Leveraging Generative and Probabilistic Models for Diagnostics of Cyber-Physical Systems

Alvaro Piedrafita[1], Leonardo Barbini[2],

[1,2] *TNO-ESI, Eindhoven, The Netherlands*
alvaro.piedrafitapostigo@tno.nl
leonardo.barbini@tno.nl

ABSTRACT

A critical task for system operators is the precise identification of the root causes underlying an error situation. This identification is fundamental in deciding optimal maintenance actions, such as replacing a component versus calibrating it. However, the actual causes of an error are often neither measured nor unique. The measured quantities are the result of complex interactions between different error causes and system variables. Root cause identification in this context becomes a matter of inferring hidden causes from their measurable effects. This challenge is notably pronounced in cyber-physical systems comprising control loops. Control mechanisms, integral to maintaining system performance, introduce a layer of complexity in diagnostics and ultimately complicate the isolation of the underlying causes of errors. To address this challenge, we introduce a two-step approach to derive the hidden causes as a statistical inference task. First, we develop a generative model leveraging existing control software and expert-based insights into the mechanisms of errors, i.e., a simulator of synthetic data given some hidden error causes. Then, we transform the generative model into a probabilistic program on which statistical inference can be executed within a probabilistic programming language framework. This inference effectively estimates the hidden causes given some measured data from the system. Being intrinsically a statistical approach, these inferences come with a confidence interval. We applied this methodology to an industrial printer's sheet transport belt, operating in a closed-loop configuration. Our approach successfully discerned the contributions of three distinct hidden causes to the belt's deviation from its intended position. This paper highlights the efficacy of generative modeling followed by a probabilistic programming approach in unraveling complex interactions within cyber-physical systems for optimal maintenance.

1. INTRODUCTION

In order to match the increasing market demands on overall equipment effectiveness, industrial manufacturers of cyber-physical systems need efficient methods to diagnose system malfunctions. In practice, finding the root cause of such malfunctions is challenging for several reasons, ranging from technological to human and organizational ones.

On the technological side, the high demands on performance lead to increasingly complex systems with many intertwined control mechanisms that obscure the path from a root cause to its measurable effects (Borth & Barbini, 2019). The lack of direct observability for each cause of malfunctions forces the diagnostic to infer the many root causes from their effects on the few measured observables. Moreover, these observables are often not measured for diagnostic purposes but rather for control and performance ones, i.e. are indirect.

On the human and organizational side, the knowledge needed to solve difficult diagnostic cases is within the design and engineering departments, while the responsibility of offering diagnostic support lies on the service department (van Gerwen, Barbini, & Nägele, 2022). The transfer of the necessary knowledge is thus a difficult process that relies on expensive escalation-based approaches, i.e. design and engineering departments are called in by service to support the diagnostic reasoning. Finally, the struggle to timely train the service personnel capable of executing the needed diagnostic reasoning is a growing concern in the face of relentlessly increasing system complexity.

To tackle the points above we propose a method that focuses on two pillars. First, capture in models the knowledge of the system behavior, e.g. control loops, together with its failures. This should be done iteratively within the design and engineering departments, by incrementally incorporating knowledge on failures occurring in the field. Second, support the diagnostic reasoning process by performing statistical inference on the above models together with data from an error situation in the field, inferring the hidden causes of errors in a Bayesian way. This is the contribution of this methodology to the service department.

The rationale behind the proposed approach is that humans have the knowledge and the inclination to reason *forward*, i.e. in a simulation-like manner from the causes of a failure towards the resulting effects. Many such simulation models are readily available in industrial companies. Conversely, it is more challenging for humans to perform *inverse* diagnostic reasoning from the effects towards the causes: they need computational support to do so. In this paper we leverage the available system expertise and modeling capabilities of humans, with statistical inference tools to achieve diagnostic reasoning support.

The remainder of the paper is organized as follows: below we give an overview of the relevant literature. In Section 3 we introduce the details of our approach. In Section 4 we first apply our methodology to synthetic data and then to real data from an industrial system, finally we conclude our paper and give directions for future research in Section 5.

2. Literature review

The proposed method has its foundations in model-based diagnostics (De Kleer & Kurien, 2003) and specifically in its probabilistic implementation with Bayesian networks (Lucas, 2001; Srinivas, 1995). In this context, Bayesian networks, a type of probabilistic graphical model, are used to infer the likelihood of causes based on observed data via Bayes' theorem. The quantity of interest for the diagnosis is the posterior probability of cause C given observations O, computed as $P(C|O) = P(O|C) \cdot P(C)/P(O)$.

The present paper extends the previous work in two directions. First, we model and reason with continuous random variables, rather than discrete. This is fundamental when tackling performance issues, i.e. scenarios where the system's components are not described by a neat dichotomy of states, such as *normal* or *abnormal*, but rather sit in a continuous spectrum of states. Second, we model and reason on dynamic processes rather than on static ones. This is needed when diagnosing systems with feedback control loops and when the cause of failures shows a time-dependent behavior. In the literature, such systems are often modeled with dynamic Bayesian networks (Bartram & Mahadevan, 2015), but this is cumbersome and very quickly results in very large models, so we propose a different approach.

In this paper, we perform statistical inference on dynamic models with continuous random variables by using a probabilistic programming paradigm (van de Meent, Paige, Yang, & Wood, 2018). The proposed probabilistic programming approach can be seen as a generalization of Bayesian filtering and smoothing methods (Särkkä & Svensson, 2023) such as Kalman filters and particle filters. Several methods have been introduced in the probabilistic programming literature to perform such statistical inference, sampling-based methods like Markov chain Monte Carlo (van de Meent et al., 2018), gradient based methods (Kucukelbir, Tran, Ranganath, Gelman, & Blei, 2017) and analytic methods like message passing (Cox, van de Laar, & de Vries, 2019), or combinations thereof (Cox et al., 2019). In this paper we rely on Markov chain Monte Carlo using the Python library Numpyro (Phan, Pradhan, & Jankowiak, 2019). The proposed methodology makes use of simulation models to generate synthetic data for validation and fine-tuning of the inference models. The use of synthetic data has been explored before in other fields, see (Tremblay et al., 2018) on the use of synthetic data in deep learning, and (Cranmer, Brehmer, & Louppe, 2020) for a discussion on the use of simulation for inference.

3. Methodology

Our methodology is schematically represented in Figure 1. It uses two models, *simulation* and *inference*, and develops in three phases, *creation, validation* and *usage* phases. These are represented with different colors in the figure; orange, blue and green, respectively.

In the *creation* phase, we first compile a *simulation* model using knowledge of the system, thus re-using already available control models, and augmenting these with (conjectured) models of failure mechanisms. The latter heavily relies on expert knowledge based on historical failures. The *simulation model* outputs synthetic data given a single or a combination of failure mechanisms. The *simulation model* is then transformed into an *inference model*. This transformation is not computational, i.e. it requires additional modeling. For example, some aspects of the simulation might be deemed irrelevant or negligible and dropped from the inference. Further, one could decide to decompose a single *simulation model* into multiple *inference models*. We will return to this in Section 4.2.

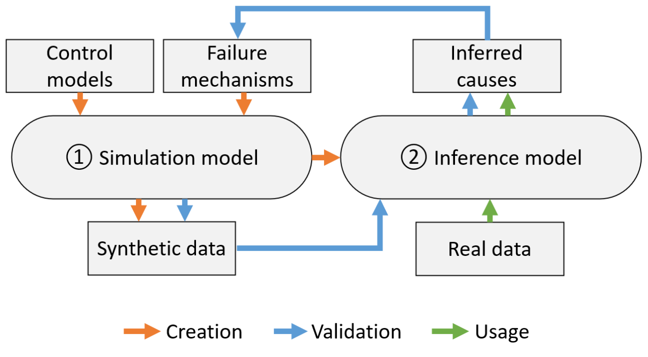

Figure 1. Schematic representation of the proposed methodology

In the *validation* phase, the goal is to verify that the constructed inference model indeed provides an inverse to the data generation process. We do so by testing whether the model can correctly infer the hidden causes of failures in synthetic data generated in simulation models. We repeat this for different types and combinations of failures.

Finally, in the *usage* phase, we use the *inference model* on the real data coming from a system in the field to infer the hidden causes of failures. The optimal service action, e.g. part replacement versus cleaning, is then decided based on these inferred causes.

4. APPLICATION

We apply the proposed methodology to a subsystem of a Canon Production Printing (CPP) industrial printer. This subsystem contains a conveyor belt that rests horizontally on two cylinders. The cylinders rotate at a variable speed and transmit this movement to the belt. For this subsystem, it is required that the belt is at the center of both cylinders, perpendicular to the directions of its movement. To fulfill this requirement, one of the cylinders can be tilted by raising or lowering it. The mechanism responsible for this tilting is driven by a motor. In the remainder of this paper, we will refer to it as the Z-position motor. This tilting causes the belt to slide up or down the cylinder each revolution by an amount proportional to the Z-motor position. Every few revolutions the position of the belt is measured and a correction is computed by a Proportional Integral (PI) controller, resulting in an adjustment of the Z-motor position. This steering action is necessary to counter the various causes that make the belt drift away from its intended position.

Our goal here is to discern the unknown causes of this drift and to infer their strength, given the available data on the belt and motor positions over time. This is crucial from a maintenance perspective, to define the best service action in those cases in which, despite the control mechanism, the belt goes out of its intended position. Following our methodology, we first make a model relating the known, i.e. measured, and the unknown variables of this system. Then we conjecture the functional form of the unknown variables to create a complete simulation model. In the next Section, we use the equations of the PI-controller for the former and expert knowledge for the latter.

4.1. Simulation model

Every step of the PI-controller begins with a measurement of the belt position. This belt position must be a function of the previous belt position, the previous motor correction, and the drift incurred between the current measurement and the previous one. Based on the current positions of both the belt and Z-motor, the position of the latter is updated by a PI controller with the goal of returning the belt to its intended position. The equations modeling this behavior are:

$$\begin{cases} \mathsf{belt}_k = & \mathsf{belt}_{k-1} - \alpha \cdot \mathsf{motor}_{k-1} + \mathsf{drift}_k \\ \mathsf{integral}_k = & c_{int}(\mathsf{belt}_k + \mathsf{belt}_{k-1}) + \mathsf{integral}_{k-1} \\ \mathsf{motor}_k = & c_{prop}\mathsf{belt}_k + \mathsf{integral}_k \end{cases} \tag{1}$$

Where α, c_{int}, and c_{prop} are known proportionality constants and subscripts $(\cdot)_k$ corresponds to the value at sample k. All three quantities are measured. Notice that in Equation (1) the last 2 equations are taken directly from the implementation of the controller.

Not contained in these equations is the condition that the steering motor stays within a bounded range. If the necessary correction is outside these bounds, the motor will stay at the limit of its range, causing the belt to drift outside of its desired position. Throughout this paper, we assume that the motor and the belt position sensor never fail. This assumption can be relaxed, if necessary, and the proposed methodology can still be applied.

In Equation (1) the drift can be computed at all times since it is a function of the belt and motor positions, both measured. What remains unknown are the different error mechanisms and how they add up to the total drift. For this, we use expert knowledge.

We conjecture that the drift results from the linear combination of five causes:

- Calibration: the belt might not be completely horizontal when the Z-motor is at position 0. This results in a constant calibration error c.
- Misalignment: the belt might not be well aligned with the previous component of the printer, which results in pages coming into the belt with a lateral velocity relative to the direction of motion of the belt, causing drag to one side. This results in a constant misalignment error m that is present only when the machine is printing.
- Degradation: the belt material might wear out and deform over time, resulting in a time-dependent drift D_k. We conjecture this degradation to be exponential and with an unknown deformation direction.
- Sheets: when the pages make contact with the belt, they might cause a perturbation to its position, depending on the properties of the pages. This would result in a train of pulses P_k with varying amplitude and width, present only when the machine is printing.
- Noise: we finally conjecture that all other sources of error add up to a Gaussian term $\varepsilon_k \sim \mathcal{N}(0, \sigma)$ with unknown variance and zero mean.

These causes are described by the following equations:

$$\mathsf{drift}_k = c + \mathsf{print}_k(m + P_k) + D_k + \varepsilon_k \tag{2}$$

$$D_k = s(4^{\delta k} - 1) \tag{3}$$

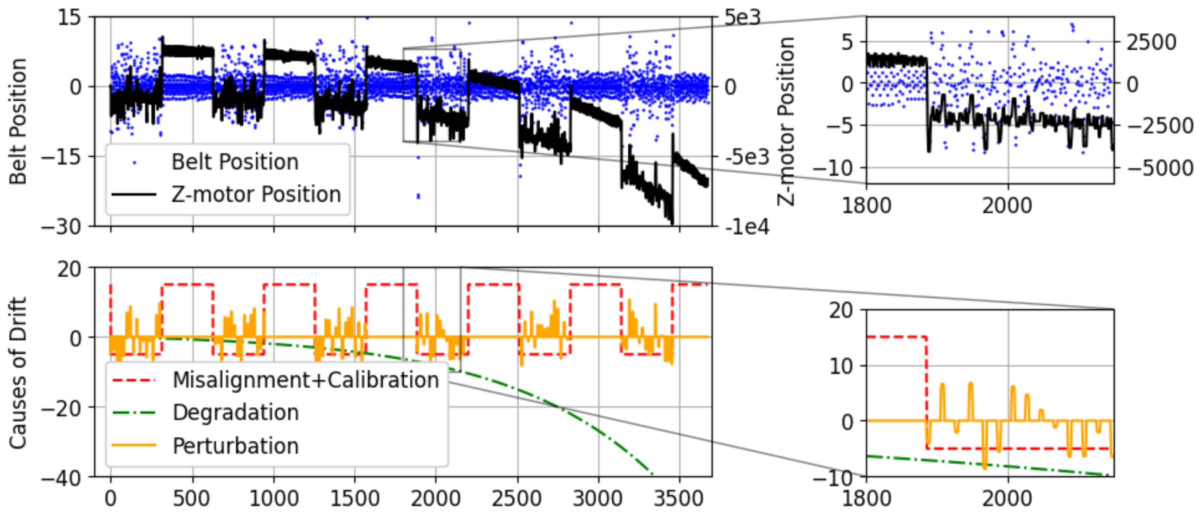

Figure 2. Example synthetic data produced with the simulation model. Observe how the motor displacement follows the sum of the three hidden contributors to the belt drift. See main text for a detailed explanation.

In equations (2,3) we can see the decomposition of the drift into its different terms, with the conjectured form of the degradation as being an exponential with exponent parameter $\delta \geq 0$ and sign $s \in \{-1, 1\}$. print_k is the variable that represents whether the machine is printing and takes values in $\{0, 1\}$. The variables c, m, D_k and P_k in Equation (2) are unknown, while print_k and drift_k are known quantities. In the interest of brevity, we have not included here the detailed equations of the perturbation term P_k.

Considered together, equations (1,2,3) describe a model of the system. The model has been implemented in Python, allowing us to compute simulations such as the one shown in Figure 2. The first author can provide the code if needed to an interested reader.

In the top plot of the figure, we represent the observable time series belt_k and motor_k from Equation (1) in a double-axis, left for the belt and right for the Z-motor. Observe the different units for each time series. We can see that the (simulated) PI-controller is capable of keeping the system controlled in the presence of the Drift causes, shown in the plot below, as evidenced by the belt position remaining stable around 0. It does this by adjusting the Z-motor position. Eventually, the Z-motor will hit its limit, after which the position of the belt quickly drifts away from its intended position (not shown in the plot). In the bottom plot, we show the different contributors to the drift of the belt, for simplicity of visualization we have combined calibration and misalignment in a single one.

4.2. Inference model

The next step in our methodology is to translate the simulation model into a probabilistic model suitable for inference. In such a model, one describes the unknown variables of the simulation as hidden, i.e. unobserved, random variables. Then one describes the known, i.e. observable, variables as functions of the unknown variables, therefore random variables themselves, but which are observed. These functions relating observable and hidden variables can be: probabilistic (e.g. perturbation), or deterministic (e.g. degradation as a function of s and δ), and need not be invertible. The task of these models is to infer the probability distributions of the hidden random variables that best *explain* the observations. We use the framework of probabilistic programming to instantiate these models and perform inference. Numpyro, see (Phan et al., 2019), is the probabilistic programming language of choice for this work.

Considering the temporal nature of our data and the controlled step-wise nature of the system, we propose a Bayesian state-space model as the probabilistic description. A Bayesian state-space model is a dynamical system of equations relating random variables. The system is determined by the observability and update equations. The observability equation (4) connects the vector of observed variables $\vec{y}_k$ at time k to the vector of hidden variables $\vec{\theta}_k$, external observable variables $\vec{x}_k$ and noise term $\vec{\varepsilon}_k$. The update equation (5) connects the vector of hidden variables at time k with the vector of hidden variables at time $k-1$ and the update noise $\vec{\eta}_k$. Together, they define the system:

$$\vec{y}_k = \mathbf{A}_k \vec{\theta}_k + \mathbf{B}_k \cdot \vec{x}_k + \vec{\varepsilon}_k. \quad (4)$$
$$\vec{\theta}_k = \mathbf{G}_k \cdot \vec{\theta}_{k-1} + \vec{\eta}_k, \quad (5)$$

Where $\mathbf{A_k}$ and $\mathbf{B_k}$ are matrices, $\vec{\varepsilon}_k$ is the observation noise vector at time t, $\mathbf{G_k}$ is often called the innovation or transition matrix at time k and $\vec{\eta}_k$ is the update noise. For this system to be fully Bayesian, we can treat the matrices $\mathbf{A_k}$, $\mathbf{B_k}$, and $\mathbf{G_k}$, or their coefficients, as random variables themselves and give them Bayesian priors. The equations of a Bayesian state space model describe the evolution of the hidden and observed variables, but not the evolution of their probability distributions. That is the task that the probabilistic program computes in the background.

The translation from a simulation model like that described by equations (1,2,3) into a Bayesian state-space model is not unique and need not be 1-to-1. For instance, the modeler is free to leave elements of the simulation model out of the inference model, implicitly leaving them as contributions to the noise term. They are also free to choose which unknown variables in the simulation model should be mapped to hidden variables in the Bayesian state space model and which should be expressed as coefficients of the matrices $\mathbf{A_k}$, $\mathbf{B_k}$, and $\mathbf{G_k}$, which are treated as static random variables.

For our conveyor belt, we have only one observed variable, $y_k := \mathsf{drift}_k$. The parameters of the model are defined as

$$\mathbf{A} := [1, 0], \quad \mathbf{B} := [c, m], \tag{6}$$

$$\mathbf{G}_k := \begin{bmatrix} 4^\delta & s \cdot 4^\delta - 1 \\ 0 & 1 \end{bmatrix}. \tag{7}$$

The external variables are $\vec{x}_k = [1, \mathsf{print}_k]$, the hidden variables are $\vec{\theta}_k = [D_k, 1]$ and the noise terms are $\varepsilon_k \sim \mathcal{N}(0, \sigma)$ and $\eta_k := 0$.

For simplicity, we choose not to model perturbation P_k explicitly and let it be absorbed by the noise term ε_k. To make this model fully Bayesian, we assign prior distributions to the random variables c, m, δ, s, and σ. We choose uninformative uniform distributions in the interval $[-80, 80]$ for c, m (the whole range of the belt), a wide uniform distributions in $[0, 0.5]$ for δ, uniform 50% prior for each value of s, and a half-normal distribution with width 1 for σ. The ranges of these priors are chosen by domain experts to encompass all plausible failure configurations.

Once we have expressed the model in this form, the probabilistic programming language performs inference by approximating the joint probability distribution of the hidden and observable variables given a particular observation, and applying Bayes rule. Table 1 shows the result of performing inference on the synthetic dataset from Figure 2. Observe how the inferred values for degradation, calibration and misalignment match their real values.

We conclude that the different causes of drift can be inferred from the synthetic data. This gives us reason to believe that, as long as the data from the real system is sufficiently approximated by the simulation model, the inference can also be performed on the real data. This will be shown in the next section.

Table 1. Inferred values for the different sources of drift considered. Errors express a 2-σ confidence interval.

Parameter	Real value	Inferred value
c	15.00	15.02 ± 0.18
m	-20.00	-20.02 ± 0.20
δ	$8.00\,e^{-4}$	$8.00e^{-4} \pm 7e^{-6}$
s	1.00	1.00 ± 0.00

4.3. Results

Given the long service life of the belts, we adapted the inference procedure to better fit the diagnostic needs in the field by inferring the state of the belt at regular intervals. This allows us to track the state of the machine over time. Algorithm 1 outlines the procedure for this periodic computation of degradation, calibration and misalignment given a stream of field data that is split into periods. For each period, the inference engine takes as input the estimated degradation at the beginning of the period, infers the calibration, misalignment and decay exponent, then computes the additional degradation corresponding to that period, and passes the latter to the next iteration.

Algorithm 1 Iterative belt drift inference

```
$Data \leftarrow [period_1, \ldots, period_n]$
$params \leftarrow []$
$prev\_D \leftarrow 0$
for $period$ in $Data$ do
    $(c, m, \delta, s) \leftarrow InferParams(period, prev\_D)$
    $D \leftarrow CompDegradation(period, \delta, s, prev\_D)$
    $prev\_D \leftarrow D$
    $params.append([c, m, \delta, s, D])$
end for
return $params$
```

The probabilistic programming comes into play in this algorithm when $params[i] = [c^{(i)}, m^{(i)}, \delta^{(i)}, D^{(i)}]$, the inferred parameters for period i, are treated as probability distributions, rather than point estimates. Inferring these distributions is done through the use of Markov Chain Monte Carlo methods for approximating a total probability distribution. The posterior probability conditioned on the observed data is computed via the Bayes-Laplace rule, rather than a standard parameter fitting technique like minimum square error. This is all handled in the background by the probabilistic programming library and implemented via the function $InferParams$ in the algorithm.

To compute the additional degradation in a given period we modify our hypothesis for the degradation (see eq. (3)) to allow for varying decay exponents over the different periods.

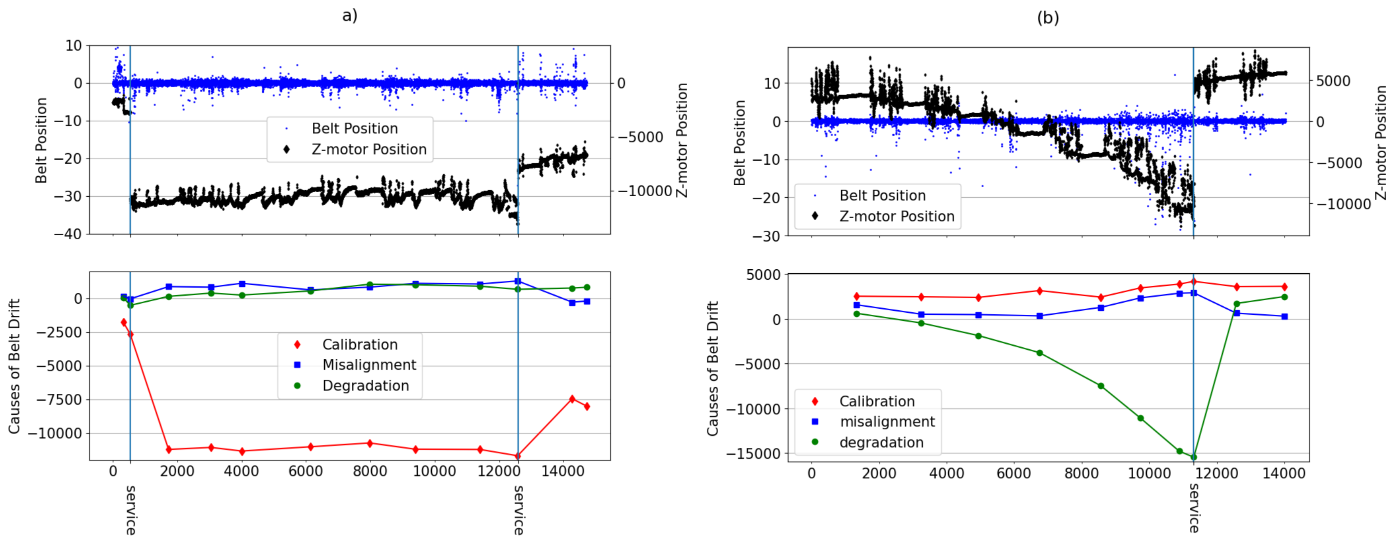

Figure 3. Example of measured data where miscalibration (a), and degradation (b) are the main causes of a belt position error. The Belt and Z-motor positions are measured, while the causes of belt drift in the bottom plots are inferred. The sources of drift are shown here in the units of the Z-position motor rather than the belt for comparison with the former.

We assume that degradation in period i follows the equations:

$$D_k^{(i)} = s \cdot 4^{\delta^{(i)}k} + \mathcal{C}^{(i)}, \quad (8)$$

$$\mathcal{C}^{(i)} = prev_D - s \cdot 4^{\delta^{(i)}k_{i-1}}, \quad (9)$$

where $prev_D = D^{(i-1)} = D_{k_{i-1}}^{(i-1)}$ is the computed degradation at the end of period $i-1$, k_{i-1} is the step that marks the end of period $i-1$, and $\delta^{(i)}$ is the computed decay exponent in period i output by $InferParams$. This condition ensures that degradation grows exponentially and that degradation at the beginning of one period is equal to degradation at the end of the previous period, i.e. it ensures continuity. The computed degradation at the end of period i is then $D^{(i)} := D_{k_i}^{(i)}$.

In Figure 3 we apply the procedure to two typical examples from the field of belts where excessive degradation or miscalibration are the cause of a service action by a service engineer.

Although Figure 3 shows the average degradation, misalignment and miscalibration for each period, we also compute posterior distributions for each parameter, alongside a noise parameter for each period, not shown in the plot. Observe how the three different causes of drift are correctly discriminated and tracked over time. Equipped with these results, a service engineer would perform a replacement of the belt in Figure 3-(b) and a re-calibration of the belt in Figure 3-(a), a much less material and time-consuming action.

5. Conclusions

In this paper, we proposed a methodology for model-based diagnostics of cyber-physical systems leveraging generative and inference models. The generative model is compiled using already available knowledge on failure mechanisms, together with control models, and serves a dual function. On the one hand, it helps validate the expert knowledge on failures, by comparing the results of simulations to data from incidents in the field. On the other hand, it is used to validate the inference models by providing us with a controlled test bench in which to test the ability of the inference model to distinguish the different causes of errors. The inference model is derived from the generative model and is used with field data from real incidents to perform root-cause diagnosis.

We then applied our methodology to the case of an industrial conveyor belt in a closed control loop configuration, with several hidden mechanisms driving the belt out of its desired position. With the proposed methodology we correctly identify the different causes of drift of the belt, thus offering valuable advice for the optimal maintenance action.

To the authors' knowledge, this is the first time probabilistic programming has been used for diagnostics of cyber-physical systems. We believe the present paper proves its utility as a tool for probabilistic modeling and inference in the prognostic and health management domain, opening the door to model-driven and physics-inspired diagnostics. In applying the proposed methodology to the case of an industrial conveyor belt we have identified several aspects for future research. The probabilistic programming framework has a prediction functionality that could be used to make prognostic forecasts of remaining useful life. In future research, we plan on adding this aspect to our methodology. Further, the translation of generative models into inference models is a manual process requiring a certain degree of familiarity with inference and statistical modeling. How to automate, fully or partially, the translation from simulation to inference models remains an open question. Finally, the proposed methodology has been scoped and tested at a subsystem level, comprising a belt, motor, and control mechanism. How to make such models composable and much larger for system-level diagnosis remains a challenge to be addressed in future research.

ACKNOWLEDGMENT

The research is carried out as part of the Carefree program under the responsibility of TNO-ESI in cooperation with Canon Production Printing. The research activities are supported by the Netherlands Ministry of Economic Affairs and Climate, and TKI-HTSM.

The authors would like to thank Robert Passmann and Peter Kruizinga for providing valuable discussions and comments.

REFERENCES

Bartram, G., & Mahadevan, S. (2015). Probabilistic prognosis with dynamic bayesian networks. *International Journal of Prognostics and Health Management*, *6*(4).

Borth, M., & Barbini, L. (2019). Probabilistic health and mission readiness assessment at system-level. In *Proceedings of the annual conference of the phm society* (Vol. 11).

Cox, M., van de Laar, T., & de Vries, B. (2019). A factor graph approach to automated design of bayesian signal processing algorithms. *International Journal of Approximate Reasoning*, *104*, 185–204.

Cranmer, K., Brehmer, J., & Louppe, G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, *117*(48), 30055–30062.

De Kleer, J., & Kurien, J. (2003). Fundamentals of model-based diagnosis. *IFAC Proceedings Volumes*, *36*(5), 25–36.

Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., & Blei, D. M. (2017). Automatic differentiation variational inference. *Journal of machine learning research*.

Lucas, P. J. (2001). Bayesian model-based diagnosis. *International Journal of Approximate Reasoning*, *27*(2), 99–119.

Phan, D., Pradhan, N., & Jankowiak, M. (2019). *Composable effects for flexible and accelerated probabilistic programming in numpyro.*

Särkkä, S., & Svensson, L. (2023). *Bayesian filtering and smoothing* (Vol. 17). Cambridge university press.

Srinivas, S. (1995). *Modeling techniques and algorithms for probabilistic model-based diagnosis and repair.* stanford university.

Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., … Birchfield, S. (2018). Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the ieee conference on computer vision and pattern recognition workshops* (pp. 969–977).

van de Meent, J.-W., Paige, B., Yang, H., & Wood, F. (2018). An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*.

van Gerwen, E., Barbini, L., & Nägele, T. (2022). Integrating system failure diagnostics into model-based system engineering. *INSIGHT*, *25*(4), 51–57.