Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

LOCALIZING FAULTS IN DIGITAL TWIN MODELS BY USING TIME SERIES CLASSIFICATION TECHNIQUES

Joost Mertens[1], and Joachim Denil[1]

[1]Faculty of Applied Eng.: Electronics and ICT, University of Antwerp, Antwerp, BELGIUM

ABSTRACT

To provide its services, a digital twin system relies on models whose behavior adequately simulates/mimics that of the twinned system. Deviations between the behaviors necessitate (user) intervention to realign the model and twinned system. We can intervene in the model (e.g. reinitialization, reparametrization) or in the twinned system (e.g. physical adjustment, replacement of parts). In either case, the taken action depends on the cause of the deviation. One way to find the cause is to use a classifier on the digital twin system's data. In a digital twin system, the data is generally a time-series of states, inputs and outputs. In this paper, we apply a state-of-the-art time series classification technique to detect and classify faults in a digital twin system of a scale-model gantry crane. We find that a classifier trained on simulation data yields adequate performance when applied on to the real system.

1 INTRODUCTION

Cyber-Physical Systems are gaining more and more innovative digital features, driven by the fourth industrial revolution. One of those features is the Digital Twin, a digital model representing a real-world (not necessarily physical) system, which is fed with data gathered during the operation of that real-world system, and which can be adjusted based on that data whenever necessary (Wright and Davidson 2020). A Digital Twin's goal is to provide a set of services that can be exploited by users or other smart manufacturing systems (Cimino et al. 2019). Examples of such services are real-time state monitoring, energy consumption forecasting, user operation guidance, intelligent optimization and failure analysis/prediction (Tao et al. 2018). Some of these services, for example the energy consumption forecasting, rely on high fidelity models of the real-world system. This makes it paramount that the digital model is a valid representation of the real-world system, otherwise the digital twin services will deliver suboptimal or even negative guidance.

The act of checking for similarity/divergence between a model and the real-world system it models is called model validation, traditionally this was applied offline (Sargent 2009; Oberkampf and Roy 2010), but in recent digital twin validation applications we also see it being applied online, at runtime (Lugaresi et al. 2022; Lugaresi et al. 2023; Overbeck et al. 2021; Calvo-Bascones et al. 2023). To determine if there is divergence, online validation is generally coupled with a threshold, which, when breached, indicates warnings/failures. A threshold approach can largely pinpoint which components failed in the system, but, since there may be multiple causes for a threshold breach, identification of the cause within the component is harder, though not impossible, as demonstrated in (Calvo-Bascones et al. 2023).

In this paper, we look to demonstrate that state-of-the-art Time Series Classification (TSC) techniques can be used to augment threshold based model-validation by allowing us to pinpoint the specific observed faults when a component shows unexpected behavior. This knowledge is useful, since it helps to speed up the resolution of the fault. To do so, we identify a set of operational classes of a scale-model gantry crane, for which we generate a set of training traces using only the digital model in the crane's digital twin. We validate this virtually trained classifier against traces made on the real system. The remainder of this paper is structured as follows: section 2 discusses related work on online validation of digital twins. Next, section 3 briefly covers the state-of-the-art in Time Series Classification. Afterward, section 4 describes the gantry-crane setup, and Section 5 describes the way the training and validation data are created. Thereafter, Sections 6 and 7 show the results and discuss them in more detail. Lastly, section 8 concludes the paper.

2 RELATED WORK

We find related work in the fields of online/continuous validation of digital twins, and in the field of anomaly/fault detection. In the online/continuous validation of digital twins, a recurring theme is that of the continuous calculation of validation metrics, e.g. distances, that are compared with acceptance thresholds. When the thresholds are surpassed, (targeted) model updates can be triggered.

Calvo-Bascones et al. (2023) propose an approach based on a concept they term Snitch Digital Twins (SDT). An SDT captures the variations between two variables, for example, the variation between the exhaust gas temperature of two cylinders in a combustion engine. By applying a sliding window over all these variations, they extract three behavioral features: the density distribution of the variable values based on quantiles, the slope of the best-fit line and the intercept point of the best-fit line. An encoder maps the feature values to an index pattern, which is used to detect and diagnose behavioral anomalies.

Lugaresi et al. (2022) apply dynamic time warping (DTW) as a distance measure in the online validation of a manufacturing system. DTW assesses the similarity between two time series and is able to cope with temporal shifts in the data. This is beneficial when comparing measured data with simulated digital twin data, since exact alignment of the samples is not necessary. Based on this distance, they set a warning and failure threshold that triggers the recalibration of the digital twin. They demonstrate that this setup works by applying it to a small manufacturing plant containing two workers and two conveyors. They further this approach with two additional metrics taken from the field of bio-informatics: the Longest Common Sub-sequence (LCSS) and the Modified Longest Common Sub-sequence (mLCSS) (Lugaresi et al. 2023).

Overbeck et al. (2021) implement a continuous validation scheme based on Key Performance Indicators of a production system. They utilize the relative variation and normalized root-mean-square error on the number of produced parts to characterize the operational behavior of the system. Based on thresholds set on these metrics, they perform targeted model updates by looking where the threshold was breached. They apply their technique to a car-parts assembly system consisting of an assembly and testing area. Here they can pinpoint which of the two areas is causing the fault, and update the parameters accordingly.

Muñoz et al. (2022) utlize the Fréchet distance in combination with the Needlemann-Wunsch algorithm to measure the similarity between a digital twin and its actual system. The Needlemann-Wunsch algorithm is used to align the traces, which might be shifted and/or compressed in time, while the Fréchet distance calculates a value signifying the similarity.

One of the techniques in anomaly/fault detection is called model-based fault detection (Gertler 1998). These techniques use data coming from an explicit model of the monitored system to obtain what is called analytical redundancy. Here, the model's outputs are compared with real sensor measurements. The difference between the two are called residuals and can be evaluated to signal faults in the system. Computing the residuals might further utilize techniques such as Kalman filtering.

Feng et al. (2021) applied a Kalman filter-based anomaly detector in the demonstrator case of an incubator system. With their technique, the inadvertent opening of the lid of the incubator can be detected.

Machine learning techniques are also finding their way into anomaly detection, Castellani et al. (2021) use a weakly supervised approach, using the digital twin of a company facility to generate normal operational data, whilst adding a small dataset of labeled anomalies from real-world measurements.

At a higher abstraction level, there are definite similarities between the related work and the techniques presented in this paper, in that we look to detect anomalies in a system, and pinpoint where they occur such that they can be corrected. The main differentiator is that, rather than using a distance with threshold based approach, we take a classification based approach. One advantage of the classification based approach is that it achieves these two goals at once: the system traces are either "normal" or not, and if not, the fault class tells us where the problem exists. A disadvantage is that all fault classes need to be foreseen in advance. The two techniques are not mutually exclusive, nor does a classification based approach supersedes a threshold based approach. Rather, likely, a classification based approach can be used to augment a threshold based approach. For example, the production systems in (Overbeck et al. 2021; Lugaresi et al. 2022; Lugaresi et al. 2023) are simulated as discrete event simulations. These are at a rather high level of abstraction. When the threshold techniques detect something is wrong with the worker cell, a classification based technique can pinpoint exactly what is wrong within that cell.

3 STATE-OF-THE-ART IN TIME SERIES CLASSIFICATION

In Time Series Classification (TSC), the goal is to predict a discrete class for a (multivariate) time series. In this field, the UCR Time Series Archive (Dau et al. 2018) is used by the community to test newly developed algorithms. This yields an objective comparison between the time series classification algorithms. Four algorithms, and their closely related variations, are considered state of the art (Middlehurst et al. 2021): InceptionTime(Ismail Fawaz et al. 2020), TS-CHIEF (Shifaz et al. 2020), ROCKET (Dempster et al. 2020) and HIVE-COTE version 2.0 (HC2) (Middlehurst et al. 2021). Of these four techniques, HC2 is the most accurate one, whilst ROCKET and by extension MiniRocket (Dempster et al. 2021) are by far the fastest to train (Middlehurst et al. 2021).

ROCKET (**R**and**O**m **C**onvolutional **KE**rnel **T**ransform) consists of a random convolutional kernel transformation which transforms a dataset into a set of features, on which a linear classifier can be trained. To create the feature map, each kernel is convolved over each input time series. For its kernels, ROCKET generates 10 000 kernels, each of which is randomized in length, weights, bias, dilation and padding. Of those parameters, dilation is important, since it allows for the detection of features at differing scales and frequencies by a kernel that is otherwise the same. The parameters of this randomization have been experimentally determined on a subset of the UCR Time Series Archive. Subsequently, from each feature map, two aggregate features are computed: the maximum value, and the proportion of positive values. As such, the 10 000 kernels produce 20 000 features. On this set of features, a classifier can be trained, Dempster et al. (2020) recommend either training a ridge regression or a logistic regression depending on the training dataset size. ROCKET has a training complexity that is linear in time series length as well as number of training samples. ROCKET is much faster than other state-of-the-art classifiers, Middlehurst et al. (2021) find it to take 2.85 hours to train the UCR dataset, which is 30 times faster than InceptionTime, the fastest of the other previously mentioned techniques.

MiniRocket (**MINI**mally **R**and**O**m **C**onvolutional **KE**rnel **T**ransform) is a variant of ROCKET, which speeds it up significantly without loss in accuracy (Dempster et al. 2021). MiniRocket maintains the same approach as Rocket: transform an input time series using random convolutional kernels and train a classifier on the transformed features. However, rather than random kernels, it uses a smaller, fixed set of kernels that are mostly deterministic, and it utilizes the properties of those kernels to its advantage to calculate the transform faster. Dempster et al. (2021) show that MiniRocket is on average 30 times faster than ROCKET on the UCR Time Series Archive. As such, MiniRocket impresses by being more accurate than other similarly fast methods, whilst also being faster than other similarly accurate methods.

The highest accuracies in TSC are achieved by ensemble methods, which combine multiple classifiers, each built on a different data representation. These classifiers attain remarkable accuracies at the cost of longer training times due to the different classifiers in the ensemble. HIVE-COTE 2.0 (Middlehurst et al. 2021) is such an ensemble method, which combines four classifiers: a shapelet, convolution, dictionary and interval based approach. A control unit combines the four predictions, for HIVE-COTE 2.0, this control unit is called the Cross-validation Accuracy Weighted Probabilistic Ensemble (CAWPE). CAWPE works by weighing the predicition probabilities of the ensemble components by an accuracy estimate of the component on new (unseen) data. For the convolution based approach, HIVE-COTE 2.0 uses an ensemble of ROCKET classifiers called The Arsenal. Normally, ROCKET with a ridge regressor does not provide class probabilities on the predictions, but by combining multiple ROCKET classifiers in an ensemble, a probability can be obtained through cross-validation of the ridge regression classifiers. Naturally, being

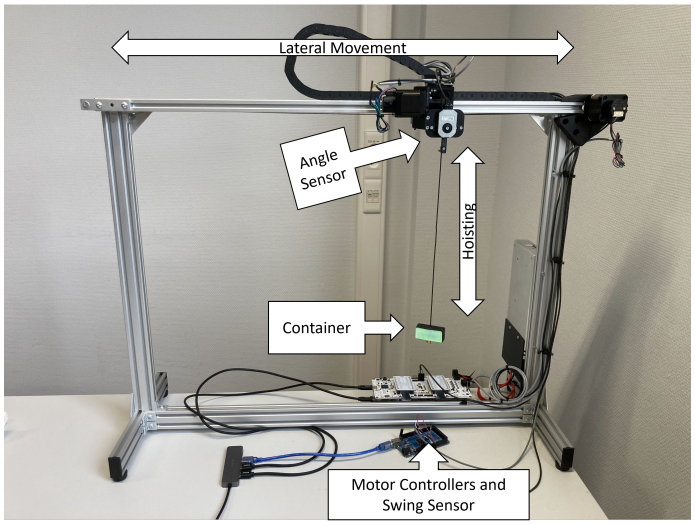

Figure 1: The lab-scale gantry crane.

composed of multiple ROCKET instances, the Arsenal takes longer to train than a single ROCKET instance, but with the benefit of providing predicted class probabilities.

4 USE CASE

The use case on which we demonstrate the technique is that of a lab-scale gantry crane and its accompanying digital twin. A harbor gantry crane's task is to load and unload containers from the quay to docked ships. Figure 1 shows the miniaturized gantry crane and its components. Moving the containers should be done in such a way as to minimize the residual swinging motion at the lowering position. To this end, our gantry crane uses a digital twin service that lets it plan an optimal trajectory to move the container. The crane's motor controllers then enact that trajectory to move the container.

During its operation, this gantry crane runs a continuous validation scheme with threshold levels to indicate potential failures. Every real-world trajectory enactment is paired with the execution of a set of replicated, simulated enactments. Data of both the real-world and the simulations is written to a database, after which a validation component calculates the validation metrics for that trajectory enactment and checks the thresholds. All of this is visualized in a dashboarding application. In what follows, to remain concise, rather than discuss this entire system, we limit ourselves to those components used to generate the data for the training and validation of the classifier.

4.1 Physical Gantry Crane

The crane can move containers horizontally and hoist them vertically. The motors performing this movement are fitted with optical encoders, such that the motor controllers can perform very precise and energy efficient movement. The drives communicate with a host computer over USB, allowing us to set the target position, velocities and other parameters. The cart that rolls along the upper rail has been fitted with an encoder with feeler gauges to sense the container's swinging motion (angle). This encoder is read by an Arduino Mega which communicates the swing angle to the host computer over USB.

4.2 Digital Twin System Components

The digital twin system components used to generate data for our use case are shown in Figure 2. They are:

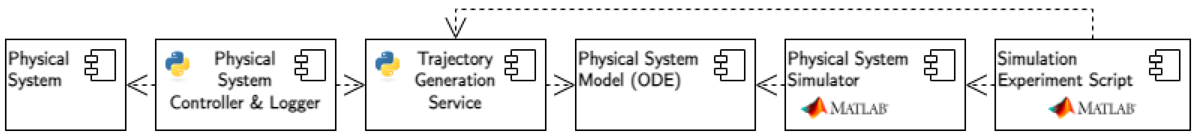
Physical System
Physical System Controller & Logger
Trajectory Generation Service
Physical System Model (ODE)
Physical System Simulator
MATLAB
Simulation Experiment Script
MATLAB

Figure 2: Components used in generating the training and validation data. An arrow means "Uses".

- The **Physical System**, this is the Physical System as described thus far.
- The **Physical System Controller & Logger** uses the **Trajectory Generation Service** to request an optimal trajectory for the movement of the container. It enacts that trajectory on the **Physical System**. Data of the enactment of that trajectory is simply logged to .mat files.
- The **Trajectory Generation Service** is an optimal control problem solver based on the Rockit toolbox (Gillis et al. 2020). Using the **Physical System Model**, the solver solves for a trajectory that moves the container as fast as possible while minimizing the residual swing at the end position.
- the **Physical System** is a set of Ordinary Differential Equations (ODEs) of the crane's kinematics, and its associated parameters and constraints,
- The **Simulation Experiment Scripts** are a set of Matlab scripts used to generate the training data for the TSC. They set up the **Physical System Simulator**, and they use the **Trajectory Generation Service** to generate enactable trajectories.
- The **Physical System Simulator** is in this case Simulink, which can simulate the **Physical System Model**. The Physical System Simulink Model has been enhanced with fault-injection capabilities.

It should be noted that we described the components as if they request the trajectories on the fly, but in fact, as we will describe later, the trajectories are generated up front in batch, then enacted in batch, either by the **Physical System Controller** or the **Simulation Experiment Scripts**. This was done to save some time, since the trajectories would otherwise have been generated 5 times, as they are the same.

4.3 Definition of Operational Classes

We identified 4 operational classes for the gantry crane, discussed below:

1. **Normal operation:** the crane operates as normal.
2. **Roller fault:** a roller fault is a fault related to the roller wheels of the cart limiting the maximum acceleration. It has two main causes, either the adjustable roller wheels are adjusted too tightly around the rail, or the rollers/rails have gunk on them, increasing friction. We can observe this type of fault by inspecting the motor power, since it will be higher to attain a similar acceleration as before. Alternatively, we can also observe this fault by inspecting the acceleration itself, which will be capped to the maximum attainable value given the motor's maximum power. Because our model does not contain the electrical characteristics of the driving motor, we implement this fault as a saturation on the acceleration signal. This fault does not require a trigger, it is either on or off. There are two ways to recover from the roller fault: we could physically inspect the crane and adjust the roller wheels/clean the rail, or we could reparametrize the acceleration limit of the model to the maximum acceleration observed in the physical crane.
3. **Belt fault:** a belt fault is related to the belt connecting the cart to the motor. When improperly maintained, the belt might snap. When this happens, the cart is no longer connected to the motor and forces are no longer transferred. Subsequently, the cart's velocity drops to zero. In the model, we implement this fault as a stuck to zero fault of the cart velocity with a timed trigger.
   We can only recover from a belt fault by replacing the belt.

4. **Rope length fault:** a rope length fault is related to an incorrectly measured hoist rope length. It might occur due to the missing of encoder steps or incorrect spooling of the rope on the drum. In the model, this can be implemented as a stuck-at-constant fault.
   Initial recovery from this fault can be done by a reinitialization of the hoist, this zeros the rope length again. If the fault keeps occurring, physical inspection of the drum/encoder might be necessary.

5 METHOD

5.1 Training and Validation

In this work, we opt to use MiniRocket with a ridge regression classifier for its short training time and high accuracy. Because class prediction probabilities can be interesting when inspecting the results of the trained classifier, we also opt to use the Arsenal. In testing, we also attempted to train two other state-of-the-art classifiers: HIVE-COTE v2.0, since it has the best performance and InceptionTime since it can be trained on GPU's. However, after multiple days of training on our available workstation, we cancelled the training process since it had not yet finished. We deemed the likely marginally better accuracy was not worth the additional training time, especially since Middlehurst et al. (2021) already demonstrated that MiniRocket yields similarly accurate results for a much shorter training time.

We trained our classifiers on three datasets: the measured (simulation results) dataset, the difference between the measured data and the reference trajectory, and a combined dataset that combines the measured and the difference datasets. For MiniRocket we did so with and without added noise. For the Arsenal, we only trained on the datasets with added noise. For the Arsenal we used 25 MiniRocket estimators with 10 0000 kernels. A train/test split of 0.75/0.25 was used for the training. To validate the trained classifier, we applied it to the measured data. The following sections describe the data collection in more detail.

5.2 Training Data Generation

We generate the training data by simulating a set of reference trajectories that are representative of real crane movements. Each simulation is replicated multiple times by randomly sampling the initial conditions (IC) from a distribution characterizing the uncertainty of those IC. Finally, each of the resulting time series was normalized to have a mean of 0 and a standard deviation of 1. This was repeated for each of the classes of operation. In more detail, we undertook the following steps:

1. The distances for the lateral movement of the cart were sampled over the entire width of the gantry, in discrete steps equal to the width of the container. This means moves over a distance between 5 cm to 80 cm, in increments of 5 cm. The same technique was used for the rope length, which was sampled over the entire height of the gantry, in discrete steps equal to the height of the container. This means lengths between 2 cm and 60 cm, in increments of 2 cm. Trajectories were generated both from left to right, and from right to left. This yields 960 reference trajectories.
2. Each of the reference trajectories is simulated, with 10 replications for each trajectory. In Each replication, the estimated (measured) initial position $\hat{x}_0$, rope-length $\hat{r}$, rope-angle $\hat{\theta}_0$ and rope-angular-velocity $\hat{\omega}_0$ were sampled from random distributions representing their uncertainty (measurement error). Those distributions are:
   - $\hat{x}_0$: a normal distribution with mean x_0 (obtained from the reference trajectory) and standard deviation of 0.00025 m (domain expert estimation).
   - $\hat{r}$: a normal distribution with mean r (obtained from the reference trajectory) and standard deviation of 0.00025 m (domain expert estimation).
   - $\hat{\theta}_0$: a logistics distribution with mean θ (obtained from the reference trajectory) and standard deviation of 0.000198 radians (obtained experimentally). The standard deviation and distribution type were obtained by repeatedly measuring θ after manually stabilizing the container.

- $\hat{\omega}_0$: a logistics distribution with mean ω (obtained from the reference trajectory) and standard deviation of 0.000159 radians/s (obtained experimentally). The standard deviation and distribution type were obtained by repeatedly measuring ω after manually stabilizing the container.

Additionally, for the faulty operational classes, randomly sampled faults were injected:

- Roller fault: discussed later, we used a different approach for this type of fault.
- Rope fault: the erroneous rope length r_e was sampled from a uniform distribution between 2 cm and 60 cm. The rope length is considered erroneous when it differs more than +/- 1 cm of the true rope length r (obtained from the reference trajectory). If the random sample fell within +/- 1 cm of r, a new one was drawn until it no longer did.
- Snapped belt fault: a random snapping time t_{snap} was drawn from a uniform distribution between 0 and 1. The drawn value was subsequently multiplied with the length of the trajectory.

3. From the reference trajectories and the simulation data, two sets of data are created: the trajectories as simulated and the difference between the simulated data and the reference trajectory.
4. The data are normalized such that each variable has a mean of 0 and a standard deviation of 1.

The roller fault was simulated in a slightly different way for reproducibility reasons. On the real crane, injecting the roller fault would mean to physically tighten the rollers. This is both hard to quantify, and hard to reproduce due to the tightening mechanism. It would also prove troublesome to reset the roller tightness to the original state. Therefore, for the measurements of the roller fault, rather than tighten the rollers, we opted to generate trajectories with a maximum acceleration higher than the $1.4 m/s^2$ the crane nominally handles. For the data the effect is the same: the real crane seemingly executes the trajectory at a lower maximum acceleration, whilst implementation wise it is far easier to measure, since it does not require physical alteration to the crane. To be consistent in our approach between the measurements on the crane and the generation of the training data, the same technique was also used in simulation. Therefore, for the roller fault, we generated a separate set of reference trajectories, with a maximum acceleration sampled from a uniform distribution between 2 and 4 m/s^2. We opted for the lower bound of 2 rather than the nominal 1.4 to introduce a slight gap between faulty and normal behavior.

For each of the classes of operation this setup normally yields 960 reference trajectories, however the trajectory generator service failed to generate 8 trajectories for the roller fault, yielding only 952 useable reference trajectories. With the 10 replications of each simulation, this yields 38320 data traces to train on.

5.3 Validation Data Measurement

To validate the classifier, we need measurements of trajectories enacted on the real crane. To perform these measurements, we undertook the following steps:

1. The distances for the lateral movement of the cart were sampled over the entire width of the gantry. Due to physical constraints not present in the simulation, the maximum width is only 60 cm rather than 80. Furthermore, we opted to only simulate moves of at least 12 cm. We discretized the moves in steps of 2 cm, rather than 5. This thus means moves over a distance between 12 cm to 60 cm, in increments of 2 cm. To discretize the rope length, we opted for 4 discrete rope lengths: 20, 30, 40 and 50 cm. We did so because this length required manual checking. Trajectories were generated only from left to right. This yields 100 reference trajectories.
2. A subset of the reference trajectories is measured according to the following rules:
   - Normal: Each of the reference trajectories is measured once.
   - Roller fault: Each of the reference trajectories is measured once. As discussed earlier on, the reference trajectories of the roller fault were generated separately.
   - Rope fault: The trajectories with a distance of 12, 28, 44 and 60 cm were selected. These trajectories were replicated 6 times each with a rope length error of +/- 2cm, +/- 5cm and +/- 10 cm from the nominal lengths of 20, 30, 40 and 50 cm. This would yield only 96 measurements,

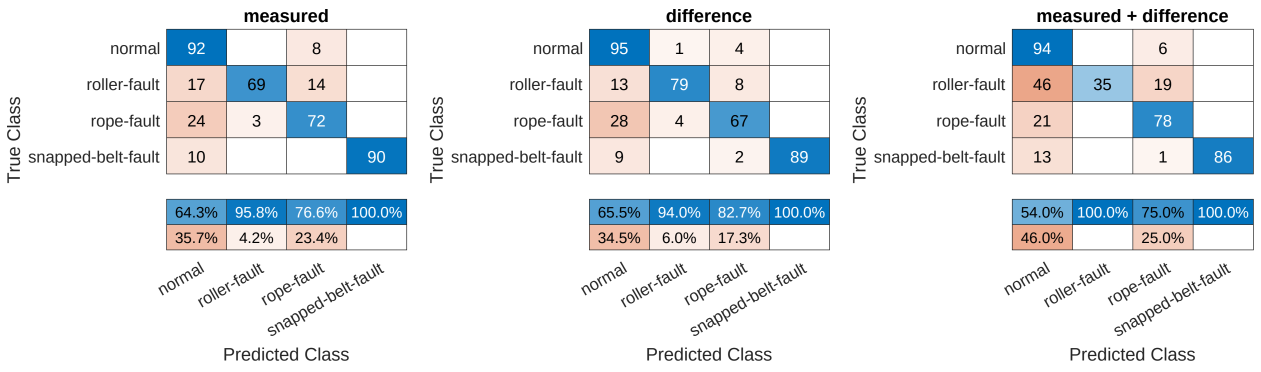

Figure 3: Confusion matrices for the validation results of MiniRocket trained on noiseless data.

so to even out the number of measurements per class, we also made 4 additional measurements with a rope length of -20 cm.

- Snapped belt fault: each of the reference trajectories is measured once, with a snapped belt injected at a random point t_{snap} in time.

3. From the reference trajectories and the measured data, two sets of data are obtained: the trajectories as measured and the difference between the measured data and the reference trajectory.
4. The data are normalized such that each variable has a mean of 0 and a standard deviation of 1.

5.4 Addition of Noise to Training Data

When using simulated training data, it is considered good practice to add noise sampled from the same distribution as the real measurement noise to that data (Jakobi et al. 1995). To perform this noise characterization, we generated 4 trajectories at the extremes of the crane's movement. This means with a rope length of 10 or 50 cm, and a travel distance of 5 or 60 cm. This yields 4 trajectories, each of which was replicated 10 times to characterize the noise. We found the following distributions for our variables:

- x: A normal distribution with a standard deviation of 6.8352e-04 m.
- v: A logistic distribution with a standard deviation of 0.0017 m/s.
- a: A logistic distribution with a standard deviation of 0.4377 m/s^2.
- θ: A normal distribution with a standard deviation of 0.004 rad.
- ω: A logistic distribution with a standard deviation of 0.1489 rad/s.

6 RESULTS

6.1 MiniRocket Results for Noiseless Data

The confusion matrices of these test results are not shown, they achieve 99% test accuracies in all cases.

The confusion matrices of the validation results are shown in Figure 3. The validation results show a significant drop in accuracies. The models trained on the measured, difference and combined datasets get 81, 83 and 73% accuracy respectively. Interestingly, the model trained on the combined data scores less than the two do separately.

6.2 MiniRocket Results with Noisy Data

Adding the noise to the training data has no particular effect on the test accuracies. They are still nearly perfect, achieving 99% test accuracies in all cases. The confusion matrices are thus not shown.

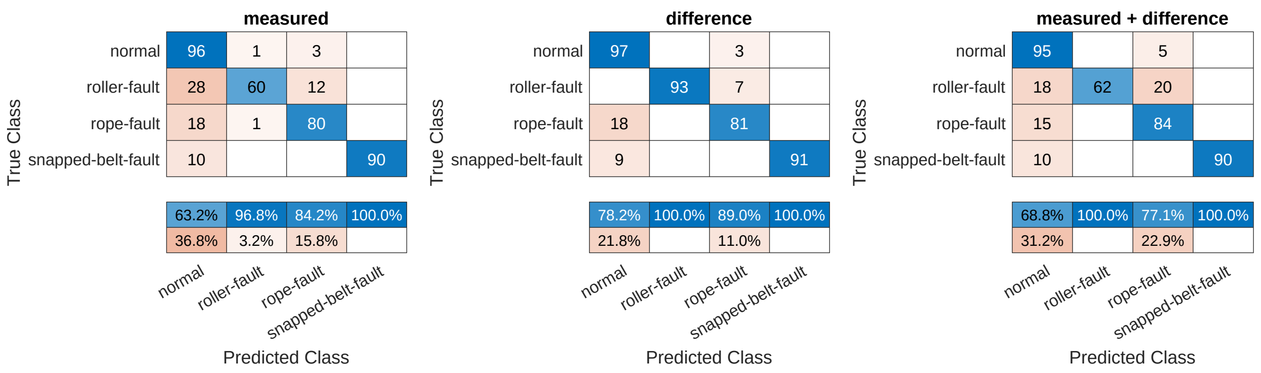

Figure 4: Confusion matrices for the validation results of MiniRocket trained on noisy data.

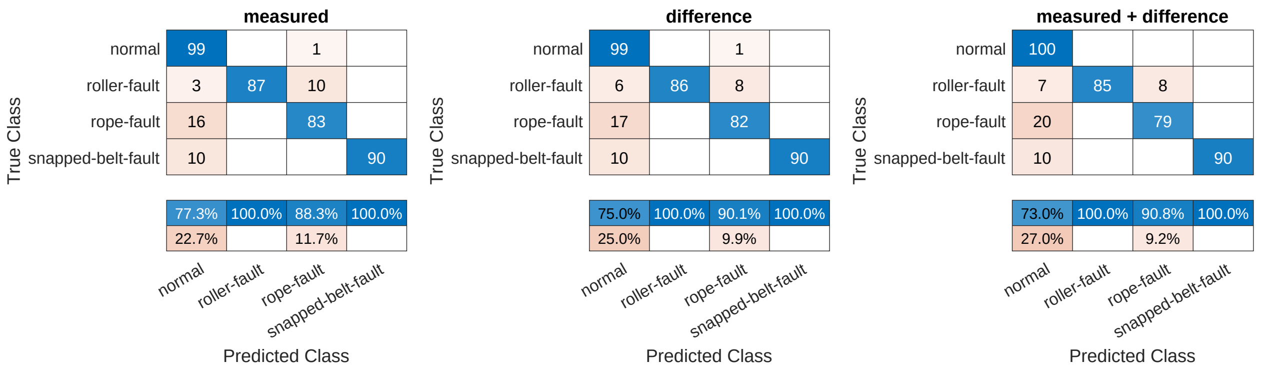

Figure 5: Confusion matrices for the validation results of the Arsenal trained on noisy data.

The confusion matrices of the validation results are shown in Figure 4. Training on the noisy data has a positive effect on the validation accuracies. For the models trained on the measured, difference and combined sets, these are now respectively 82, 91 and 83%. Especially the 91% is a great result.

6.3 Arsenal Results

The confusion matrices of these test results are not shown, they achieve 99% test accuracies in all cases.

The confusion matrices of the validation results are shown in Figure 5. The Arsenal shows better results than MiniRocket in general, the models trained on the measured, difference and combined sets achieve accuracies of respectively 90, 89 and 89%.

7 DISCUSSION

7.1 Explaining the Difference in Accuracy

The results show that for MiniRocket the different datasets have up to 10% different validation accuracies, while for the Arsenal the results are essentially the same. A likely explanation is in the implementation of multivariate MiniRocket. The MiniRocket transformation produces the same number of features regardless of the number of variables (channels). To do so, it randomly samples variables (without replacement) to which a combination of kernels and dilations is applied. Therefore, not every kernel - dilation combination is applied to every channel. By random chance, the best performing channel - kernel - dilation combination might not be selected. This is also the likely reason why the results for the Arsenal are much more alike, since it trains multiple instances of MiniRocket, which reduces the effect of the random variable sampling.

7.2 Inspection of the Misclassifications

There are some hypotheses regarding the misclassifications. We'll try to confirm or refute those. In this analysis, we looked at the results of the Arsenal classifier trained on the difference dataset.

We expect the incorrectly classified rope fault to likely occur for the injected faults of +/- 2cm, as this is the closest to the normal case. Analyzing the results shows that this is indeed the case, with 4 cases being of the +2 cm fault, 6 of the -2 cm fault, but also 7 of the -5 cm fault. The lengths on which the faults were injected were 8 times 50 cm, 6 times 40 cm and 3 times 30 cm. This seems logical, as the faults are absolute, their effect becomes relatively smaller for the longer rope lengths.

For the snapped belt fault, a possible hypothesis is that the random injection of the fault happened closely to the end of the trajectory (over 95%), and that, since the cart is already strongly decelerating at that point, the classifier sees this as normal behavior. Inspecting the results, we can debunk this claim, since only one faulty trajectory fits this description, all others have evenly spaced fault times.

Regarding the roller-fault, we expect the faulty trajectories to have lower maximum accelerations (towards $2m/s^2$), since those would be closest in behavior to the normal trajectories. Inspecting the faulty predictions reveals a slight skew towards the 2.2 to 2.4 m/s^2 trajectories, but not heavy enough to be conclusive, since there are also faulty predictions where the acceleration is 3.7 and 3.9 m/s^2.

It should be noted that these hypotheses stem from our understanding of the traces, but that the classifiers are trained on the ROCKET-transformed features, for which these hypotheses potentially make no sense.

We also inspected the Arsenal class probabilities, hypothesizing that perhaps the faulty predictions scored a low class probability. Here we saw that, compared to the correct predictions, where the predicted class probabilities are higher than 90% for about 80% of the correct predictions, the faulty predictions show less confidence, with about 40% of the predictions yielding a class probability of 90% or higher. Furthermore, about half of the faulty predictions are around a probability of only 60%. There thus is a clear difference in prediction probability distributions, though we had expected the distribution of the incorrect predictions to be even more skewed towards low probabilities.

7.3 Training and Classification Speeds

These timings are measured on a 32 core AMD Threadripper 3970X machine with 128 GB of RAM. Training one MiniRocket and one The Arsenal classifier takes respectively around 5.5 minutes and 2.7 hours. Predictions are fast for both MiniRocket and The Arsenal, taking respectively slightly less than 1 ms and about 50 ms per prediction. The prediction times are many times lower than the trajectory times, which range from 0.6 to 2.8 s. This means both classifiers could be used at runtime if needed.

7.4 Fault Classes

The current set of fault classes is limited, and other classes could be added. One such class could be the "belt skips teeth" class, in which an incorrectly tensioned belt skips teeth under high acceleration. This results in a temporary drop in the velocity of the cart. Besides conceiving more potential faults, the current faults could be refined, e.g. by slimming down the gap between the normal and the faulty class, which can be done for the rope-fault and the roller-fault. We could also refine the faults by looking further into their behavior, for example we observed that, while the roller-fault is implemented symmetrically w.r.t. acceleration and deceleration, the cart actually achieves asymmetric maximum acceleration and decelerations.

7.5 Generalizability and Potential need for Domain Adaptation

A big concern in this approach is the generalizability to other models. The question is mainly if our approach to train on simulation data only is sufficient for other types of models (other than simple kinematics), and if the loss in accuracy when applied to measured data is acceptable. In this case the results carry over well, since the model is a fairly simple kinematics model, yet even so we see a loss of accuracy of about 10%. A more complex model containing more details has the potential to generate data about more aspects of the system, and would therefore allow for more types of faults to be caught. Such a model could include, e.g. the behavior of the motor controllers or the electrical characteristics of the motors themselves. Whether training on simulation data only is sufficient depends on the similarity between the model and the real world, the caveat is that obtaining this similarity becomes harder as the model grows more complex. In case larger losses are observed, a potential approach would be to look into the topic of domain adaptation, which is the field of applying learned models to other domains (in this case it would be from the model domain to the real system's domain). Examples for time series adaptation can be found in (Shi et al. 2022; Wilson et al. 2020).

7.6 Substitute or Augmentation of Online Validation

The TSC approach both determines if a system is operating normally or not, and if it is not, determines the class of fault at the same time. However, since it is unclear how the classifier deals with unforeseen causes of problems, it is not a drop-in replacement for threshold based online validation. With threshold checking, unforeseen causes of faults would just breach the threshold, with the TSC approach the classifier will likely classify the fault as the behaviorally closest fault in the training set, which is not desirable since it could be the "normal" class. As such, this approach should be seen as an augmentation of online validation, providing potentially more information once the threshold is breached.

8 CONCLUSION

In conclusion, we set out with the idea to augment threshold based online validation with TSC techniques, as to accurately pinpoint the source of the fault causing threshold breaches. With the help of a case study of a lab-scale gantry crane, we demonstrated that MiniRocket and the Arsenal, two state-of-the-art TSC techniques, are up to this task. To train these classifiers, we only used simulated data stemming from the crane's models in the crane's digital twin system. We observed some loss in accuracy when applying the classifiers to measured data. It remains to be seen how the classifiers scale with more fault classes, and how well the idea of training only on simulated data generalizes to other, more complex, models. In the future, we aim to incorporate this classifier into our crane's Digital Twin System in such a way that the fault resolution can happen completely automatically, without user intervention, which is currently the case.

ACKNOWLEDGMENTS

Joost Mertens is funded by the Research Foundation Flanders (FWO) through SB research grant 1SD3421N.

REFERENCES

Calvo-Bascones, P., A. Voisin, P. Do, and M. A. Sanz-Bobi. 2023. "A Collaborative Network of Digital Twins for Anomaly Detection Applications of Complex Systems. Snitch Digital Twin Concept". *Computers in Industry* 144:103767.

Castellani, A., S. Schmitt, and S. Squartini. 2021. "Real-World Anomaly Detection by Using Digital Twin Systems and Weakly Supervised Learning". *IEEE Transactions on Industrial Informatics* 17(7):4733–4742.

Cimino, C., E. Negri, and L. Fumagalli. 2019. "Review of Digital Twin Applications in Manufacturing". *Computers in Industry* 113:103130.

Dau, H. A., A. J. Bagnall, K. Kamgar, C. M. Yeh, Y. Zhu, S. Gharghabi *et al.* 2018. "The UCR Time Series Archive". *arXiv: 1810.07758*.

Dempster, A., F. Petitjean, and G. I. Webb. 2020. "ROCKET: Exceptionally Fast and Accurate Time Series Classification Using Random Convolutional Kernels". *Data Mining and Knowledge Discovery* 34(5):1454–1495.

Dempster, A., D. F. Schmidt, and G. I. Webb. 2021. "MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification". In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, edited by J. Tang and T. Derr, 248–257. New York: Association for Computing Machinery.

Feng, H., C. Gomes, C. Thule, K. Lausdahl, A. Iosifidis, and P. G. Larsen. 2021. "Introduction to Digital Twin Engineering". In *2021 Annual Modeling and Simulation Conference (ANNSIM)*, edited by C. Ruiz Martin, M. J. Blas, and A. Inostrosa Psijas, 1–12. New York: Institute of Electrical and Electronics Engineers.

Gertler, J. 1998. *Fault Detection and Diagnosis in Engineering Systems*. Boca Raton: CRC press.

Gillis, J., B. Vandewal, G. Pipeleers, and J. Swevers. 2020. “Effortless Modeling of Optimal Control Problems with Rockit”. In *39th Benelux Meeting on Systems and Control*, edited by R. Carloni, B. Jayawardhana, and E. Lefeber, 138. Groningen: University of Groningen.

Ismail Fawaz, H., B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber *et al*. 2020. “InceptionTime: Finding AlexNet for Time Series Classification”. *Data Mining and Knowledge Discovery* 34(6):1936–1962.

Jakobi, N., P. Husbands, and I. Harvey. 1995. “Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics”. In *Advances in Artificial Life*, edited by F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, 704–720. Berlin, Heidelberg: Springer.

Lugaresi, G., S. Gangemi, G. Gazzoni, and A. Matta. 2022. “Online Validation of Simulation-Based Digital Twins Exploiting Time Series Analysis”. In *2022 Winter Simulation Conference (WSC)*, 2912–2923 https://doi.org/10.1109/WSC57314.2022.10015346.

Lugaresi, G., S. Gangemi, G. Gazzoni, and A. Matta. 2023. “Online Validation of Digital Twins for Manufacturing Systems”. *Computers in Industry* 150:103942.

Middlehurst, M., J. Large, M. Flynn, J. Lines, A. Bostrom, and A. Bagnall. 2021. “HIVE-COTE 2.0: a New Meta Ensemble for Time Series Classification”. *Machine Learning* 110(11):3211–3243.

Muñoz, P., M. Wimmer, J. Troya, and A. Vallecillo. 2022. “Using Trace Alignments for Measuring the Similarity between a Physical and Its Digital Twin”. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, edited by T. Kühn and V. Sousa, 503–510. New York: Association for Computing Machinery.

Oberkampf, W. L. and C. J. Roy. 2010. *Verification and Validation in Scientific Computing*. Cambridge: Cambridge University Press.

Overbeck, L., A. Le Louarn, O. Brützel, N. Stricker, and G. Lanza. 2021. “Continuous Validation and Updating for High Accuracy of Digital Twins of Production Systems”. In *Simulation in Produktion und Logistik 2021*, edited by J. Franke, 609–617. Göttingen, Germany: Cuvillier Verlag.

Sargent, R. G. 2009. “Verification and Validation of Simulation Models”. In *2009 Winter Simulation Conference (WSC)*, 162–176 https://doi.org/10.1109/WSC.2009.5429327.

Shi, Y., X. Ying, and J. Yang. 2022. “Deep Unsupervised Domain Adaptation with Time Series Sensor Data: A Survey”. *Sensors* 22(15):5507.

Shifaz, A., C. Pelletier, F. Petitjean, and G. I. Webb. 2020. “TS-CHIEF: a Scalable and Accurate Forest Algorithm for Time Series Classification”. *Data Mining and Knowledge Discovery* 34(3):742–775.

Tao, F., J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui. 2018. “Digital Twin-Driven Product Design, Manufacturing and Service with Big Data”. *The International Journal of Advanced Manufacturing Technology* 94(9):3563–3576.

Wilson, G., J. R. Doppa, and D. J. Cook. 2020. “Multi-Source Deep Domain Adaptation with Weak Supervision for Time-Series Sensor Data”. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, edited by J. Tang and B. A. Prakash, 1768–1778. New York: Association for Computing Machinery.

Wright, L. and S. Davidson. 2020. “How to Tell the Difference Between a Model and a Digital Twin”. *Advanced Modeling and Simulation in Engineering Sciences* 7(1):1–13.

AUTHOR BIOGRAPHIES

JOOST MERTENS is a PhD student at the University of Antwerp, Faculty of Applied Engineering in Electronics and ICT. His research interest is the continuous evolution of Cyber-Physical Systems. Particularly, he is interested in the synchronization of digital twins and their actual system, through validation processes. He is also interested in the usage of the Digital Twin throughout the software release management process. His email address is joost.mertens@uantwerpen.be.

JOACHIM DENIL is an associate professor at the University of Antwerp, Faculty of Applied Engineering in Electronics and ICT. His research interests are enabling methods, techniques and tools to design, verify and evolve Cyber-Physical Systems. Specifically, he is interested in the performance modelling and simulation, model-based systems engineering, and verification and validation of models and systems. He serves as an associate editor of the Transaction of the SCS: Simulation. His email address is joachim.denil@uantwerpen.be.