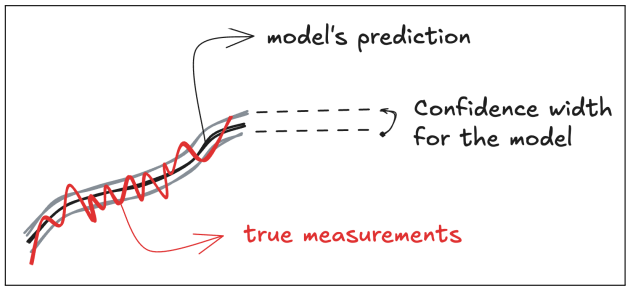


Figure 1: Example of fautly measurements (in red) with the nominal prediction (in black). (a) Non detection since the faulty measurements are within the large confidence interval. (b) possible detection when the confidence interval is tighter around the prediction.

order $r \in \mathbb{N}$. These parameters set the threshold η to:

$$\eta = \mu_{\mathcal{M}} + \left[\frac{\mu_r}{\mathtt{FA}}\right]^{1/r} \quad \text{where} \quad \mu_r := \mathbb{E}\left[|\mathcal{M} - \mu_{\mathcal{M}}|^r\right] \tag{2}$$

where $\mu_{\mathcal{M}}$ is the mean of $\mathcal{M}[e(\cdot)|_{t-T}^{t}]$ over all instances of t contained in the training data used to fit the model.

Notice that the use of a decision that is based on the observation of the error over a time window as expressed in (1), enhances the robustness of alarm against measurement outliers and noise. A possible definition of $\mathcal{M}$ might be the number of instants over the observation interval $[t-T, t]$ where $|e|$ exceeds some thresholds that would be defined using the statistics of errors on the training data. Defining this aspect is a necessary step that is still to be done regardless of the underlying model which produces the error e. Whatever is the latter, the level of its precision determines the quality of the final outcome. This precision is what the present contribution is about.

A typical situation is the one corresponding to $\hat{y}$ being the output of some *surrogate* model such as a NN model. Namely:

$$\hat{y}(t) = \mathtt{NN}(x(t), x(t-\tau), \ldots, x(t-n\tau)) \tag{3}$$

where $x(t) \in \mathbb{R}^{n_x}$ represents the measurements at instant t of the independent variables on which the targeted variable y depends. In the case of the four d.o.f manipulator robot considered in this paper for instance, $y = T_s$ is the torque applied at the s-th link of the robot while $x \in \mathbb{R}^{12}$ stands for the triplet of vectors $(q, \dot{q}, \ddot{q})$ representing the generalized coordinates with their two first derivatives at the four links.

In the present paper, we consider a more general measure of the prediction error that takes the following *implicit* form:

$$\mathcal{M}\Big[e(\cdot)\Big] \text{ where } e(t) := \varphi(x(t), y(t)) \tag{4}$$

$$:= \min_{\kappa \in \{1,\ldots,n_m\}} \Big[|y(t) - P_\kappa(x(t))|\Big] \tag{5}$$

where for each $\kappa \in \{1, \ldots, n_m\}$, $P_\kappa(\cdot)$ is a multivariate polynomial (MP) in the variable x. In other words, the error $e(t)$ at instant t is the distance between the measured value $y(t)$ and the *closest* value among the n_m values $P_\kappa(x(t))$ provided by the polynomials P_κ, $\kappa \in \{1, \ldots, n_m\}$ evaluated at the same argument $x(t)$.

It is important to underline the major difference between the standard approach where $\hat{y}$ stands for the predicted value of the targeted variable y leading to the residual being defined by $e = y - \hat{y}$ and the approach defined by (5) where the model does not provide a single prediction but n_m *candidate* values, namely $P_\kappa(x(t))$ for $\kappa \in \{1, \ldots, n_m\}$. Nevertheless, using these values, a residual can be defined by (5) and as long as it is a residual that is needed in (1) to characterize the domain of normal behavior, both approaches are legitimate and end up with a residual to be compared to a threshold.

Remark 1. *Notice that the error e provided by* (5) *can be artificially made as small as desired by taking a very large number n_m of* ***constant*** *polynomials taking each a specific value in the domain of possible values of the target y regardless of the value of the argument x. More precisely, denoting by $\Delta_y = y_{max} - y_{min}$, under the assumption of uniform distribution of the data inside the domain of excursion, the $q \in [0, 1]$ percentile of the error induced by this trivial solution would be given by:*

$$\mathtt{percentile}(|e|, q) = \frac{q\Delta_y}{2(n_m - 1)} \tag{6}$$

That is the reason why we shall systematically compare the achieved precision to the above trivially possible one. Notice that while using such high values of n_m reduces the residual, it might also reduce, and even ultimately remove, the detection capability as any value lying within the excursion observed during the training would lead to small residual regardless of the presence of faulty behavior. It is shown later on that in the proposed algorithm, n_m is not chosen by the designer, rather, it results from the required precision threshold `th` *used in the algorithm. The smaller* `th`*, the higher is the number n_m of sub-models needed to achieve it (see section 3 for more details).*

Notice that for a 4-links robot, four residual generators are considered which all share the same 12-dimensional state (feature) vector while each addresses a specific torque (T_s) that is applied at the s-th link with $s \in \{1, \ldots, 4\}$:

$$x := \begin{bmatrix} q \\ \dot{q} \\ \ddot{q} \end{bmatrix} \in \mathbb{R}^{12} \quad ; \quad y = T_s, \quad s \in \{1, \ldots, 4\} \tag{7}$$

This leads to 4 Piece-Wise Multivariate Polynomials (PWMP), consisting each of possibly different number n_{m_s} of polynomials. Notice that when describing the methodology, the index s might be dropped while keeping in mind that the methodology is to be applied identically to each instance of the problem that is defined for a specific value of $s \in \{1, \ldots, 4\}$.

Given the previously described challenges, the contributions of the present paper can be summarized as follows:

- A generic application-independent algorithm is proposed to derive, for a given targeted label, a set of sparse MPs $P_\kappa, \kappa = 1, \ldots, n_m$ involved in (5). The algorithm uses a recently developed sparse least squares solver [1] together with a specific technique that enables to explore the training data which is a tiny subset of the available working data. The fitting algorithm shows three major appealing features, namely: 1) Sparsity which is a key feature when it comes to avoid the risk of over-fitting in any learning paradigm. 2) The need for small amount of training data, at least when compared to NN, and 3) Shorter computation time that is orders-of-magnitude lower than the one needed by NN models and this while producing tighter residuals.

- The proposed algorithm is applied to sensors measurement acquired on a real `Staübli` 4-link manipulator robot (involving 12 states and 4 control inputs). The resulting four residual generators are then applied to a much larger test dataset in order to assess its generalization capabilities in producing small residuals with small number n_m of polynomials. The comparison with the residual generated by state-of-the-art NN models shows higher accuracies, orders of magnitude shorter computation times and far less active parameters involved in the resulting models.

- A proof of concept is provided regarding the use of the so designed residual generators to detect small anomalies leading to barely noticeable differences in the raw measurement time-series for a set of representative faulty behaviors. Moreover, the results suggest that the proposed framework enables not only the detection but the localization of the fault among the different axes.

The paper is organized as follows: Section 2 introduces some definitions and notation used throughout the paper and states the problem to be addressed by the proposed algorithm. Section 3 presents the proposed algorithm together with some formal results. The collected data and the data preparation steps are then presented in Section 4 before the results are discussed in Section 5. The paper ends with Section 6 that summarizes the paper's findings and discusses some possible future investigation tracks.

2. Definitions, notation and problem statement

Given a vector of features $x \in \mathbb{R}^{n_x}$, recall that a multi-variate polynomial (MP) $P(x)$ in x takes the following form:

$$P(x) = \sum_{j=1}^{n_c} c_j \phi_j(x) \text{ where } \phi_j(x) = \prod_{\ell=1}^{n_x} x_\ell^{p_{\ell j}} \tag{8}$$

where ϕ_j is referred to as the j-th monomial of P. The degree d_j of a monomial ϕ_j is defined by $d_j = \sum_{\ell=1}^{n_x} p_{\ell j}$. The degree of the polynomial P is the maximum degree of its monomials with non vanishing coefficient c_j, namely $d = \max_{j=1}^{n_c} \{d_j \mid c_j \neq 0\}$. Given the dimension n_x of x and the degree d of the polynomial, the number n_c of possible monomials is given by:

$$n_c = \binom{n_x + d}{d} \tag{9}$$

Now given a dataset consisting of collected measurements of x and some target variable y, namely:

$$\mathcal{D} := \left\{x^{(i)}, y^{(i)}\right\}_{i=1}^{n_D} \tag{10}$$

and given a polynomial degree d, the *polynomial features* matrix, denoted hereafter by $\Phi(\mathcal{D}, d)$, is the matrix having the element at row i and column j defined by:

$$[\Phi(\mathcal{D}, d)]_{i,j} := \phi_j(x^{(i)}) \tag{11}$$

where ϕ_j are the j-th monomial among the n_c candidate monomials. In other words, the column j of the matrix corresponds to the value of the j-th monomial at all the instances of the feature vector $\{x^{(i)}\}_{i=1}^{n_D}$ while the i-th row of the matrix corresponds to the vector of values of the monomials $\{\phi_j(\cdot)\}_{j=1}^{n_c}$ at the specific instance $x^{(i)}$, called also *sample*, of the features vector.

Having the polynomial features matrix Φ at hand, one might try to determine the vector of coefficients c appearing in (8) by solving the following linear system of equations in the unknown vector of coefficients $c \in \mathbb{R}^{n_c}$:

$$\mathcal{Y}(\mathcal{D}) \approx [\Phi(\mathcal{D}, d)]\, c \quad ; \quad c \in \mathbb{R}^{n_c} \tag{12}$$

where $\mathcal{Y}(\mathcal{D})$ is the vector of targeted labels in the dataset $\mathcal{D}$:

$$\mathcal{Y}(\mathcal{D}) := \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n_D)} \end{bmatrix} \tag{13}$$

Notice however that such an attempt assumes that the underlying relationships are polynomial over all the domain of values spanned by the measurements. This is obviously not true in general and specifically for the manipulator robots because of the presence of products of sinusoidal-like terms involving partial sums of the angular positions. Therefore, using a single polynomial to fit the entire data would lead to high level of residuals and hence to a low anomaly detection capability as discussed earlier.

On the other hand, since it can be argued that a large class of relationships (almost any, actually) can be represented by piece-wise polynomial approximations, a more reasonable attempt would be to assume that there is a partition of the dataset $\mathcal{D}$ of the form:

$$\left\{\mathcal{D}_\kappa\right\}_{\kappa=1}^{n_m} \quad \text{where} \quad \mathcal{D}_\kappa \subset \mathcal{D} \tag{14}$$

over each of which the restricted version of the above problem (12) becomes:

$$\mathcal{Y}(\mathcal{D}_\kappa) \approx [\Phi(\mathcal{D}_\kappa, d)]\, c^{(\kappa)} \quad ; \quad c^{(\kappa)} \in \mathbb{R}^{n_c} \tag{15}$$

leading to a different MP defined for each subset $\mathcal{D}_\kappa$ of the data.

Notice that given the high number n_c of degrees of freedom, solving the above system in a standard least squares sense is not an option because of the high risk of over-fitting. This is the reason why each of the linear systems of equations invoked in (15) should be solved in a parsimonious way, namely by attempting to maximize the number of zero components inside each of the vector of coefficients $c^{(\kappa)}$ involved in (15).

This can be done using well established solvers that are made freely available in numerical packages such that the Machine Learning dedicated `scikit-learn` [14] which provides the `lassoLarsCV` excellent solver among some others. In the forthcoming development, another solver, namely the `sclars`, that implements a modified least angle regression logic (see Chapter 10 of [1]) is used because of its scalability given the very large number of unknowns potentially involved (The scalability of the algorithm is investigated in [1] where it is shown that problems with up to 400,000 d.o.f might be solved in less than few minutes while standard `scikit-learn` solvers struggles when the number of d.o.f is greater than 35,000.). Nevertheless, in order to keep the presentation independent of this particular choice, it is assumed hereafter that such a sparse solver, denoted by $\mathcal{S}$, is used leading to the sparse solution's operation denoted as follows:

$$c^{(\kappa)} \leftarrow \mathcal{S}(\Phi_\kappa, \mathcal{Y}_\kappa) \tag{16}$$

where $\mathcal{S}$ designates the linear systems sparse solver while the arguments designate the features sub-matrix and the label sub-vector respectively.

In what follows, the following straightforward definition is considered in the statement of the results that underlines the rationale behind the algorithm:

Definition 1 (A perfect solver). *A sparse solver $\mathcal{S}$ is said to be perfect if for any polynomial features matrix $\Phi \in \mathbb{R}^{n_\ell \times n_c}$ such that $n_\ell > n_c$ and all $c \in \mathbb{R}^{n_c}$, the following equality:*

$$c = \mathcal{S}(\Phi, [\Phi]c) \tag{17}$$

holds true.

which simply means that when feeding the algorithm with data issued from an exact polynomial representation, then the algorithm perfectly finds its coefficients.

Notice that the discussion above does not tell how to compute the appropriate partition (14) as this is the aim of Section 3. But assuming that this is appropriately done, the process results in the derivation of a set of MPs $\{P_\kappa\}_{\kappa=1}^{n_m}$ that are defined by the above computed $c^{(\kappa)}$. Namely:

$$P_\kappa(x) = \sum_{j=1}^{n_c} c_j^{(\kappa)} \phi_j(x) \tag{18}$$

that can be used as described in the introduction in order to compute the error e defined by (5).

Remark 2. *It is important to attract the reader's attention to the fact that the subsets $\mathcal{D}_\kappa$, $\kappa = 1, \ldots, n_m$, are only intermediate entities that are not used once the algorithm ends. Only the set of computed polynomials $\{P_\kappa\}_{\kappa=1}^{n_m}$ are used to produce residuals. As a matter of fact, attempts were made to fit a multi-class classifier that provides the subset index $\kappa(x)$ as a function of the features vector x so that a true predictor can be defined by:*

$$\hat{y} := P_{\kappa(x)}(x) \tag{19}$$

but these attempts were not systematically successful. This can be, at least partially, explained by the high precision of the implicit relationships (see later). Indeed, if for such precision levels it was possible to classify the regions, this would have meant that the torques can be obtained as precise explicit functions of the kinematic variables and we know that this is not true because the torque values also depends on some additional pieces of information such as the set-point and/or the tracking errors on these variables. This is the reason why only the set of polynomials is used to provide the prediction error according to the implicit relationship (5)*.*

Let us end this section with some additional notation that is extensively used in the presentation of the algorithm: Given a matrix $M \in \mathbb{R}^{n_1 \times n_x}$, the notation M_i denote the i-th row of the matrix. For each subset of rows numbers $I \subset \{1, \ldots, n_1\}$, the notation M_I denotes the sub-matrix defined by the rows with indexes contained in I. Given $x \in \mathbb{R}^{n_x}$, the hypercube of size ρ, with center x, is denoted by $\mathbb{H}_\rho(x)$, namely:

$$\mathbb{H}_\rho(x) := \{\xi \mid |\xi_i - x_i| \leq \rho\} \tag{20}$$

Moreover, the set of rows contained in M_I that lie in $\mathbb{H}_\rho(x)$ is shortly denoted by $\mathbb{H}_\rho\,(x \mid (M, I))$. Now if $\Phi(\mathcal{D}, d)$ plays the role of M, each set of indexes I can be used to define a subset $\mathbb{H}_\rho\,(x \mid (\Phi, I))$ that can play the role of candidate to be a member of the partition $\{\mathcal{D}_\kappa\}$ discussed above. This is developed in the following section.

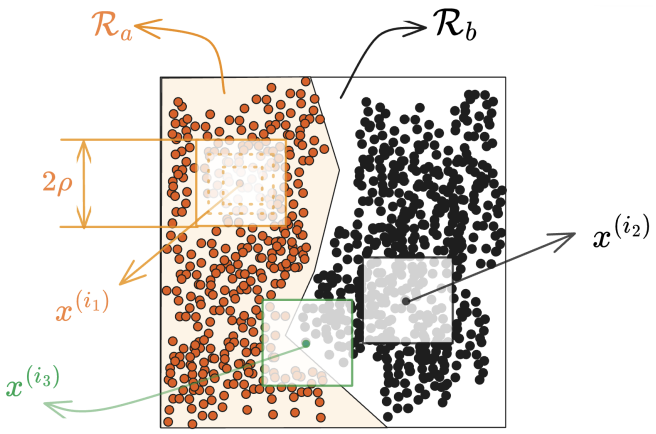


Figure 2: Illustration of a simple ideal case.

3. The proposed algorithm

In order to introduce the algorithm, let us consider, for the sake of illustration the case where $n_x = 2$ and assume that a relationship is defined by the following piece-wise polynomial (Fig.2):

$$y := \begin{cases} P_a(x) & \text{if } x \in \mathcal{R}_a \\ P_b(x) & \text{if } x \in \mathcal{R}_b \end{cases} \tag{21}$$

Assume also that one disposes of a dataset $\mathcal{D}$ containing the 2D features instances represented graphically by the cloud of points shown in Fig.2 and their corresponding y values.

The main idea is to acknowledge that under the assumption that the relationship between x and y is piece-wise polynomial, one can iteratively *identify* the involved polynomials by executing the following set of steps:

1. Randomly select a sample $x^{(i)}$ in the data set.
2. Tune the size ρ of the hypercube $\mathbb{H}_\rho(x^{(i)})$ (exemplified in Fig.2 by one of the white back-grounded squares) so that it contains a sufficient number of samples, say $N_{\min}$, of the dataset. This number ($N_{\min}$) depends on the degree of the polynomial to be identified since for a given n_x, this determines the number of unknowns.
3. Solve the associated sparse optimization problem (16) to compute, *parsimoniously*, the best polynomial, say P_i that fits the data in $\mathbb{H}_\rho(x^{(i)})$. By parsimoniously, it is meant that the solver should involve a penalty on the number of non-zero coefficients in order to avoid over-fitting in presence of high number of degrees of freedom. This naturally also enhances the robustness to measurement noise.
4. If the polynomial P_i produces a regression residual that is *sufficiently small* (lower than some targeted precision threshold `th`), then add the polynomial to the set of admissible polynomials to be considered in the final solution, namely the set previously denoted by $\{P_\kappa\}_\kappa$. For instance, considering our illustrative example, this step would be successful if the randomly sampled instance is similar to $x^{(i_1)}$ or $x^{(i_2)}$ shown in Fig.2 while the choice of $x^{(i_3)}$ might probably lead to failure.
5. In case the polynomial is admissible, evaluate it over the remaining data and remove all the instances at which it provides sufficiently precise prediction. Notice that the set of removed instances necessarily contains the instances that lie inside the hypercube $\mathbb{H}_\rho(x^{(i)})$ but it is generally much larger. For instance, considering the example of Fig.2, if the randomly sampled instance is $x^{(i_1)}$, then P_a would be identified and when evaluated on the remaining dataset, all the instances belonging to the region $\mathcal{R}_a$ would be removed as P_a is supposed to capture the relationship over all these instances.
6. If the condition stated in step 4. is not satisfied (the residual of the fit is too high) then discard the polynomial and go to step 1. for a new randomly generated sample $x^{(i)}$ among the remaining ones unless the maximum number of iterations is reached or the number of remaining samples is too small, in which case, terminate the algorithm and the return the set of admissible polynomials that have been gathered through the iterative process.

The previously explained steps are gathered in the following condensed algorithm:

Algorithm 1 Identification of the set of polynomials $\{P_\kappa\}_\kappa$

Input parameters: $\mathcal{D}$, $N_{\min}$, `th`, `max_iter`
Initialization: `i` $\leftarrow 0$, $\mathcal{D}_{\text{remaining}} \leftarrow \mathcal{D}$, $\mathcal{P} = \emptyset$
while (card($\mathcal{D}_{\text{remaining}}$) $> N_{\min}$) and (`i`$<$ `max_iter`) **do**
  $x^{(i)} \leftarrow$ random selection in $\mathcal{D}_{\text{remaining}}$. ▷ Step 1.
  $\rho \leftarrow$ such that card($\mathbb{H}_\rho(x^{(i)})$) $> N_{\min}$. ▷ Step 2.
  $P_i \leftarrow$ fit the data in $\mathbb{H}_\rho(x^{(i)})$ parsimoniously. ▷ Step 3.
  if (Residual($P_i \mid \mathbb{H}_\rho(x^{(i)})$) $<$ `th`) **then** ▷ Step 4.
    $\mathcal{P} \leftarrow \mathcal{P} \bigcup \{P_i\}$ ▷ Step 4.
    $\mathcal{D}_{\text{ok}} \leftarrow \{d \in D_{\text{remaining}} \mid \text{Residual}(P_i \mid d) \leq$ `th`$\}$
    $\mathcal{D}_{\text{remaining}} \leftarrow \mathcal{D}_{\text{remaining}} - \mathcal{D}_{\text{ok}}$ ▷ Step 5.
  end if
  $i \leftarrow i + 1$
end while
return a

Obviously, in the case of the illustrative example of Fig.2, the algorithm stops as soon as two samples such as $x^{(i_1)}$ and $x^{(i_2)}$ are selected. Notice however that for this to happen, two conditions are necessary:

1. The number of samples is sufficiently high so that the minimum number $N_{\min}$ of instances can be obtained within a sufficiently small hypercube that lies exclusively in a single region.
2. The number of needed polynomials is moderate so that the size of regions is sufficiently high to induce decently

high probability that the selected hypercubes $\mathbb{H}_\rho(x^{(i)})$ do not span over multiple regions.

The above discussion leads to the following result that provides the ideal foundation of the algorithm's success:

Proposition 1 (Ideal setting).
If the following conditions hold true:

1. *The targeted label y is defined by a piece-wise polynomial relationship of degree lower than d that involves a finite number of regions* $\{\mathcal{R}_\kappa\}_{\kappa=1}^{n_m}$.
2. *The degree of the model used is greater than d.*
3. *The sparse solver is perfect and the measurements are noise-free.*
4. *The number of iterations is not limited.*

then for a sufficiently high population of samples inside each of the regions $\mathcal{R}_\kappa$*, it is probabilistically certain that the algorithm, when executed using the degree d, stops after a finite number of iterations delivering a set of polynomials that includes all those involved in the relationship producing the data.*

Proof. Indeed, as the size of the population of samples increases inside each region $\mathcal{R}_\kappa$, **the size** $\rho > \mathbf{0}$ needed for any $\mathbb{H}_\rho(x)$, $x \in \mathcal{R}_\kappa$ to contain a number of samples n_ℓ that is greater than the number of features n_c (associated to the pair (n_x, d) given by (9)) **decreases**. This ultimately makes the probability of randomly selecting a center $x^{(i)}$ such that the hypercube $\mathbb{H}_\rho(x^{(i)})$ entirely lies in a single region $\mathcal{R}_\kappa$ rigorously > 0. This makes the probability of this events taking place equals to 1 in a process involving an unbounded number of random sampling steps (Assumption 4). As this is true for all regions and since the solver is assumed to be perfect and the measurements are noise-free (Assumptions 3), the truly involved polynomials enter the returned set with probability 1. Moreover, as soon as each one enters the set, the associated region $\mathcal{R}_\kappa$ is entirely removed (Assumptions 2, 3) meaning that the algorithm eventually stops given that the number of regions is limited by Assumption 1. □

Obviously, the framework of Proposition 1 sketches the ideal situation which enables to capture the main rationale behind the algorithm. More realistic formulations can be derived at the price of some technical definitions and ad-hoc assumptions that would make the main rational less easy to grasp. Given that the results proposed in the next sections are obtained using real-life experimental data, it might legitimately represent a decent alternative for the real-life validation of the proposed algorithm. As a matter of fact, such a validation would have been anyway the only ultimate proof of worthiness and an unavoidable step in the path towards an assessment of the relevance and efficiency of the proposed algorithm.

4. The dataset and algorithm's parameters

The working data used in this section in order to validate the proposed framework are collected on a `Staübli` TS2-80 4-links robot such as the one shown in Fig.3. Quite a rich set of scenarios is executed where the robot performs random point-to-point movements with randomly chosen points spanning almost the entire workspace, random velocities, random accelerations and decelerations. Using this data collection process, the final working data consists of 200 trajectories, each of which lasts 30 seconds in average. Measurements are acquired at 250Hz. This leads to a dataset containing 1,600,000 samples. Typical one-minute profiles[1] of the torque and angular position on axis 1 are shown in Fig.4.

Figure 3: The Stäubli 4-links robot manipulator providing the working data used in Section 4. Axis 3 is translational while all the remaining ones are rotational.

The algorithm is applied using $x = (q, \dot{q}, \ddot{q}) \in \mathbb{R}^{12}$ as features vector where $\dot{q}$ and $\ddot{q}$ are computed directly from the measured angular position thanks to its high precision. The label is considered to be the current-induced evaluation of the controlling torque T_s on each axis $s \in \{1, \ldots, 4\}$ as expressed by the torque's model as used by the control loop inside the robot. This leads to four residual generation problems that all share the same above mentioned 12-dimensional features vector x.

The available data has been partitioned into train dataset (10%, namely 20 experiments) and test dataset (the remaining 90%, namely 180 experiments). The feature matrix is first normalized by the 99% percentile of their absolute values on the training data and the same normalizing parameters are later applied to the test data.

The initial size ρ of the hypercube used in algorithm, as described in Section 3, is taken equal to 0.02 and when necessary an increase rate of 1.2 is applied until the hypercube contain $N_{\min} = 100$ instances. The fit over a hypercube is considered to be admissible if the following condition holds on the fitting error:

$$\mathcal{M}_q\left[e(\cdot)\right] := \left.\frac{\texttt{percentile}(|e(\cdot)|, q)}{\texttt{percentile}(|y(\cdot)|, 50)}\right|_{q=95} \leq \texttt{th} \qquad (22)$$

[1] The numerical values has been omitted for industrial privacy reasons.

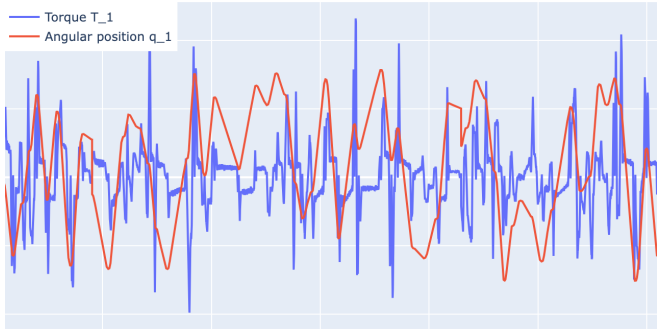


Figure 4: Typical experimental profiles during one minute showing torque's and angular position's evolution (axis 1).

where the following six possible values of the threshold:

$$\texttt{th} \in \{0.1, 0.15, 0.2, 0.25, 0.3, 0.4\} \tag{23}$$

have been found appropriate after few trials on training data as it leads to moderate number of models with quite small fitting residuals (see hereafter). Notice that too small values of `th` lead to too many failures in Step 4 of the algorithm (see Page 5) while too large values yield worthless model showing too large confidence intervals.

5. Results

Fig.5 shows the convergence results for the four fitting problems where the labels are T_s, $s = 1, \ldots, 4$ when the precision threshold is fixed to `th`=0.3 [see (22)]. Notice that since the iterations involve random selection of hypercube centers, convergence details varies among runs. The results shown in Fig.5 are given for the sake of illustration of typical behavior.

Notice that the plots show two different y-scales. The decreasing curves y-value, representing the **percentage** of the still unremoved data, are to be read on the right side y-scale while the increasing ones, representing the number of non zero coefficients already involved in the polynomial models, are to be read using the left y-scale. The x-axis abscissa refers to the number of polynomial models involved where the final value is precisely n_m that is reported on Table 1. This table also provides the total number of non zero coefficients and the associated computation time for each axis, each threshold `th`.

Table 2 shows the normalized percentiles of residuals as defined by (22) for $q \in \{50, 80, 90, 95, 99\}$ per axis and for all the values of the precision threshold `th` belonging to the set defined by (23). Notice that for $q = 50\%, 90\%$ and 99%, the residuals percentiles that would have been obtained should the trivial set of constant polynomials invoked in Remark 1 be used are given after the | symbol.

The relevance of the precision reported in Table 2 are examined via two means. In the first, comparison is done with the same

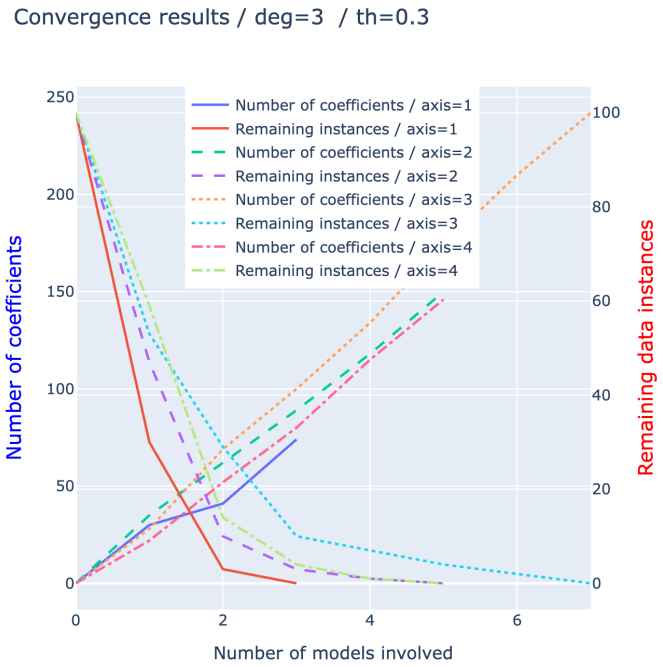


Figure 5: Convergence results for the four identification problems when the precision threshold is set to `th`=0.3 and a polynomial degree=3 is used in (22). Notice that the number of models increases as the iterations proceed as explained in Section 3.

th	#models (n_m)	#coeffs	cpu (sec)	axis
0.4	2	55	15	1
0.4	3	72	15	2
0.4	4	112	22	3
0.4	3	95	19	4
0.3	3	74	16	1
0.3	5	150	20	2
0.3	7	242	27	3
0.3	5	146	22	4
0.25	4	116	20	1
0.25	6	159	24	2
0.25	5	168	22	3
0.25	9	271	33	4
0.2	6	135	28	1
0.2	9	262	38	2
0.2	13	424	33	3
0.2	13	380	40	4
0.15	13	387	39	1
0.15	10	276	40	2
0.15	16	534	41	3
0.15	20	556	53	4
0.1	22	585	63	1
0.1	29	871	78	2
0.1	19	621	60	3
0.1	32	869	86	4

Table 1: Characteristics (number of polynomials n_m, total number `#coeffs` of non zero coefficients) and computation times of the identified models for precision thresholds `th`$\in \{0.1, 0.15, 0.2, 0.25, 0.3, 0.4\}$. (`cpu` times are obtained on a `MacBook Pro, Apple M3 Pro, 18 Go RAM` running on Sequoia 15.3.1).

Axis	th	50%	80%	90%	95%	99%
1	0.40	0.12| 13.41	0.24	0.29| 24.15	0.35	0.47| 26.57
1	0.30	0.09| 6.71	0.19	0.26| 12.1	0.32	0.46| 13.2
1	0.25	0.08| 4.47	0.17	0.23| 8.05	0.27	0.38| 8.85
1	0.20	0.06| 2.68	0.13	0.19| 4.83	0.24	0.37| 5.31
1	0.15	0.03| 1.12	0.07	0.11| 2.01	0.15	0.24| 2.21
1	0.10	0.02| 0.64	0.04	0.06| 1.15	0.09	0.16| 1.26
2	0.40	0.13| 6.80	0.25	0.33| 12.24	0.40	0.58| 13.46
2	0.30	0.08| 3.34	0.17	0.24| 6.12	0.30	0.46| 6.73
2	0.25	0.07| 2.72	0.17	0.24| 4.90	0.30	0.47| 5.38
2	0.20	0.05| 1.70	0.12	0.17| 3.05	0.22	0.36| 3.37
2	0.15	0.04| 1.51	0.10	0.14| 2.71	0.18	0.31| 2.99
2	0.10	0.02| 0.49	0.04	0.06| 0.87	0.09	0.17| 0.96
3	0.40	0.09| 1.64	0.19	0.26| 2.96	0.32	0.44| 3.25
3	0.30	0.04| 0.82	0.08	0.12| 1.48	0.15	0.22| 1.26
3	0.25	0.06| 1.23	0.11	0.15| 2.22	0.19	0.27| 2.44
3	0.20	0.03| 0.41	0.08	0.11| 0.74	0.13	0.18| 0.81
3	0.15	0.02| 0.33	0.06	0.08| 0.59	0.10	0.14| 0.65
3	0.10	0.02| 0.27	0.05	0.07| 0.49	0.08	0.12| 0.54
4	0.40	0.10| 2.09	0.20	0.26| 3.76	0.31	0.41| 4.14
4	0.30	0.08| 1.04	0.15	0.20| 1.88	0.24	0.32| 2.07
4	0.25	0.05| 0.52	0.11	0.15| 0.94	0.19	0.28| 1.04
4	0.20	0.04| 0.35	0.09	0.12| 0.63	0.14	0.20| 0.69
4	0.15	0.03| 0.22	0.06	0.09| 0.40	0.11	0.16| 0.44
4	0.10	0.02| 0.14	0.04	0.06| 0.24	0.08	0.12| 0.27

Table 2: Residuals $\mathcal{M}_q$ defined by (22) computed on the unseen (test) experimental data as achieved by the proposed piece-wise polynomial implicit model which is fitted on 10% of the available data. For $q \in \{50, 90, 99\}$ the q-percentile of the error, expressed by (6), that would be achieved by the trivial solution under uniform data distribution is given after the | symbol.

precision measures that is obtained using NN models (Section 5.1). Then the resulting residual generators are used to detect representative slight anomalies that are introduced in the data (Section 5.2).

5.1. Precision comparison with DNN models

In order to assess the precision level of the resulting residuals generators, a comparison is proposed with the residuals that might be obtained using a deep learning model involving gated recurrent cells that is briefly described here for the sake of completeness. To our knowledge, such models are the most accurate data driven models for learning the whole dynamics of a robot that can be found in the literature. Gated recurrent cells such as GRU and LSTM cells have been introduced to face traditional vanishing and exploding gradient problems when learning deep networks through long time-series. Such cells consists in combinations of neural layers, usually called "gates", designed to focus on relevant long term and short term dependencies and ignore useless ones.

Whereas LSTM networks have a complex internal structure, GRU are known for their lighter architecture which reduces mathematical complexity and computational cost while leading to similar accuracy performances. Since we have performed tests with both networks types that confirmed this result, we use in this paper a model based on GRU cells. The mathematical

Model name	seq_len	#hidden_dim	#coeffs	cpu (hours)
DNN_128_32	128	32	31224	3h52
DNN_128_128	128	128	235128	5h43
DNN_1_32	1	32	31224	3h27
DNN_1_128	1	128	235128	3h40

Table 3: Characteristics and computation times of the built DNN models for different hidden dimensions and seq_len. (cpu times are obtained on a Dell PC, with Intel Core i7 vPRO processor, with 32 Go RAM and NVIDIA RTX A2000 GPU, running on windows 10.

expression of GRU is recalled in the equations below

$$z_t = \sigma\left(W_z x_t + U_z h_{t-1} + b_z\right) \tag{24}$$

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right) \tag{25}$$

$$\widetilde{h}_t = \tanh\left(W_{\widetilde{h}} x_t + U_{\widetilde{h}}\left(h_{t-1} \odot r_t\right) + b_{\widetilde{h}}\right) \tag{26}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \widetilde{h}_t \tag{27}$$

where $\odot$ is the element-wise product, tanh and σ are respectively the hyperbolic tangent and sigmoid activation functions and $W_z, U_z, W_r, U_r, W_{\widetilde{h}}, U_{\widetilde{h}}, b_z, b_r, b_{\widetilde{h}}$ are matrices and vectors of weights and biases and $z_t, r_t, \widetilde{h}_t$ are the gates states that enable to compute the hidden state h_t. The deep learning model first uses two layers of GRU cells to compute a hidden state which is then fed to a dense layer which predicts a 4-th dimensional vector representing the four torques T_s, $s = 1, \ldots, 4$.

The models are implemented in PyTorch [13] and trained to minimize the negative log-likelihood of the normal distribution. Optimization is performed using Adam with an exponentially decreasing learning rate starting from 0.0001. A traditional 80-20% split is made for train and test sets. Additionally, a small validation set is created by taking a few samples from the test set, in order to implement an early stopping mechanism that allows us to limit overfitting. The networks are trained for a maximum of 200 epochs, with batches of size 64.

In order to compare the proposed piece-wise polynomial implicit models to DNN models, we tried a few different configurations by tuning some hyper-parameters. Such as the hidden_dim parameter that corresponds to the hidden state's dimension of the GRU cells (which strongly impacts the model complexity) and the seq_len parameter that corresponds to the length of the time series' that are used as input sequences of the DNN (setting seq_len to 1 is equivalent to use the same input as defined in (7) while setting it to higher values corresponds to add past features in the input).

Table 3 summarizes some of the tested DNN models and gives for each the number of trainable parameters and training time. These values are to be compared with the number of active coefficients and the computation times for the proposed structure as shown in Table 1.

Table 4 gives for each DNN models the obtained performance

Axis	Model name	50%	80%	90%	95%	99%
1	DNN_1_32	0.15	0.28	0.36	0.44	0.65
1	DNN_1_128	0.10	0.21	0.28	0.35	0.51
1	DNN_128_32	0.07	0.14	0.19	0.24	0.36
1	DNN_128_128	0.06	0.13	0.17	0.22	0.34
1	Proposed/th=0.20	**0.06**	**0.13**	**0.19**	**0.24**	**0.37**
1	Proposed/th=0.15	**0.03**	**0.07**	**0.11**	**0.15**	**0.24**
1	Proposed/th=0.1	**0.02**	**0.04**	**0.06**	**0.09**	**0.16**
2	DNN_1_32	0.23	0.40	0.53	0.70	1.17
2	DNN_1_128	0.12	0.26	0.37	0.47	0.71
2	DNN_128_32	0.08	0.16	0.23	0.31	0.54
2	DNN_128_128	0.05	0.13	0.17	0.22	0.35
2	Proposed/th=0.20	**0.05**	**0.12**	**0.17**	**0.22**	**0.36**
2	Proposed/th=0.15	**0.04**	**0.10**	**0.14**	**0.18**	**0.31**
2	Proposed/th=0.1	**0.02**	**0.04**	**0.06**	**0.09**	**0.17**
3	DNN_1_32	0.24	0.40	0.48	0.54	0.69
3	DNN_1_128	0.15	0.31	0.40	0.48	0.59
3	DNN_128_32	0.23	0.38	0.47	0.54	0.69
3	DNN_128_128	0.05	0.12	0.16	0.20	0.42
3	Proposed/th=0.20	**0.03**	**0.08**	**0.11**	**0.13**	**0.18**
3	Proposed/th=0.15	**0.02**	**0.06**	**0.08**	**0.10**	**0.14**
3	Proposed/th=0.1	**0.02**	**0.05**	**0.07**	**0.08**	**0.12**
4	DNN_1_32	0.34	0.59	0.71	0.79	0.94
4	DNN_1_128	0.12	0.26	0.40	0.53	0.76
4	DNN_128_32	0.12	0.25	0.35	0.46	0.71
4	DNN_128_128	0.10	0.20	0.28	0.37	0.62
4	Proposed/th=0.20	**0.04**	**0.09**	**0.12**	**0.14**	**0.20**
4	Proposed/th=0.15	**0.03**	**0.06**	**0.09**	**0.11**	**0.16**
4	Proposed/th=0.1	**0.02**	**0.04**	**0.06**	**0.08**	**0.12**

Table 4: Achieved residuals $\mathcal{M}_q$ defined by (22), for $q \in \{50\%, 80\%, \ldots, 99\%\}$ by the DNN architecture and the proposed piece wise polynomial model on the test experiments.

metrics expressed in terms of the $\mathcal{M}_q$ measure as defined by (5) together with the corresponding performance of the piece-wise polynomial structure, already shown in Table 2, that are incorporated for the sake of easiness of comparison. For the sake of completeness, Table 5 provides the values of more traditional metrics such as the mean absolute error (MAE) or the mean squared error (MSE) for the previously presented polynomial models and the best DNN model that we have identified.

Axis	Model	MAE	MSE	n_m
1	DNN_128_128	0.036	2.1×10^{-3}	–
1	Proposed/th=0.40	0.063	6.5×10^{-3}	2
1	Proposed/th=0.30	0.053	4.9×10^{-3}	3
1	Proposed/th=0.25	0.047	3.7×10^{-3}	4
1	Proposed/th=0.20	0.036	2.6×10^{-3}	6
1	Proposed/th=0.15	0.021	2.0×10^{-3}	13
1	Proposed/th=0.1	0.012	4.0×10^{-4}	22
2	DNN_128_128	0.021	9.0×10^{-4}	–
2	Proposed/th=0.40	0.039	2.6×10^{-3}	3
2	Proposed/th=0.30	0.027	1.4×10^{-3}	5
2	Proposed/th=0.25	0.026	1.3×10^{-3}	6
2	Proposed/th=0.20	0.019	7.4×10^{-4}	9
2	Proposed/th=0.15	0.015	4.0×10^{-4}	10
2	Proposed/th=0.1	0.007	4.0×10^{-4}	29
3	DNN_128_128	0.008	1.4×10^{-4}	–
3	Proposed/th=0.40	0.012	2.4×10^{-4}	4
3	Proposed/th=0.30	0.005	5.1×10^{-5}	7
3	Proposed/th=0.25	0.007	8.8×10^{-5}	5
3	Proposed/th=0.20	0.004	3.8×10^{-5}	13
3	Proposed/th=0.15	0.003	2.2×10^{-5}	16
3	Proposed/th=0.1	0.002	1.6×10^{-5}	19
4	DNN_128_128	0.005	5.7×10^{-5}	–
4	Proposed/th=0.40	0.004	3.9×10^{-5}	3
4	Proposed/th=0.30	0.004	2.3×10^{-5}	5
4	Proposed/th=0.25	0.003	1.4×10^{-5}	9
4	Proposed/th=0.20	0.002	7.9×10^{-6}	13
4	Proposed/th=0.15	0.002	4.5×10^{-6}	20
4	Proposed/th=0.1	0.001	2.3×10^{-6}	32

Table 5: Comparison of polynomial models and deep learning models with conventional metrics.

Discussion

The results clearly show that the precision achieved by the proposed structure and algorithm are quite *competitive* compared to the DNN alternative while requiring computation times that are drastically shorter. The parsimonious number of *active* (non zero) coefficients shown in Table 1, compared to the huge number of coefficients for DNN as shown in Table 3 should inspire higher confidence in terms of extrapolation ability. Table 5 shows that it is possible to achieve the precision of the best DNN model with a piece-wise polynomial implicit models involving $n_m = 6$ sub-models for axis 1, $n_m = 9$ sub-models for axis 2, $n_m = 5$ sub-models for axis 3 and $n_m = 3$ sub-models for axis 4.

5.2. Example of use in anomaly detection

In this section, an example of use is shown where the models developed are leveraged in order to perform anomaly detection and localization in the 4-axis manipulator robot. First of all, the three anomalies that are introduced for the sake of illustration are described before the behavior of the residuals generated by the implicit piece-wise polynomial-based models are examined.

5.3. The Introduced defaults

Recall that each of the models developed earlier provides a residual generator for the axis for which it has been identified. This residual takes the following form at each instant t:

$$\varphi(y(t), x(t)) \tag{28}$$

where y stands for the torque applied to the specific axis while $x \in \mathbb{R}^{12}$ is the feature vector that has been previously described.

The principle of the validation that is provided in the present section is to focus on the set of anomalies/defaults that translate into variations, denoted by e_y, of the applied torque y. Therefore, by examining the associated residual:

$$\varphi(y(t) + e_y(t), x(t)) \tag{29}$$

one can check whether it is possible to detect these anomalies/defaults using the identified piece-wise models. Moreover, one can also examine whether a default on one axis triggers significant increase of the residual, only on the specific axis on which the default is introduced and not on the others. If such is the case, then not only can the fault be detected but it would also be localized among the four axis.

Three different anomalies are introduced hereafter, namely:

– A relative error on the torque:

$$e_y = \lambda_r \times y \tag{30}$$

– A friction induced default leading to error e_y expressed by:

$$e_y = \lambda_f \times \dot{q}_i \tag{31}$$

where i is the axis number.

– A constant bias on the torque representing current sensor default:

$$e_y = \lambda_b \tag{32}$$

These errors have been introduced on the test dataset at three different intervals as shown in Fig.6. Then Figures 7-9 show zooms on these three intervals in order to show the rather small and barely perceptible differences between y and $y + e_y$.

r	th	axis	η	Figure
2	0.1	1	0.016	Fig. 10-(a)
3	0.1	1	0.015	Fig. 10-(a)
2	0.3	1	0.071	Fig. 10-(b)
3	0.3	1	0.066	Fig. 10-(b)
2	0.1	2	0.010	Fig. 11-(a)
3	0.1	2	0.009	Fig. 11-(a)
2	0.3	2	0.035	Fig. 11-(b)
3	0.3	2	0.033	Fig. 11-(b)

Table 6: Values of the alarm firing threshold η computed according to (2) for different values of the moment r, for axis 1 and 2 and different values of th$\in \{0.1, 0.3\}$ used to fit the model. The corresponding figures relevant to the each threshold are also mentioned int he last column.

5.4. Anomalies detection results

The detection results when the three defaults are introduced on Axis 1 are shown in Fig.10 for the two identified models corresponding to th=0.1 and th=0.3 respectively. A moving averaging window of length 10000 is used in order to more clearly see the increase of the residual statistics. Notice for instance that a 10% relative error leads to 250% increase on the level of implicit residual value. Moreover, it can be observed that when the error affects Axis 1, the residual of the other axis are unchanged.

The same results are shown in Fig.11 when the same defaults are introduced on Axis 2. Here again, a drastic amplification is operated on the residual of Axis 2 while the remaining axis's residual remain unchanged.

Table 6 gives the alarm firing threshold η defined by (2) for the residual generator associated to axis 1 and 2 and th$\in \{0.1, 0.3\}$ clearly showing that the residual bursts shown in Figures 10 and 11 would have been appropriately detected with with almost no harmful level of false alarm.

6. Conclusion and future works

In this paper, an implicit piece-wise multi-variate polynomial structure and an associated fitting algorithm are proposed to capture the normality of non-linear dynamical systems. The framework is applied to the challenging case of a multi-link manipulator robots and the results, obtained using real-life experimental collected data, in terms of precision are compared to the state-of-the-art DNN showing competitive performance both in terms of precision, computation time and sparsity. The use of the so defined normality characterization in achieving the task of anomaly detection also shows promising results.

Despite these encouraging results, the algorithm in its current setting shows some issues that need to be investigated in future works. Indeed, the algorithm is not fully automated in the sense that few trials are needed to properly determine the interval of appropriate values of the threshold th and the degree of the polynomial to be used. Moreover, when several candidate

solutions involving different pairs of (threshold, degree) are eligible, it is not clear what is the *best one* to be used. This is because lower `th` induces higher number of models n_m leading to antagonistic effects on the detection rate.

Undergoing investigation concerns also the collection of real-life failure induced measurements in order to assess the ability of the resulting residual generators to address the problem of detecting real-life various and unexpected anomalies manipulator robots. On the other hand, the application of the methodology sketched in the current contribution to other application domains is under investigation.

References

[1] Alamir, M., 2025. Nonlinear control of uncertain systems. Conventional and learning-based alternatives with Python. Springer-Nature (under press).

[2] Deisenroth, M.P., Fox, D., Rasmussen, C.E., 2015. Gaussian processes for data-efficient learning in robotics and control. IEEE Transactions on Pattern Analysis and Machine Intelligence 37, 408–423. URL: http://dx.doi.org/10.1109/TPAMI.2013.218, doi:10.1109/tpami.2013.218.

[3] Ferreira, J., Crisóstomo, M., Coimbra, A., 2007. Simulation control of a biped robot with support vector regression, pp. 1 – 6. doi:10.1109/WISP.2007.4447538.

[4] Gillespie, M.T., Best, C.M., Townsend, E.C., Wingate, D., Killpack, M.D., 2018. Learning nonlinear dynamic models of soft robots for model predictive control with neural networks, in: 2018 IEEE International Conference on Soft Robotics (RoboSoft), pp. 39–45. doi:10.1109/ROBOSOFT.2018.8404894.

[5] Hitzler, K., Meier, F., Schaal, S., Asfour, T., 2019. Learning and adaptation of inverse dynamics models: A comparison, in: 2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids), pp. 491–498. doi:10.1109/Humanoids43949.2019.9035048.

[6] Liu, N., Li, L., Hao, B., Yang, L., Hu, T., Xue, T., Wang, S., 2019. Modeling and simulation of robot inverse dynamics using lstm-based deep learning algorithm for smart cities and factories. IEEE Access 7, 173989–173998. URL: https://api.semanticscholar.org/CorpusID:209322142.

[7] Liu, N., Li, L., Hao, B., Yang, L., Hu, T., Xue, T., Wang, S., Shao, X., 2020. Semiparametric deep learning manipulator inverse dynamics modeling method for smart city and industrial applications. Complex. 2020, 9053715:1–9053715:11. URL: https://api.semanticscholar.org/CorpusID:220523971.

[8] Lutter, M., Ritter, C., Peters, J., 2019. Deep lagrangian networks: Using physics as model prior for deep learning. CoRR abs/1907.04490. URL: http://arxiv.org/abs/1907.04490, arXiv:1907.04490.

[9] Lutter, M., Silberbauer, J., Watson, J., Peters, J., 2021. A differentiable newton-euler algorithm for real-world robotics. CoRR abs/2110.12422. URL: https://arxiv.org/abs/2110.12422, arXiv:2110.12422.

[10] Mukhopadhyay, R., Chaki, R., Sutradhar, A., Chattopadhyay, P., 2019. Model learning for robotic manipulators using recurrent neural networks, in: TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), pp. 2251–2256. doi:10.1109/TENCON.2019.8929622.

[11] Nguyen-Tuong, D., Scholkopf, B., Peters, J., 2009. Sparse online model learning for robot control with support vector regression. 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems , 3121–3126URL: https://api.semanticscholar.org/CorpusID:1751266.

[12] Nguyen-Tuong, D., Seeger, M., Peters, J., Koller, D., Schuurmans, D., Bengio, Y., Bottou, L., 2008. Local gaussian process regression for real time online model learning and control. Advances in Neural Information Processing Systems 21: Proceedings of the 2008 Conference, 1193-1200 (2009) .

[13] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library. URL: https://arxiv.org/abs/1912.01703, arXiv:1912.01703.

[14] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.

[15] Prokhorov, I.V., Statulevicius, V., 2000. Limit theorems of probability theory. volume 6. Springer Science & Business Media.

[16] Reuss, M., van Duijkeren, N., Krug, R., Becker, P., Shaj, V., Neumann, G., 2022. End-to-end learning of hybrid inverse dynamics models for precise and compliant impedance control. doi:10.15607/RSS.2022.XVIII.066.

[17] Rueckert, E., Nakatenus, M., Tosatto, S., Peters, J., 2017. Learning inverse dynamics models in o(n) time with lstm networks. doi:10.1109/HUMANOIDS.2017.8246965.

[18] Valencia-Vidal, B., Ros, E., Abadia, I., Luque, N., 2023. Bidirectional recurrent learning of inverse dynamic models for robots with elastic joints: a real-time real-world implementation. Frontiers in Neurorobotics 17. doi:10.3389/fnbot.2023.1166911.

[19] Wang, S., Shao, X., Yang, L., Liu, N., 2020. Deep learning aided dynamic parameter identification of 6-dof robot manipulators. IEEE Access PP, 1–1. doi:10.1109/ACCESS.2020.3012196.

[20] Yilmaz, N., Wu, J.Y., Kazanzides, P., Tumerdem, U., 2020. Neural network based inverse dynamics identification and external force estimation on the da vinci research kit. doi:10.1109/ICRA40945.2020.9197445.

[21] Çallar, T.C., Böttger, S., 2022. Hybrid learning of time-series inverse dynamics models for locally isotropic robot motion. IEEE Robotics and Automation Letters doi:10.48550/arXiv.2211.12921.

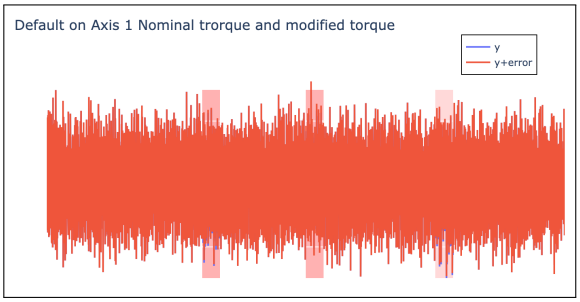


Figure 6: The three intervals (marked by the three red rectangular regions) where the defaults are introduced. Zooms over each interval are provided in figures 7, 8 and 9.

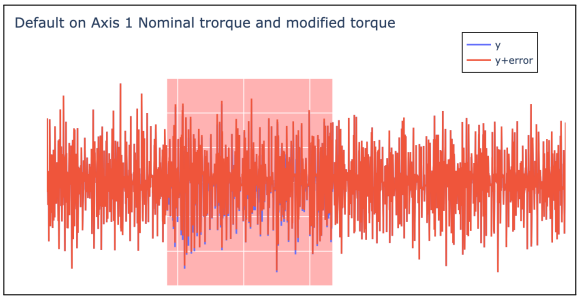


Figure 7: Zoom on the interval where the first default is introduced (relative error) Showing y and $y + e_y$. This corresponds to the case where $\lambda_r = 10\%$ in (30).

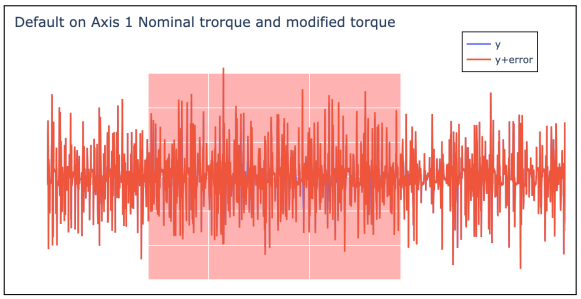


Figure 8: Zoom on the interval where the second default is introduced (friction induced error) Showing y and $y + e_y$. This corresponds to the case where $\lambda_f = 0.1$ in (31).

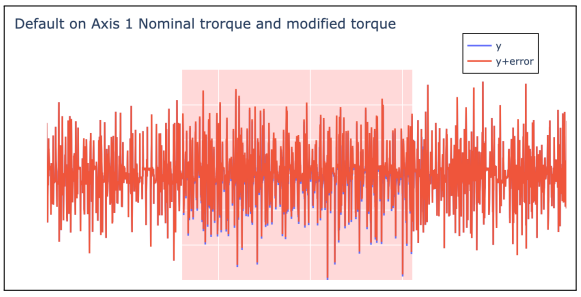
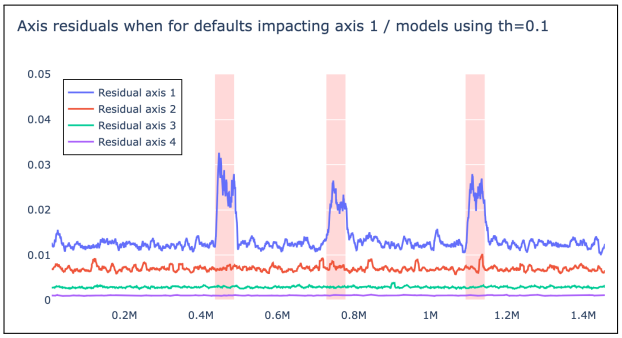


Figure 9: Zoom on the interval where the third default is introduced (constant bias) Showing y and $y + e_y$. This corresponds to the case where $\lambda_b = 0.12$ in (32).

(a)



(b)

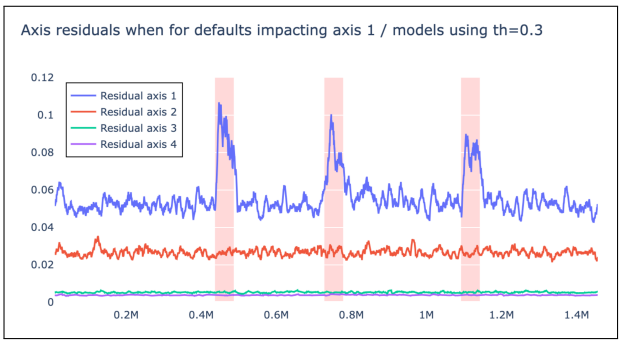
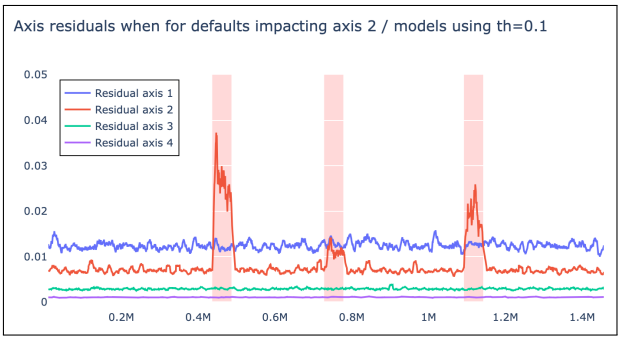


Figure 10: Detection results when the defaults are introduced on Axis 1. (a) Model with `th`=0.1 ($n_m = 22$ for Axis 1, see Table 5) and (b) Model with `th`=0.3 ($n_m = 3$ for Axis1, see Table 5). Notice how the residual is increased only for the axis where the defaults are introduced. Notice that the alarm threshold η computed from (2) for $r = 2$ is equal to 0.16 for (a) and 0.07 for (b).

(a)



(b)

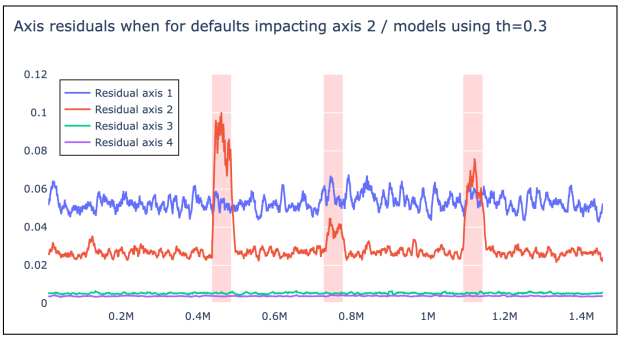


Figure 11: Detection results when the defaults are introduced on Axis 2. (a) Model with `th`=0.1 (n_m = 29 for Axis 2, see Table 5) and (b) Model with `th`=0.3 (n_m = 5 for Axis 2, see Table 5). Notice how the residual is increased only for the axis where the defaults are introduced. Notice that the alarm threshold η computed from (2) for $r = 2$ is equal to 0.01 for (a) and 0.035 for (b).