Recurrent Neural Networks for real-time distributed collaborative prognostics

Adrià Salvador Palau*, Kshitij Bakliwal†, Maharshi Harshadbhai Dhada†, Tim Pearce*, and Ajith Kumar Parlikad*
*Department of Engineering, Institute for Manufacturing, University of Cambridge, Cambridge, CB3 0FS, UK
Email: as2636@cam.ac.uk, aknp2@cam.ac.uk
†Indian Institute of Technology (IIT) Indore, Khandwa Road, Simrol, Indore 453552 India
Email: ee140002018@iiti.ac.in, me140003015@iiti.ac.in

***Abstract*—We present the first steps towards real-time distributed collaborative prognostics enabled by an implementation of the Weibull Time To Event - Recurrent Neural Network (WTTE-RNN) algorithm. In our system, assets determine their time to failure (TTF) in real-time according to an asset-specific model that is obtained in collaboration with other similar assets in the asset fleet. The presented approach builds on the emergent field of similarity analysis in asset management, and extends it to distributed collaborative prognostics. We show how through collaboration between assets and distributed prognostics, competitive time to failure estimates can be obtained.** [1]

I. Introduction

The earth's biosphere is formed by trillions of interdependent beings that continuously compete, cooperate and learn from each other. In the animal kingdom animals often live clustered in smaller groups, such as cultures, pods, hives, packs etc. Animals from the same species can learn completely different strategies in order to thrive in different environments and situations. For example, different pods of killer whales hunt in many distinct ways, and teach other members in the pod these diverse hunting strategies [1]. Humans do similarly: the typical life of a coal miner from central North America has very little to do with the life of a journalist in the East coast. Humans have thrived as a species within other reasons because they excel in adapting and learning the best strategies conducive to their living environment. These strategies are not only group specific, but also individually adapted to the potential of each human given by its health, intellectual ability etc.

In the recent decades, industrial assets have acquired some of the capabilities that until now were exclusive to the animal kingdom. With the spread of IoT technologies, assets are now able to communicate, process data, act over and sense from the medium [2]. These animal-like properties have fuelled research aimed to provide assets with a certain degree of agency, setting the basis for concepts such as *intelligent* industrial assets [3]. Despite the declared intelligence of these industrial agents, collaboration in asset fleets has been largely unexplored. However, a rich field of algorithmic solutions based on biological learning strategies has existed for years, these include computational frameworks such as reinforcement learning and swarm intelligence in Multi Agent Systems [4]. In swarm intelligence, approaches such as Particle Swarm Optimization [5], [6] and the artificial bee and ant colony algorithms have obtained remarkable success in solving problems using biologically-inspired distributed, cooperative architectures [7], [8].

The algorithmic approaches described before have succeeded to artificially replicate the diverse, adaptable, distributed, flexible, resilient and scalable nature of biological Multi Agent Systems up to a reasonable extent [9]. However, these approaches often do not refer to truly distributed scenarios, because the agents in such systems are usually not linked to real assets in the physical world. Instead, the agents in these approaches are often positions in the solution space (Particle Swarm Optimisation) or theoretical constructs aimed to approximate these solutions (artificial bee algorithm). Ideally, one would like the agents in an industrial Multi Agent System to be assigned to truly physically distributed assets (typically industrial machines). Architectures to bridge this gap have for long existed, and much of the theoretical framework has been proposed in the fields of Holonic Manufacturing systems and Multi Agent Systems applied to manufacturing [9], [10].

Despite existing multi agent architectures claiming their employability for all kinds of asset management scenarios, very few examples exist to day where collaborative learning is used to improve asset prognostics in a distributed real-time Multi Agent System. Some of the closest proposed approaches are federated learning [11], cohort analysis [12], and similarity analysis [13], although only the later has been used in asset prognostics. Federated learning is a technique introduced in 2017 that enables deep neural networks to train locally in smart-phones in order to update a general model in the cloud without the need of sharing private agent data [11]. Federated learning, however, does not consider similarity between clients and does not necessarily produce a client-specific model. Cohort analysis is an old concept, based on clustering individuals in a population according to their similarity in behaviour at a

[1]Sponsors: Adrià Salvador work was sponsored by a Doctoral Scholarship provided by "Fundació la Caixa". This research was also supported by SustainOwner (Sustainable Design and Management of Industrial Assets through Total Value and Cost of Ownership), a project sponsored by the EU Framework Programme Horizon 2020, MSCA- RISE-2014: Marie Skodowska-Curie Research and Innovation Staff Exchange (Rise) (grant agreement number 645733 Sustain-owner H2020-MSCA-RISE-2014) . Copyright: 978-1-5090-0382-2/16/$31.00 ©2018 IEEE

given time period. Although extensively applied in fields such as the social sciences and medicine, to our best knowledge cohort analysis has not yet been explicitly used in the field of asset management.

In this paper, we present the first implementation of distributed collaborative prognostics using a combination of state-of-the-art prognostics algorithms and an ad-hoc architecture extracted from [14]. In the proposed implementation, each asset is assigned a Digital Twin that processes asset data and performs prognostics. This Digital Twin is also able to connect to a Social Platform that collects data from it and other Digital Twins and enables inter-asset communication, and thus cooperation.

The prognostics algorithm implemented in the Digital Twins is based on the Weibull Time To Event Recurrent Neural Networks (WTTE-RNN) algorithm [15]. Proposed in 2016, the WTTE-RNN approach combines survival analysis theory with Recurrent Neural Networks in order to train a Weibull probability distribution of the remaining time to event for censored and uncensored sensor data. The algorithm is chosen for its simplicity and versatility, and because it is fit for the purpose of discrete event prediction.

Following the Abstract and the Introduction, this paper commences by describing the system proposed to enable distributed collaborative prognostics (Sec. II). The description of the system is divided in two parts: a sketch of the employed architecture (Sec. II-A), and a description of the methodology proposed to implement collaborative prognostics (Sec. III). After the system description, its implementation for the case of the C-MAPPS turbine degradation data set is presented in detail in Sec. III. Firstly, the process of data preparation is discussed (Sec. III-A), this is followed by a description of the employed algorithms (Sec. III-B). The section concludes with a presentation and discussion of the results obtained in the C-MAPPS data set (Sec. III-C). The paper concludes with a discussion and a summary of future work proposed by the authors (Sec. IV), followed by a discussion of computational time constraints (Sec. III-D).

II. System Description

A. Architecture

The architecture used in this paper is a modified hierarchical architecture with three layers: Virtual Assets, Digital Twins, and a Social Platform (see Figure 1) [9], [14]. Such kind of architecture is a modification of a purely hierarchical architecture, featuring also horizontal communication between agents on the same level [9]. In the lower layer of the architecture, each asset is assigned a Virtual Asset, which is simply a piece of code that standardizes the asset's data and sends it to the asset's Digital Twin. This design is chosen in order to allow a unique Digital Twin design for a wide range of industrial assets. The Digital Twin processes the data from the Virtual Assets, and from other assets in the fleet to perform asset management tasks using its analytics engine. Apart of receiving data from the Virtual Asset, the Digital Twin can also communicate with the Social Platform, which forms the uppermost layer of the system's architecture. The Platform can perform different tasks, from collecting and presenting system-level data to running clustering algorithms to form groups of collaborating assets.

In the sections that follow, we describe how we have designed the Virtual Assets, Digital Twins and Social Platform in order to adapt them to perform distributed collaborative prognostics. The original architecture, designed to facilitate collaborative learning, is described in more detail in [14].

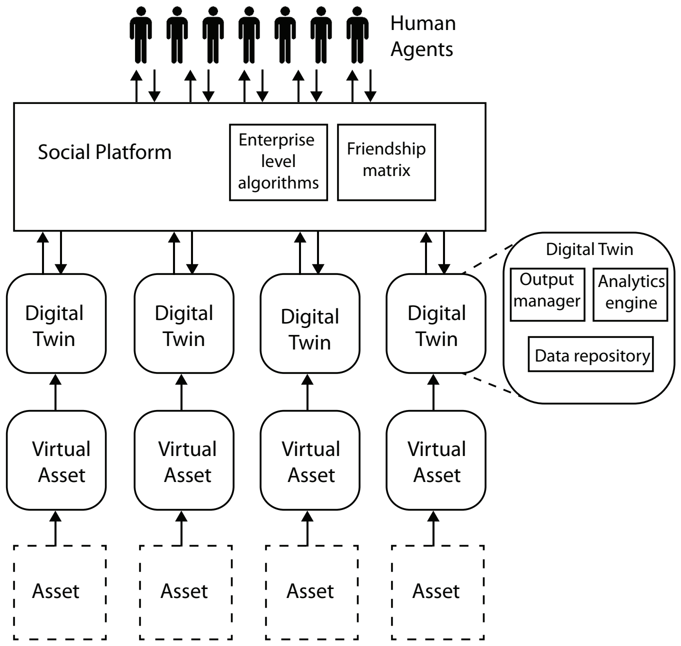


Fig. 1. Sketch of the architecture used in this paper. Black arrows indicate communications between its elements. Human agents and assets are not considered to be part of the software architecture, as they are elements in the physical world. The architecture is directly extracted from [14].

1) Virtual Asset: as the lowest-level blocks of the system's architecture, the Virtual Assets are originally designed to standardize the data coming from their corresponding assets. However, another function from these architectural elements can be to simulate machines working in real time from an existing prognostics data set [14]. In the later case, the Virtual Assets load the data from a data file, convert it into the system's standard format and send it at fixed time intervals to their assigned Digital Twins. This task can, of course, be omitted in case of dealing with real-time machines. In all cases the data is divided in three main components: a set of features (sensor values together with the time at which they have been recorded), a set of timed events (related to machine failures or warnings), and a machine identifier. The machine identifier must be designed so that it contains all the information related to the asset not featured directly in the feature and event data (for example, machine make, country of deployment, etc).

2) Digital Twin: each Virtual Asset has its assigned Digital Twin, a software element consisting of three building blocks: a Data Repository, an Output Manager, and an Analytics Engine. The Analytics Engine of the Digital Twin is responsible for performing prognostics, and separating the events received from the Virtual Asset to discern between failure events and other events unrelated to machine prognostics. The Output Manager is responsible for managing the communication between the Digital Twin and the Virtual Asset, and between the Digital Twin and the Social Platform (the higher layer of the system's architecture). Finally, the Data Repository stores and manages the data generated by the Analytics engine and the Output Manager. This data is divided in five main sets: the three sets of data generated by the Virtual Assets mentioned in the previous paragraph, a set of variables defined by the Social Platform, and a set of variables generated by the Prognostics algorithm. It is important to mention that collaborative prognostics will involve data sharing between assets. Therefore, the data used in the Analytics Engine of the Digital Twins consists not only of the data coming from the Virtual Asset assigned to each Digital Twin, but of data coming from other collaborating Digital Twins.

Only a subset of the data generated by the prognostics algorithm is kept permanently in the Data Repository: the information determining the prognostics model at each time step and the model prediction or predictions. These parameters are then used to estimate the performance of the prognostics algorithm. The variables defined by the Social Platform stored in the Digital Twin's Data Repository are parameters such as information on the events of interest for a particular Digital Twin, the similarity distances between the twin's asset and each other asset in his groups of collaborating assets, and information regarding the features to be used in the prognostics algorithm.

3) Social Platform: as the highest-level layer of the system's architecture, the Social Platform is responsible for enabling and regulating communication between Digital Twins. Additionally, the Social Platform runs enterprise-level algorithms. These algorithms are aimed at (1) forming groups of collaborating Assets, and (2) retrieving and plotting enterprise-level information, specifically measures of the accuracy of the prognostics computed in the Digital Twins.

The Social Platform uses features, event information and machine identifiers from the Digital Twins in order to form groups of collaborating assets. The information about which assets will collaborate with each other is saved in a matrix, featuring also scalar distances between assets. This is known as the Friendship Matrix. These distances can be calculated as per definition of the Asset Manager, from simple euclidean distances including only continuous attributes, or from heterogeneous distances including also discrete asset attributes such as the asset identifier.

B. Real-time collaborative prognostics

1) Prognostics: In order to perform real-time prognostics, we choose to implement a novel machine learning approach known as Weibull Time To Event - Recurrent Neural Networks (WTTE-RNN) proposed in 2016 [15]. This approach has the benefit that it is designed to solve multivariate time to event prediction problems using both censored and uncensored data in the training of the Recurrent Neural Network. For the sake of completeness we provide a brief description here. The reader is referred to [15] for full details.

The WTTE-RNN approach combines techniques from survival analysis and Recurrent Neural Networks. In WTTE-RNN, a log-likelihood loss function is proposed that allows training a Recurrent Neural Network to provide the two determining parameters of a Weibull probability distribution of the Time To Event for a vector of multi-sensor feature data. The proposed log-likelihood function to be maximized by the Recurrent Neural Network is:

$$\log(\mathcal{L}) = \sum_{n=1}^{N}\sum_{t=0}^{T_n} u_t^n \log\left[\Pr\left(Y_t^n = y_t^n | x_{0:t}^n\right)\right] + (1-u_t^n)\log\left[\Pr\left(Y_t^n > y_t^n | x_{0:t}^n\right)\right]. \quad (1)$$

Where u_t^n indicates whether the observation at time t is censored (the real failure time, $u_t^n = 0$ has not yet been observed). The first term in the right hand side of the equation is $u_t^n \log\left[\Pr\left(Y_t^n = y_t^n | x_{0:t}^n\right)\right]$, which simply means: *in case that the real time to failure ($u_t^n = 1$, uncensored) has been observed, maximise the probability of the predicted time to failure Y_t^n being equal to the real time to failure y_t^n given the known values of the sensor value time series before time t, $x_{0:t}^n$.* The second term, $(1-u_t^n)\log\left[\Pr\left(Y_t^n > y_t^n | x_{0:t}^n\right)\right]$ means: *if the real time to failure ($u_t^n = 0$, censored) has not been observed, maximise instead the probability of the predicted time to failure Y_t^n being bigger than time left until the time at which we know that there has been no failure yet (y_t^n).* The summations $\sum_{n=1}^{N}\sum_{t=0}^{T_n}$ account for the summation over all the recorded failure trajectories (N) and over all the time-steps of each trajectory (T_n). In order to clarify what do we mean by trajectory, we have included a sketch of the matrix fed to the Recurrent Neural Network (see Figure 2) The probabilities

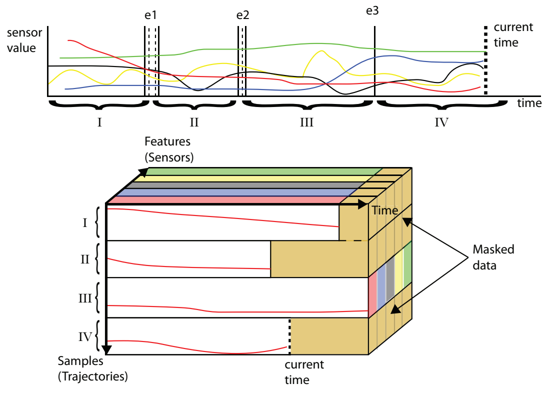


Fig. 2. Sketch showing the training data matrix fed to the asset's Recurrent Neural Network for the case of *no collaboration*. For the case of collaboration, additional trajectories from different assets in the fleet will be added to the training data matrix. Masked data refers to a fill-in number used to let the algotihm know that the values attached are not to be taken in account when training the Recurrent Neural Network.

appearing in Eq. (1) can be obtained by means of survival analysis, in essence for the discrete case it can be shown that:

$$\log(\mathcal{L}) = u\log\left(e^{d(t)} - 1\right) - \Lambda(t+1). \tag{2}$$

Where $\Lambda(t)$ is known as the cumulative hazard function and $d(t) = \Lambda(t+1) - \Lambda(t)$ is the step cumulative hazard function. $\Lambda(t)$ is defined as the integral of the hazard function ($\lambda(t)$):

$$\Lambda(t) = \int_0^t \lambda(w)dw; \qquad \lambda(t) = \lim_{\epsilon\to 0} \frac{Pr(t < T \leq t+\epsilon | T > t)}{\epsilon} \tag{3}$$

Where T is a positive random variable. If one assumes that T follows a Weibull distribution[2], the cumulative hazard is:

$$\Lambda(t) = \left(\frac{t}{\alpha}\right)^{\beta}. \tag{4}$$

Where α is the scale parameter and β is the shape parameter. Combining Eqs. (2) and (4) the discrete log-likelihood (added over all trajectories and all time-steps, and using the concept of Recurrent Cumulative Hazard Function as shown in [15]) can be shown to be:

$$\log(\mathcal{L}_d) = \sum_{n=1}^{N}\sum_{t=0}^{T_n}\left(u_t^n \log\left\{\exp\left[\left(\frac{y_t^n+1}{\alpha_t^n}\right)^{\beta_t^n} - \left(\frac{y_t^n}{\alpha_t^n}\right)^{\beta_t^n}\right] - 1\right\} - \left(\frac{y_t^n+1}{\alpha_t^n}\right)^{\beta_t^n}\right). \tag{5}$$

Where α_t^n, β_t^n are the parameters of the Weibull distribution and y_t^n is the time to event or failure at each time-step t and trajectory n. Note that the left term will appear when there is no censoring, and for $\beta_t^n \to \infty$, the Weibull distribution corresponds to the Dirac delta function at $t = \alpha_t^n$. This is the expected behaviour as the ideal prediction is a probability distribution centred at the real time to event with zero variance.

The unconstrained optimization problem to be solved by the Recurrent Neural Network can be then summarized in finding the weights w that maximize $\log(\mathcal{L}_d)$. A comprehensive description can be found in the original source [15].

The dependency of the shape of a Weibull distribution (and more specifically its variance) with its defining parameters, α and β, make solving this optimization problem often numerically difficult. Eq. (5) features some opportunities for numerical instabilities: negative values of the logarithm's argument and exploding gradients being the most common. In solving this, we followed the suggestions in [15], which can be summarized as:

1) Setting up a maximum for allowed β_t^n. If not capped by a superior limit, the optimization algorithm has a tendency to drive β_t^n to very large values, as this corresponds to a nearly perfect prediction. This tends to cause exploding gradients or over-fitting. An effect of setting such a maximum value for the shape parameter is that predictions close to failure tend to converge to low values of beta, as this is the most effective way that the optimization algorithm has to reduce the variance of the predicted distribution for low values of α_t^n.
2) Initialization of α_t^n, and β_t^n. The values at which the parameters of the Weibull distribution are initialized have been observed to have a big influence on numerical stability and on convergence to a desirable solution. In this paper, we follow the suggestion by [15] and initialize our parameters as a *geometric initialization*. This corresponds to $\beta_t^n = 1$, and $\alpha_t^n = -\frac{1}{\log\left(1-\frac{1}{1+\bar{y}_i}\right)}$, where $\bar{y}_i$ is the mean time to failure at $t = 0$, $\bar{y}_i = \frac{1}{n}\sum y_0^n$.

Each Digital Twin in the asset fleet implements a Recurrent Neural Network training algorithm in order to maximise Eq. (5), and predict further failures. While prognostics are performed in real-time, the Recurrent Neural Network can be trained at fixed time periods T_{T} in order to reduce computational costs. This is known as the system time period.

2) Collaboration: In this work, inter-asset collaboration is defined as the sharing of information between similar assets. This shared data is then weighted according to a measure of similarity and used to train asset-specific prognostics algorithms. In order to allow collaboration, the assets in the fleet must be clustered according to a method that incorporates an inter-asset difference metric d_{ij}, where i and j are indexes of the assets i and j within the fleet. For non-trivial cases, a measure of clustering tendency should be calculated first in order to assess whether the asset population has any grouping structure [16]. Measures such as the multidimensional Cox-Lewis and Hopkins statistics are good examples of metrics allowing for efficient estimation of the clustering tendency. If this tendency is found to be strong, the number of clusters in the fleet should be determined by a combination of expert input and one or more of the existing methods such as the Minimum Description Length, Aikaike Information Criterion, Bayes Information Criterion etc.

Once the clusters are set, the collaborative aspect of the presented approach comes from the data used in each Digital Twin to train its Recurrent Neural Network. Each Digital Twin obtains feature and event data from other Digital Twins in the asset fleet through communication with the Social Platform. This data is then used in the training of the Recurrent Neural Network to improve its accuracy. The data can be incorporated in the neural network's training either weighting the trajectories according to the inter-asset difference metric d_{ij}, or directly without modification (see Figure 2).

A straightforward way to perform such weighting is through ensuring that the number of trajectories to failure in each Digital Twin is in proportion to the distance of its corresponding asset with all the other different assets in the cluster. Take that $d_i^{\max}$ is the maximum inter-asset distance in a cluster calculated from asset i. The proportion of each other asset's trajectories to failures to be weighted in its Digital Twin's prognostics algorithm is then given by:

$$p_j = 1 - \frac{d_{\mathrm{ij}}}{\sum_j d_{\mathrm{ij}}}. \tag{6}$$

[2]With the following parametrization: $f(t) = \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1}\exp\left[-\left(\frac{t}{\alpha}\right)^{\beta}\right]$.

Assuming that all assets must be at least represented by one trajectory, the minimal number of trajectories to be incorporated in the Recurrent Neural Network's training from each asset in the cluster is:

$$n_j = \frac{p_j}{\min(p_j)}. \tag{7}$$

III. Implementation in the C-MAPPS data set

A. Data preparation

In order to implement the proposed approach, we use the Turbofan Engine Degradation Simulation Data Set [17] (from now on, referred as C-MAPPS data set, in reference to the code used to generate it). The proposed framework is specially relevant for machines that experience recursive failures. Instead, the C-MAPPS data set features trajectories to failure unique to every machine in the data set. In order to use this data for our approach, we must treat several trajectories to failure as if corresponding to the same asset. To do so, we only use trajectories known to belong to similar environmental conditions (corresponding to turbines operating at sea level), and divide them in 5 sets of 20 trajectories, each set representing a virtual machine, linked to a Virtual Asset. Clustering in the data set is made possible by the fact that some of there machines include failures due to fan degradation, and some other feature failures due to High Pressure Compressor degradation (see Figure 3).

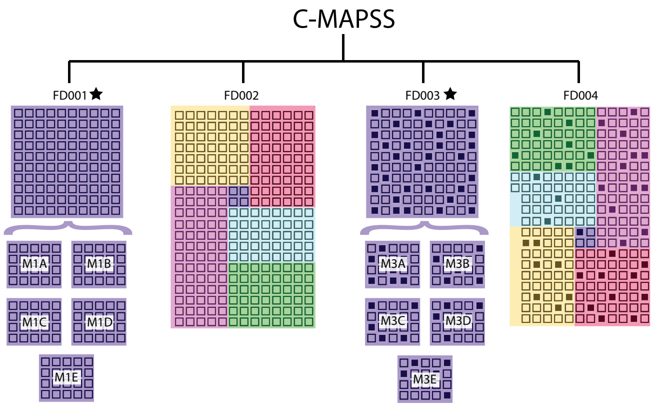
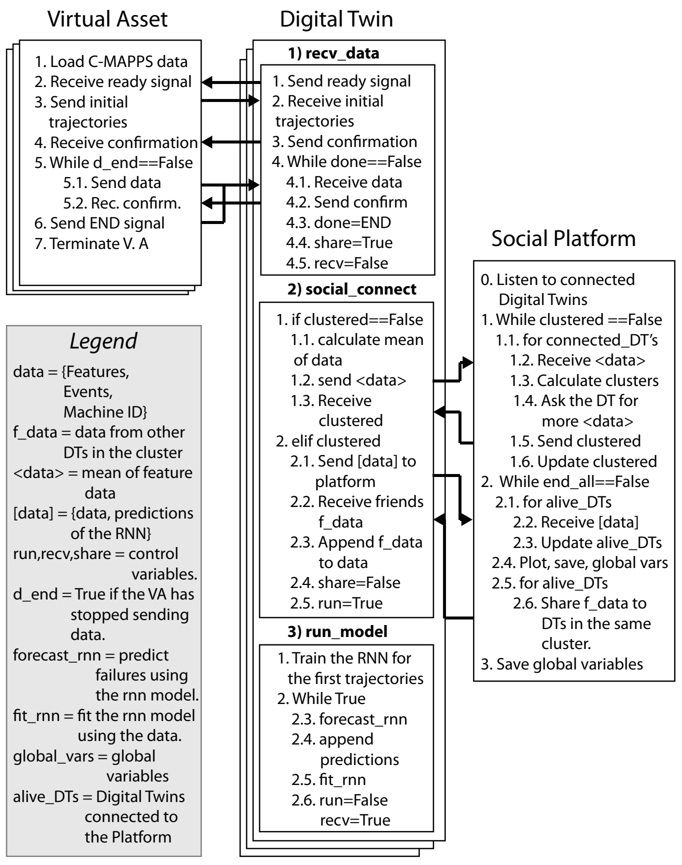


Fig. 3. Sketch of the re-structuring to the C-MAPPS data set in order to represent a subset of different assets enduring recursive failures. Each different background colour represents a different environmental condition. Note how the trajectories contained in FD002 and FD004 are not used due to the varied environmental conditions contained in these data sets, which make it harder to cluster these trajectories to consistently represent different assets. Each square represents a failure trajectory. Filled squares represent a machine failing due to fan degradation and empty squares due to High Pressure Compressor degradation.

B. Real-time collaborative prognostics algorithm

In this section, we present our algorithm written in pseudocode. In practice, the implementation has been done using Python. Concretely, we use Keras, Tensorflow and the wtte-rnn python package [18] for the machine learning implementation. The Socket library to enable communication between the elements of the architecture. A bash script has also been programmed to enable running the presented approach in a single machine or a cluster. For simplicity and control, the tasks performed by the Digital Twin are chosen to be consecutive in order to ensure control over the data flow. These tasks correspond to (1) receiving the data from the Virtual Asset, (2) sharing the data with the Social Platform, and (3) computing the asset's prognostics using the WTTE-RNN algorithm (see Figure 4). In our architecture, (1) and (2) are performed by the output manager of the Digital Twins and (3) is performed by their Analytics Engine (see Figure 1).

Our implementation presents a simplified version of the collaborative learning approach proposed in Section II-B2. Concretely, the trajectories shared among groups of similar assets are not weighted according to their difference in the clustering algorithm, but directly incorporated. In addition, censored trajectories are not included in the training of the Recurrent Neural Network. This has been done to allow for faster computation, as it reduces the number of trajectories included in the training matrix.



```
Virtual Asset
1. Load C-MAPPS data
2. Receive ready signal
3. Send initial
   trajectories
4. Receive confirmation
5. While d_end==False
    5.1. Send data
    5.2. Rec. confirm.
6. Send END signal
7. Terminate V. A
```

```
Digital Twin
1) recv_data
1. Send ready signal
2. Receive initial
   trajectories
3. Send confirmation
4. While done==False
   4.1. Receive data
   4.2. Send confirm
   4.3. done=END
   4.4. share=True
   4.5. recv=False
2) social_connect
1. if clustered==False
   1.1. calculate mean
        of data
   1.2. send <data>
   1.3. Receive
        clustered
2. elif clustered
   2.1. Send [data] to
        platform
   2.2. Receive friends
        f_data
   2.3. Append f_data
        to data
   2.4. share=False
   2.5. run=True
3) run_model
1. Train the RNN for
   the first trajectories
2. While True
   2.3. forecast_rnn
   2.4. append
        predictions
   2.5. fit_rnn
   2.6. run=False
        recv=True
```

```
Social Platform
0. Listen to connected
   Digital Twins
1. While clustered ==False
   1.1. for connected_DT's
      1.2. Receive <data>
      1.3. Calculate clusters
      1.4. Ask the DT for
           more <data>
      1.5. Send clustered
      1.6. Update clustered
2.  While end_all==False
   2.1. for alive_DTs
      2.2. Receive [data]
      2.3. Update alive_DTs
   2.4. Plot, save, global vars
   2.5. for alive_DTs
      2.6. Share f_data to
           DTs in the same
           cluster.
3. Save global variables
```

Legend	
data =	{Features, Events, Machine ID}
f_data =	data from other DTs in the cluster
<data> =	mean of feature data
[data] =	{data, predictions of the RNN}
run,recv,share =	control variables.
d_end =	True if the VA has stopped sending data.
forecast_rnn =	predict failures using the rnn model.
fit_rnn =	fit the rnn model using the data.
global_vars =	global variables
alive_DTs =	Digital Twins connected to the Platform

Fig. 4. Sketch of the system's algorithms, including the major tasks performed by each block of the architecture (see Figure 1). Black arrows are used to signify communication using the socket library. Both the Virtual Assets and the Digital Twins are drawn over stacked rectangles to signify their multiplicity in the architecture.

C. Results and discussion

We run the algorithm shown in Figure 4 with the following initial parameters: 5 initial trajectories, a system time period of $T_T = 50$ and a total of 4 Virtual Assets, 2 formed from data extracted from the FD001 data set and 2 from the FD003 data set. The Recurrent Neural Network has a LSTM architecture of $26 \times 24 \times 10 \times 2$ being the layer with 24 neurons the only recurrent layer. We use tanh as activation function for all layers except for the last two-neurons custom output layer. This last layer was proposed in [15] and converted the output of the last two neurons into α and β. No Dropout, or regularisation are used.

The Recurrent Neural Network is trained for 50 epochs (a value chosen due to computation constraints (see Sec. III-D). The Social Platform's clustering algorithm is set to separate the Virtual Assets in two clusters. Cluster convergence is defined as the time when Virtual Asset's have been determined to belong to the same cluster for a total of three consecutive time-steps. Cluster convergence is observed relatively fast, typically upon completion of six time-steps in any of the Virtual Assets. In order to asses the accuracy of our prognostics algorithm, we use the same score as suggested during the PHM08 Prognostics Data Challenge, when this data set was introduced, but we normalise to the amount of predictions in the test dataset (N_p).

$$S = \frac{1}{N_\text{P}} \sum_i^I \left[\theta(y_i - \hat{y}_i)(e^{\frac{y_i - \hat{y}_i}{13}} - 1) + \theta(\hat{y}_i - y_i)(e^{\frac{\hat{y}_i - y_i}{10}} - 1) \right], \tag{8}$$

Where $\sum_i$ indicates summation over test samples (there is a single prediction for each sample). θ is the heaviside step function. $\hat{y}_i$ is the predicted time to failure and $\hat{y}$ is the real time to failure. This score penalises late predictions as corrective repair is assumed to be less benecial than preventive repair.

Initial results point at collaborative learning outperforming learning from the rest of the assets in the fleet (see Figure 5). In order to calculate the score in a real time environment, we use the predicted time to failure point by point. This means that predictions very far from failure are incorporated into the score calculation in a much larger extent than in the case of the PHM08 Prognostics Data Challenge, where a single prediction was obtained for every censored trajectory to failure. In Figure 5 we see how the difference between collaborative learning and learning from all the assets in the fleet becomes increasingly noticeable as the simulation moves forward (higher step values). In both cases the values of the score S increase with the Step of the simulation because the likelihood of encountering larger time to failure trajectories (and thereby larger values of S) also increases. Figure 6 shows an example of the real-time output of the presented system for a particular asset in the asset fleet, showing the real time to failure (red) together with predictions for the collaborative case (green) and the non-collaborative case (blue).

The difference between collaborative an non-collaborative approaches observed in the initial experiment presented in this

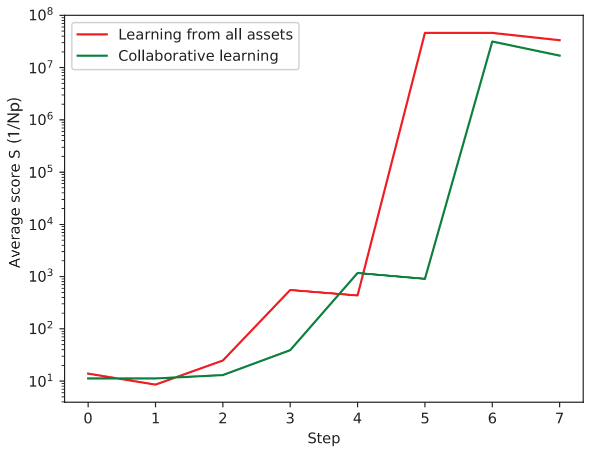


Fig. 5. Score according to Eq. (8). The green colour indicates the score when assets are collaborating with eachother, and the red color shows the score for the case when there is no collaboration. Lower values of the score correspond to more accurate predictions. In this case, the horizontal axis (Step) corresponds to a total simulation time of $Step * T_T$ (in this case $T_T = 50$).

paper must be taken with caution due to the effect of the small sample size: the number of recurrent events treated in each agent is low (from 5 to 40 trajectories), and it is likely that the samples used to train the Recurrent Neural Network are not representative of the population. This would mean that extending the sample size to dissimilar assets, like in the case of learning from all assets, might increase the accuracy of the predictions in an early stage of the assets life cycle [19]. Therefore, the non-collaborative score shown in Figure 6 (red) is aided by these effects, and in a case where enough trajectories are available for each asset, or where the number of assets in the fleet is larger, collaboration is expected to outperform non-collaboration by larger differences.

D. On real-time computational constraints

Despite recent advances in computational efficiency, training recurrent neural networks is still known to require a significant amount of processing time and power consumption. Empowering industrial assets with agents incorporating real-time deep-learning computing capabilities is then likely to be met with reasonable scepticism. In this section, we present some basic estimates of the computation time required by our approach and demonstrate its feasibility to run in real time for a realistic industrial scenario. We base our estimates on a Recurrent Neural Network running on a Intel Core i5-760 2.8GHz Quad-Core (that is, without the increased performance expected from a GPU processor). A relatively modest processor, priced at the range of \$150 for private customers.

In the C-MAPPS data set the units of time are cycles, without a direct conversion to standard time measures. We take a conservative approach by assuming that the industrial agent should be able to train a Recurrent Neural Network considering the whole dataset (FD001 and FD003), and predict

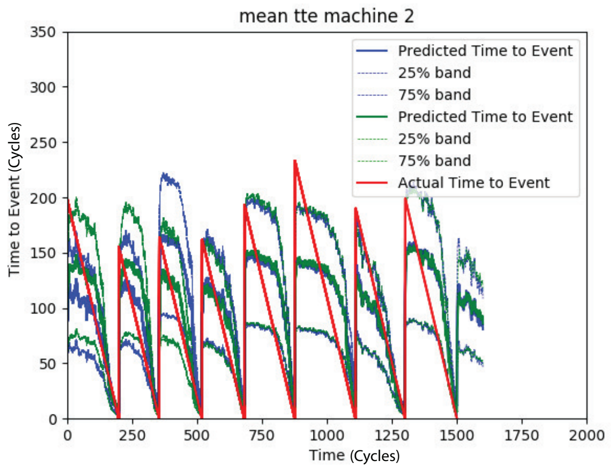


Fig. 6. Plot showing the typical output of the presented system for one of the assets in the asset fleet. The green lines show the predictions for the collaborative case, and the blue lines show the case of learning from all other assets in the fleet. The quartiles of the Weibull distribution are also shown in the 25 and 75 % bands. The actual time to event is shown as a descending line. This particular example was stopped in the middle of a trajectory (around $t = 1600$) to showcase the real-time nature of the presented system.

future failures in less than 30 minutes with reasonable accuracy. We argue that re-evaluating predictions every 30 minutes is more than sufficient for most real-life prognostics scenarios, where predictive maintenance is typically scheduled with at least several hours in advance. In Figure 7 we observe this to be the case, with the WTTE-RNN algorithm producing a score of less than 420000, which corresponds to a Pearson coefficient of 0.912 between predicted and true failures

In Figure 7, we show how computational times scale well with the decrease of the log-likelihood for up to 600 epochs. The difference between this value and the number of epochs run in the Digital Twins (50) presented in the results section III-C is due to our need to 1) run several Digital Twins within the same processor, and 2) re-train the Recurrent Neural Network in each twin several times. Therefore, we decided to compromise in the number of training epochs as our main aim was to demonstrate distributed collaborative learning for prognostics, and not so much to demonstrate the accuracy of the WTTE-RNN algorithm.

Compared with training the agent's Recurrent Neural Network, the other time constraints of the system (such as communication, data extraction, platform operation) have a vanishing contribution within a real industrial scenario.

IV. Conclusions and Future Work

We present the first steps towards the implementation of distributed real-time collaborative prognostics. In order to do so, we assign each asset in the asset fleet a Digital Twin, endowed with an Analytics Engine. Using their Analytics Engine the Digital Twins estimate their corresponding Asset's Time To Failure by means of a Recurrent Neural Network algorithm. Collaboration is achieved by sharing feature and

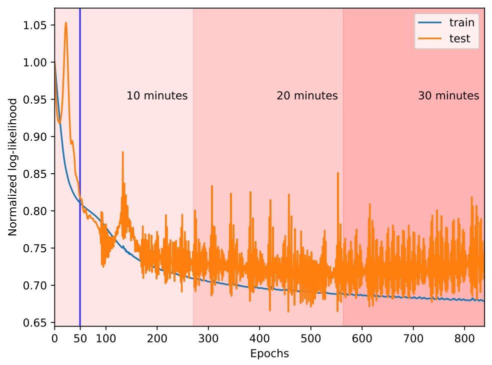


Fig. 7. Plot showing normalized training and test log-likelihood for the case of the whole dataset with respect to the number of epochs used to train the Recurrent Neural Network. The background shades of red show the timescales associated, and the blue line shows the number of epochs chosed for the example presented in this paper.

event data between pairs of Digital Twins and weighting this data in the training of the prognostics algorithm. Such an approach is specially suitable in fleets formed by assets with recursive events, as these recursive events can be used to refine each asset's prognostics algorithms.

This paper demonstrates a basic implementation of real-time distributed collaborative learning, with collaboration limited to sharing trajectories to failure in real time among clusters of similar assets. Initial findings show that the presented approach outperforms learning from the whole asset fleet. However, these findings have to be refined by increasing the number of experiments, assets in the asset fleet and epochs used in the training of the recurrent neural networks. The presented results should thus be considered as an initial proof of concept.

The C-MAPPS data set used in this paper includes only trajectories to failure obtained from identical machines. We try to avoid this pitfall by generating our own virtual machines from subsets of trajectories known to experience different failures and. Testing the proposed approach with a data set where the differences between assets are defined clearly will form the future work of the presented research.

References

[1] "Killer whale: Hebridean Whale and Dolphin Trust." [Online]. Available: https://hwdt.org/killer-whale/

[2] I. Mezei, V. Malbasa, and I. Stojmenovic, "Robot to robot," *IEEE Robotics and Automation Magazine*, vol. 17, no. 4, pp. 63–69, 2010.

[3] A. Brintrup, D. McFarlane, D. Ranasinghe, T. Sánchez López, and K. Owens, "Will intelligent assets take off? Toward self-serving aircraft," *IEEE Intelligent Systems*, vol. 26, no. 3, pp. 66–75, 2011.

[4] K. Tuyls and G. Weiss, "Multiagent learning: Basics, challenges, and prospects," *AI Magazine*, vol. 33, no. 3, pp. 41–52, 2012.

[5] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of the 1995 IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995.

[6] Y. Del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, and R. G. Harley, "Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 2, pp. 171–195, 2008.

[7] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471, 2007.

[8] M. Dorigo, G. D. Caro, and L. M. Gambardella, “Ant Algorithms for Discrete Optimization,” *Artificial Life*, vol. 5, no. 2, pp. 137–172, 1999.

[9] P. Leitão and S. Karnouskos, *Industrial Agents Emerging Applications of Software Agents in Industry*, M. Kaufmann, Ed., 2015.

[10] J. Barbosa, P. Leitão, E. Adam, and D. Trentesaux, “Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution,” *Computers in Industry*, vol. 66, pp. 99–111, 2015.

[11] H. B. Mcmahan and D. Ramage, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” vol. 54, 2017.

[12] N. D. Glenn, *Cohort Analysis*, 2nd ed. Thousand Oaks, California: Sage Publications, Inc., 2005.

[13] T. Wang, J. Yu, D. Siegel, and J. Lee, “A similarity-based prognostics approach for remaining useful life estimation of engineered systems,” *2008 International Conference on Prognostics and Health Management, PHM 2008*, no. November, 2008.

[14] K. Bakliwal and M. Harshadbhai, “A Multi Agent System architecture to implement Collaborative Learning for social industrial assets,” in *INCOM 2018*, 2017.

[15] E. Martinsson, “WTTE-RNN : Weibull Time To Event Recurrent Neural Network,” Ph.D. dissertation, Chalmers University Of Technology, 2016.

[16] E. R. Lapira, “Fault detection in a network of similar machines using clustering approach,” *Journal of Chemical Information and Modeling*, vol. 53, no. 9, pp. 1689–1699, 2013.

[17] A. Saxena, K. Goebel, D. Simon, and N. Eklund, “Damage propagation modeling for aircraft engine run-to-failure simulation,” in *2008 International Conference on Prognostics and Health Management, PHM 2008*, 2008.

[18] “Wtte-rnn.” [Online]. Available: https://github.com/ragulpr/wtte-rnn

[19] A. S. Palau, Z. Liang, D. Lütgehetmann, and A. K. Parlikad, “Collaborative prognostics in Social Asset Networks,” *Future Generation Computer Systems*, 2018.