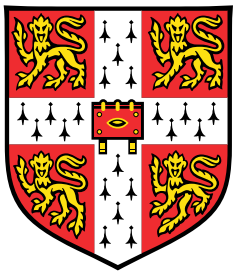
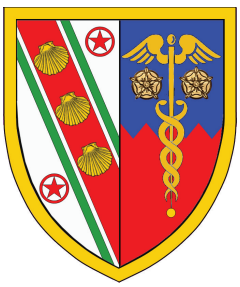


Distributed Collaborative Prognostics

PhD Thesis



Adrià Salvador Palau

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Darwin College

October 2019

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Adrià Salvador Palau
October 2019

Abstract

Distributed Collaborative Prognostics

Adrià Salvador Palau

Managing large fleets of machines in a cost-effective way is becoming more important as corporations own increasingly large amounts of assets. The steady improvement in cost and reliability of sensors, processors and communication devices has helped the spread of a new paradigm: the Internet of Things. This paradigm allows for real-time monitoring of countless physical objects, obtaining data that can be fed to machine learning algorithms to predict their future state and take managerial decisions.

Despite rapid technological change, industries have been slow to react, and it has been only recently that many have transitioned towards a new business model: servitisation. Servitisation is based on selling the services that assets provide, instead of the assets themselves. Although more companies are adopting this business model, there is a lack of solutions aimed to maximise its economic value. This thesis presents one such solution capable of predicting failures in real time, thus reducing a crucial cost contribution to asset ownership: unexpected failures. This new approach, Distributed Collaborative Prognostics, consists of providing each machine with its own particular agent, that enables it to communicate with other similar machines in order to improve its failure predictions.

This thesis implements Distributed Collaborative Prognostics in three different scenarios: (i) using a multi-agent simulation framework, (ii) using synthetic data from a well-established prognostics data set, and (iii) using real data from a fleet of industrial gas turbines. Each of these scenarios is used to study different elements of the prognostics problem. Multi-agent simulations allow for the calculation of the cost of predictive maintenance coupled with Distributed Collaborative Prognostics, and for the estimation of the cost of agent failures in different architectures. Synthetic data is used as a test bench and to study assets operating in dynamic situations. Real industrial data from the Siemens industrial gas turbine fleet serves to test the applicability of the tool in a real scenario.

This thesis concludes that Distributed Collaborative Prognostics is the adequate solution for large and heterogeneous fleets of assets operating dynamically. Its cost effectiveness depends on the value of the assets; in general, highly-valued assets are more conducive to Distributed Collaborative Prognostics, as the savings from improved failure predictions compensate the cost of enabling them with Internet of Things technologies.

I would like to dedicate this thesis first and foremost to my family, that has remained an unshakeable vessel across the events of time, and that has kept supporting me both financially and morally whatever the circumstances. You have given me the privilege of being able to live my life however I wanted, with nearly absolute freedom. Gràcies per tot!

Secondly, to Ashley Braunthal for her relentless commitment, her patience, love, optimism, and passion. You truly are a "Llir entre cards". Discovering New York together was an immense pleasure.

To Jon Roozenbeek, and Alena Giesche for their brilliance, humanity, and non-fading optimism during the events that have moved us all. Hearing you laugh and sing is truly a deeply satisfying experience. A friendship that will last forever.

To my friends in Barcelona, especially to Jeroni and Ferran for our revitalised friendship and healing trips to Paris, New York, London, Copenhagen and soon Italy!

To Christina, Thomas and Shannon. Since I met you on my first days in Darwin College I have enjoyed every single minute that we have spent together.

To Ajith Kumar Parlikad, for being a great supervisor that truly cares about his job. Flawless!

To Bodil Holst, and the leadership at MBScientific for helping me during the first year of my PhD, and for being great mentors to me. I will always be grateful.

To the folks at the Catalan Society, Open Club for Debate, and Wine and Cheese Society for your bravery and support in difficult times for freedom of speech, enjoyment and dignity.

Without a doubt, I can say that the last three years have been the best of my life. During these years I have met scores of people whom I will never forget. Each stroll to Grantchester a new idea, each pint in DarBar a new conversation to remember. For this, I have not been able to fit all of you here but you know that I hold you in the highest regard.

...

Acknowledgements

This PhD Thesis has been supported by a “la Caixa" Fellowship (ID 100010434), with code LCF/BQ/EU17/11590049.

Table of contents

List of figures **xv**

List of tables **xvii**

1 Introduction **1**
1.1 The problem of asset management 1
1.2 Maintenance and prognostics 3
1.3 A thought experiment on non-ergodicity 4
1.4 The problem of dynamism 7
1.5 Problem statement 8
1.6 Distributed Collaborative Prognostics 9
1.7 Research questions 10
1.8 Theoretical position 11
1.9 Methodology 11
1.10 Thesis outline 13

2 Research background **17**
2.1 Literature review 17
2.1.1 Methodology 18
2.1.2 Asset health and performance management 19
2.1.3 Machine learning for prognostics 23
2.1.4 Distributed systems for prognostics 24
2.2 Theoretical background 28
2.2.1 The prognosis problem 29
2.2.2 Loss functions 30
2.2.3 Neural Networks 34
2.2.4 Back propagation through time 39
2.2.5 Long Short-Term Memory variant 43

2.3 Conclusion 45

3 Distributed Collaborative Prognostics: properties and architectures 47
3.1 Properties 47
3.2 Building blocks 51
3.2.1 Virtual Asset 51
3.2.2 Agents 52
3.3 Architectures 54
3.3.1 Centralised 56
3.3.2 Hierarchical 56
3.3.3 Heterarchical 57
3.3.4 Distributed 58
3.4 Conclusion 59

4 Distributed Collaborative Prognostics with a maintenance policy 61
4.1 Description of the system 62
4.2 A basic collaborative prognosis algorithm 64
4.3 Maintenance policy 65
4.4 Simulation parameters 67
4.5 Results 68
4.5.1 Behaviour after convergence 70
4.5.2 Optimisation of the maintenance policy 70
4.6 Conclusion 71

5 An implementation of real-time Distributed Collaborative Prognostics 73
5.1 Advanced Multi-Agent Systems 74
5.2 Architecture description 74
5.3 Real-time collaborative prognostics 75
5.3.1 Clustering 79
5.4 Design of the experiments 80
5.4.1 Data preparation in the C-MAPSS data set 81
5.4.2 Experiments 82
5.5 Results 85
5.5.1 Distribution 86
5.5.2 Flexibility 86
5.5.3 Scalability 88
5.5.4 Leanness 89

5.5.5 Resilience . . . 90
5.6 Conclusion . . . 91

6 Cost implications of Distributed Collaborative Prognostics 93
6.1 Failures of the building blocks . . . 93
6.2 Cost model . . . 94
6.3 NetLogo implementation . . . 95
6.3.1 Distributed clustering algorithm . . . 96
6.4 Results . . . 97
6.5 Architecture selection . . . 100
6.6 Conclusion . . . 100

7 Case study: the Siemens gas turbine fleet 103
7.1 Description of the fleet . . . 103
7.1.1 A broad-brush description of a gas turbine . . . 104
7.1.2 The STA-RMS platform . . . 105
7.2 Preliminary data analysis . . . 106
7.2.1 Events: statistical analysis . . . 106
7.2.2 Sensors and dimensionality . . . 113
7.3 Data preparation framework . . . 117
7.3.1 Time sampling and trajectory pruning . . . 119
7.4 Non-collaborative prognostics . . . 119
7.4.1 In practice: an additional classification algorithm . . . 123
7.5 Collaborative prognostics . . . 126
7.5.1 Engineering requirements . . . 126
7.5.2 Static collaborative prognostics . . . 127
7.5.3 Real-time Distributed Collaborative Prognostics . . . 130
7.6 Conclusion . . . 132

8 Conclusion and future work 135
8.1 General conclusions . . . 135
8.2 Caveats . . . 137
8.3 Contribution to the academic knowledge . . . 138
8.4 Technologies . . . 139
8.5 Contribution to industrial practice . . . 141
8.6 Future work . . . 142

References **143**

Appendix A Numerical and analytical results **157**
A.1 Introductory example formula 157
A.2 Multi-Agent System simulations: results 158
A.3 Weibull analysis 159
A.4 Jensen-Shannnon divergence results 159

Appendix B Code **163**
B.1 Pseudocode for multi-agent simulations 163
B.2 Implementation code 165
B.2.1 Bespoke bash script 165
B.2.2 Leanness experiment 166
B.2.3 Data preparation for Principal Component Analysis 167
B.2.4 Class balance in the classification algorithm 167

List of figures

1.1 Technological and scale changes in asset management since Roman times. 2
1.2 Error ΔE_{sim}, induced by learning from similar machines instead of self-learning. 7
1.3 Sketch of Distributed Collaborative Prognostics. 10
1.4 Diagram showing all the phases of the methodology used in this thesis. . . . 11
1.5 Block diagram showing the initial exploratory phase of this PhD Thesis. . . 12
1.6 Diagram of the implementation and validation phases of this thesis. 12

2.1 Venn diagram of the intersection of the three fields reviewed in this thesis. . 17
2.2 Sketch showing the data matrix fed to the machine learning algorithm . . . 29
2.3 Diagram of a three-layer Neural Network. 35
2.4 Diagram of a Recurrent Neural Network and its weight matrices. 40
2.5 Diagram of a LSTM Neural Network and its weight matrices. 44

3.1 Block diagram of a Virtual Asset. 52
3.2 Block diagram of a Digital Twin. 53
3.3 Block diagram of a Mediator Agent. 53
3.4 Block diagram of the Social Platform. 54
3.5 Sketch of architecture layers and building blocks. 55
3.6 Block diagram of the Centralised architecture. 56
3.7 Block diagram of the Hierarchical architecture. 57
3.8 Block diagram of the Heterarchical architecture. 58
3.9 Block diagram of the Distributed architecture. 58

4.1 Diagram of the Distributed architecture implemented in NetLogo. 62
4.2 Diagram of the communications among agents in this implementation. . . . 63
4.3 Cost of the system before and after convergence with collaborative learning. 69
4.4 Cost per unit time, K for different values of N_i, η and σ. 70
4.5 Optimal (minimum cost) η in theory, compared with simulation. 71

5.1 Sketch of the architecture used in the first section of Chapter 5. 75
5.2 Training data matrix fed to the Recurrent Neural Network. 76
5.3 UML diagram of the Multi-Agent System. 77
5.4 Sketch of the Multi-Agent System, and its data structures. 78
5.5 A set of trajectories to failure extracted from the FD003 section of the data set. 82
5.6 Different operational regimes in the C-MAPSS dataset 83
5.7 Sketch of the re-structuring to the C-MAPSS and PHM08 data sets. 85
5.8 Percentile difference in accuracy between random and collaborative learning. 86
5.9 Experimental scenario 1: varying failure. 87
5.10 Experimental scenario 2: varying operational setting. 88
5.11 Experimental scenario 3: scalability. 89
5.12 Experimental scenario 4: flat reading sensors and outliers. 90
5.13 Experimental scenario 4: drift. 91

6.1 Normalised cost for every architecture with respect to the communication cost. 98
6.2 Normalised index of dispersion with respect to the noise in the system. . . . 99

7.1 Diagram of a Siemens SGT-400 twin-shaft gas turbine 105
7.2 Histogram of the time between events for two events in the fleet 107
7.3 Plot of the Weibull distributions fitted to different events of interest. 108
7.4 Sketch of subsampling of data for each event type. 109
7.5 Subsampling of data for ‘Running Trips’ in the turbine 1A. 110
7.6 Subsampling of data for two different events in turbine 4A. 110
7.7 Histograms in the turbine 2G for the event “Interduct Thermocouple Fault” 112
7.8 Pearson correlation coefficients for sensors with over 90% of uncorrupt data. 114
7.9 Pearson correlation coefficients for sensors with over 90% of uncorrupt data. 115
7.10 Summary of the data preparation framework used in this thesis. 117
7.11 Prediction of the LSTM Neural Network for Running Trip failures. 121
7.12 Plot of a termocouple in the combustion system before failure 122
7.13 Prediction of the LSTM Neural Network for machines very close to failure . 122
7.14 LSTM-predictions for Compressor Exit Thermocouple Deviation failure. . 123
7.15 Confusion matrices for the trained classifiers. 125
7.16 Sketch of the architecture used in the first section of Chapter 5. 127
7.17 Distributed Collaborative Prognostics for Running Trips in Siemens 131
7.18 Clustering results in the Siemens implementation. 131

List of tables

2.1 Terms used to search writings for the literature review of this thesis. 18

3.1 Table summarising the Multi-Agent System building blocks 55

7.1 Table showing the highest values for the Jensen-Shannon divergence. 111
7.2 Table showing minimum values for the Jensen-Shannon divergence. 111
7.3 Accuracies for centralised prognostics 123
7.4 Accuracies for 10-fold validation of the different classification algorithms . 125
7.5 List of events of interest and machines for which they are present. 128
7.6 Collaborative prognostics compared to centralised prognostics for Siemens 129

A.1 Cost contributions for no agent failure featuring quadruplets N_{P},N_{C},N_{Co},N_{pro}. 158
A.2 Cost contributions for agent failure featuring quadruplets N_{P},N_{C},N_{Co},N_{pro}. . 158
A.3 Clustering purity results at t=400 for the case of no agent failure. 159
A.4 Clustering purity results at t=400 for the case of agent failure. 159
A.5 Values of α and β for the Weibull fits in each event in the dataset 160
A.6 Jensen-Shannon divergence values for SGT-100 gas turbines 161
A.7 Jensen-Shannon divergence values for SGT-200 gas turbines 161
A.8 Jensen-Shannon divergence values for SGT-300 gas turbines 162
A.9 Jensen-Shannon divergence values for SGT-400 gas turbines 162

Chapter 1

Introduction

1.1 The problem of asset management

The problem of managing assets has been dealt with by administrators since the establishment of the first civilisations, that led to the emergence of the concept of public good[1]. The study of asset management goes back to the supervision of Roman road networks by the figure of the *curatore viarium* [2, 3]. This administrator position (unique to Italy, as in the provinces this role was performed by other civil servants) was created in response to the bad state of Roman roads in the second century BC, caused by the absence of authorities responsible for maintenance outside of the *pomeriums*[2]. To fill up this vacuum, each curatore was assigned part of the road system, in most cases only one road [3]. The effectiveness of these specific measures is disputed [3], but it is clear that good road management was crucial for the longevity and expansion of the Roman Empire. The vast network of Roman roads was used to effectively transport troops and goods from one point to another, and allowed for fast communication between the provinces and Rome [4].

Since then, the core objective of asset management has essentially remained the same: to maximise the value produced by assets during their whole life. In the Roman Empire, managing a road meant to keep it in a state good enough so that it did not hinder transportation, while making sure that the amount invested on its maintenance was justified by the value that it generated. Today, asset management is done similarly, but the definition and practice for whole-life value maximisation has been further standardised [1].

[1] In this thesis, asset refers to a *physical* "item, thing or entity that has potential or actual value to an organization" [1]. Note how the word *physical* has been added to the standard definition of the word to differentiate it from a broader category, in which assets can be patents, financial products and other entities.

[2] A maximum of one mile from the boundary of the city.

While the basic objective of asset management has not changed in nearly two thousand years, there have been two radical transformations: technology, and scale. The digital revolution of the last century has given place to the paradigm of the Internet of Things, in which physical assets are continuously monitored by sensors, their state being accessible at any given time through the internet. This shift has been paired with a substantial increase on the number of assets managed by corporations, leading to economies of scale that did not exist in the past (see Fig. 1.1).

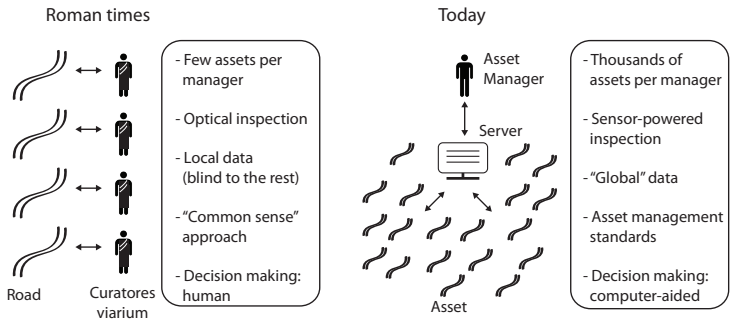


Fig. 1.1 Technological and scale changes in asset management since Roman times.

At the same time, increasing attention to environmental effects and decreasing operational margins have placed asset management at the core of many manufacturing corporations' business strategies. In the hyper-competitive playing field of today, optimising the whole-life value of a company's assets often makes the difference between running a profit or a loss.

This focus on extracting all the possible value from industrial assets has also brought forward some negative consequences. Increasing the whole-life value of an asset means to extend its operational life and decrease its need for replacements, thus lessening the revenue of the company dedicated to produce and maintain it. In manufacturing *lingua* these companies are normally called original equipment manufacturers (OEM) [5]. To adapt to this, OEM's and other industries have developed a new business model: Servitisation. Servitisation is based on selling the services that assets provide, instead of the assets themselves. In this way, the company that manufactures the asset profits from optimising its whole-life value, and the company that uses it can concentrate on its core business [6]. Servitisation is thus the opposite of programmed obsolescence. Instead of increasing sales by designing products that will fail soon, companies must now design products that last longer, as they draw profit from each hour that their products are operating without failure.

When optimising the value of individual assets through their whole lifespan, a key issue is to properly account for all the costs of their different life-phases. For a typical asset these are deployment, operation, maintenance, improvement and removal [1]. The optimisation

of whole-life asset management consists in maximising a value function in which the costs associated to these phases are properly accounted. In many real industrial scenarios, this reduces to the optimisation of the operation and maintenance phases, which involve the biggest part of the asset's lifespan and combined have the biggest effect on the overall cost. This thesis focuses on the maintenance phase, more concretely on improving failure prediction, also known as prognostics. By providing information about the future state of an asset, prognostics can be used to reduce maintenance cost and maximise operational time.

The following section discusses how prognostics can be leveraged in the optimisation of maintenance policies.

1.2 Maintenance and prognostics

Assets deteriorate over time both in their intrinsic operational capabilities and in their competitiveness relative to newly designed assets. With the absence of maintenance, most assets reach a deterioration level known as functional failure, which can be defined as the asset losing its operation capabilities [7]. Maintenance is any action that modifies the state of an asset and is aimed to prolong its operational life.

Maintenance policies are determined by how much is known about the state of the assets. In case of limited information, the simplest strategy is to correctively maintain the assets once they are found to have failed. This corresponds to the case in which the asset manager is in charge of a fleet so small that it cannot provide sufficient statistics (for example, in Roman times each *curator* managing only a few roads).

For larger asset fleets, one can start by calculating basic statistics about asset failure such as the mean time of failure, its variance, etc. This knowledge allows for the implementation of cost-effective preventive maintenance strategies: the assets are maintained *before* failure according to a pre-set policy, which is optimised according to such statistics [8].

With enough data, one can move from computing the mean and variance of the failure times of the assets in the fleet to estimating the failure time probability density function: $f(t)$. Distributional assumptions can be used to apply survival analysis to preventive maintenance planning, allowing for more nuanced maintenance policies and a quantification of the instantaneous probability of failure. Moreover, from this probability density function one can obtain the probability density function conditional to the knowledge that the machine has already survived a time t_{surv}, $f_s(t|t_{\text{surv}})$ [9]:

$$f_s(t|t_{\text{surv}}) = \frac{f(t+t_{\text{surv}})}{1-\int_0^{t_{\text{surv}}} f(s)ds}. \tag{1.1}$$

Methods based on distributional assumptions and survival analysis are among the most widespread techniques in maintenance planning as they rely on straightforward calculations that only require the failure times of the machines as input.

Since the spread of condition monitoring, asset managers have gained access to features that can be used to predict machine failures. These can be sensor values, usage statistics, etc. Thanks to this new data, it becomes possible to find a function that from the asset's sensor time-series ($x_{0:t}$) finds the probability per unit time that a machine will fail in a time T. For a given machine i, this function can be written as $f_i(T, x_{0:t})$. This possibility has given rise to a new way of understanding maintenance known as e-maintenance, that among other goals aims to integrate real-time prognostics and maintenance recommendations [10].

The methods to estimate the time at which a particular machine will fail fall into two broad categories: physical methods, reliant on understating the fundamental process behind a failure, and data-driven methods, based on statistical inference from the history of previous failures [11, 12]. Data-driven methods, which this thesis relies on, have been traditionally based on centralised architectures in which all the available data from a fleet of assets is used at once to infer a prognostics model. This practice is based on the assumption that the assets in the fleet are similar enough so that the knowledge learned from one asset can readily transfer to another asset. As will be shown in this thesis, this is often not true, and the variations in make, operational history, repair history, etc. mean that industrial asset fleets are non-ergodic: the information collected from an asset doesn't always apply to the rest.

The next section presents a thought experiment on non-ergodicity that shows the potential benefits of a solution in which assets can infer their failure times using information obtained from other similar assets.

1.3 A thought experiment on non-ergodicity

In this section, a simple thought experiment is presented to illustrate a theoretical scenario in which the consequences of collaborative learning are discussed. In the context of this experiment, collaborative learning can be defined as the ability of assets to use information from other assets to update their own failure predictions. Inter-asset differences may cause a predictive model, relevant to a particular asset, to not apply to another asset in the same fleet. For example, in a fleet of cars, the year of production, car model, and driving history will have an effect on the properties and behaviour of each car. This illustrates an effect encountered frequently when performing prognostics in an asset fleet: lack of ergodicity.

A system is ergodic if for a controlled experiment, studying a given process in several experiments running in parallel corresponds to studying the same process over time in one

experiment [13]. In other words, the mean of the sample of experiments converges to the expected value over time of a single experiment. In industrial asset fleets these properties are not fulfilled. If a machine component fails several times (for example, repeated punctures on a bicycle's tire), the mean properties of these failures do not necessarily converge to the mean properties of all the failures of that component in the fleet [14]. This means that asset fleets are often *non-ergodic*.

The lack of ergodicity is an inevitable problem in real scenarios that is usually combated by using very large data sets in which each possible example is represented, and in which a machine learning algorithm is able to automatically discern between each different case. In prognostics, however, such data is rarely available (because all efforts are put into reducing the number of failures). Thus, more ad hoc approaches must be followed.

In this thesis, I propose to vary the amount and origin of the data used to train machine learning algorithms depending on a measure of similarity between assets that is updated in real-time. In non-ergodic fleets, the suitability of using information from other assets to perform prognostics for a given asset will depend on two factors: (1) how much do we already know about this asset, and (2) how different is this asset from the assets that it will learn from. If enough is known about the asset, using data from other assets may only reduce prediction accuracy. However, if we still know too little about it, using fleet statistics to improve its prediction models will be beneficial. This is illustrated by the following thought experiment.

Assume a set of machines whose failures can be predicted by fitting a descending line to data from a single sensor. When the line crosses the x axis, the machine is predicted to fail, as failure is diagnosed if a sensor value is found to be equal or smaller than 0. The machines are assumed to be *similar*, and their similarity to be determined by the difference in the parameters determining their deterioration lines, that is the slope and the intercept[3]. The task is to determine the useful life time of a particular machine (the total operational time until failure)[4].

A simple way to do so is to assume that at time t, the predicted useful life time of a particular machine corresponds to the mean of the recorded useful life times of a subset of other similar machines (how long on average have these machines lasted). The error incurred would be:

$$E_{\text{sim}} = |T_R - \bar{T}_{sim}|. \tag{1.2}$$

T_R is the real useful life time of the machine and $\bar{T}_{sim}$ is the mean of the useful life times of the subset of other machines.

[3] Assume that machine properties are only determined by their deterioration process.

[4] Note: the remaining useful time is the useful time minus the time that the machine has been without failing.

Instead, we could fit a linear equation to the data for this particular machine that we have available and calculate the time at which the line crosses the x axis. Let's make the assumption that the sensor values are measured with Gaussian uncertainty. The probability distribution of the fitted useful life time parameter, $\bar{T}_{\text{self}}$ is a Gaussian with a standard deviation $\sigma_{\bar{T}_{\text{self}}}$. Its standard deviation decreases with the number of data points N as:

$$\sigma_{\bar{T}_{\text{self}}} \propto \frac{\sigma}{\sqrt{N}} \propto \frac{\sigma}{\sqrt{t}}. \tag{1.3}$$

Where σ is the uncertainty of each sensor measurement[5]. The mean error difference of prioritising learning from similar machines over self-learning is the mean of the difference between the collective error, E_{sim}, and the self-learning error, E_{self}. If the mean of the difference is positive, self learning will be the best strategy. If not, the optimal approach will be to learn from similar machines.

$$\Delta E_{\text{sim}} = \overline{E_{\text{sim}} - E_{\text{self}}} = E_{\text{sim}} - \overline{E}_{\text{self}}. \tag{1.4}$$

It is safe to assume that recorded failure times can be measured with vanishing uncertainty. Thus, the mean of the error obtained by learning from similar machines is assumed to be constant $\overline{E_{\text{sim}}} = E_{\text{sim}}$. The mean of the self-learning error is calculated as follows:

$$\bar{E}_{\text{self}} = 2\int_0^\infty \frac{x}{\sqrt{2\pi\sigma_{\bar{T}}^2}} e^{-\frac{x^2}{2\sigma_{\bar{T}}^2}} dx = \frac{2\sigma}{\sqrt{2\pi t}}. \tag{1.5}$$

Here, the mean of the measurements of T_{self}, $\bar{T}_{\text{self}}$ is a random variable with a probability distribution with a standard deviation given by eq. (1.3). By setting ΔE_{sim} to 0 one obtains the point at which self-learning carries the same error than learning from similar machines. From this simple thought experiment, one sees that learning from similar machines will be beneficial for the first $t = \frac{2\sigma^2}{\pi|T_R - \bar{T}_{sim}|^2}$ units of time, as self-learning will carry more error than learning from similar machines (see Fig. 1.2).

Unfortunately, self-learning requires an amount of time and data that is not always available in real industrial systems. Instead, one must often use data from the rest of the asset fleet to compensate for this scarcity. Note that if the real useful life of the machine T_{R} is equal (or very close) to the recorded mean of useful lives of similar machines, then $t \to \infty$, and self-learning becomes redundant. Large asset fleets are expected to be formed by subsets of assets with sufficient similarity between each other so that their data can be leveraged

[5] In reality, the uncertainty of the x-intercept using a least-square method stems from a more complicated relation that simplifies to this classic formula under certain conditions (see Appendix A.1).

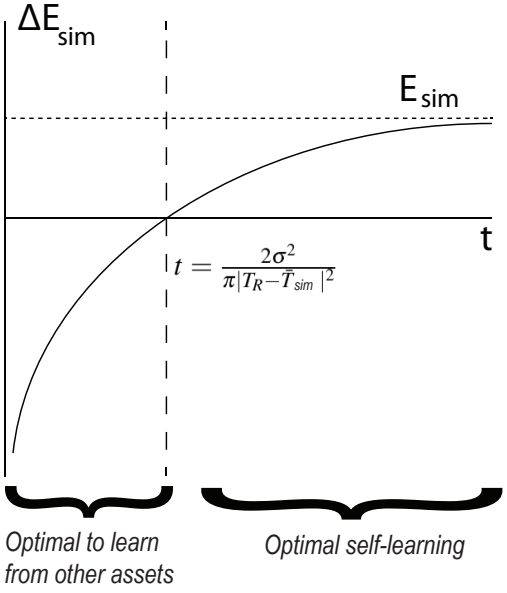


Fig. 1.2 Error ΔE_{sim}, induced by learning from similar machines instead of self-learning. With no self-knowledge (t=0), using the statistics obtained from a similar population means an asymptotically large error reduction. As t increases, $\bar{E}_{\text{self}}$ decreases until a point when using self-learning has a smaller associated average error than E_{sim}.

into prognostics without a large E_{sim}. Properly determining these subsets is key to reduce the error associated with learning from similar assets, E_{sim}. In this example, a well-defined subset of similar assets is one that minimises $|T_R - \bar{T}_{sim}|$, and therefore maximises the time for which learning from similar assets outperforms self-learning.

1.4 The problem of dynamism

The previous section showed that a proper assessment of which assets' data may be relevant to other assets is important to control for the system's lack of ergodicity. In principle, this can be done by assessing the similarity between assets in a fleet. However, an additional property of industrial asset fleets complicates this task: real industrial systems are not only non-ergodic, but they are also dynamic. This means that asset and environmental (external) properties change continuously. In order to react to this dynamism, one needs to make sure that the system is capable of adapting in real time, as it is very difficult to predict *a priori* all the scenarios that an asset will face during its operational life.

An example of the importance of dynamism in asset fleets can be found in the effect of environmental conditions in aeroplane turbines [15]. The same asset (turbine) may move from

operating in a highly corrosive environment (like a marine atmosphere) to a less corrosive one (like an inland region) [16]. It is important for the system to be able to react dynamically to these changes so that the turbine's predictive models incorporate data from similar turbines that had operated previously in the new environment. Traditional centralised prognostics approaches fail to do so, as they do not update their models in real-time.

1.5 Problem statement

This thesis focuses on solving the prognosis problem in real asset fleets. As discussed in the previous two sections, this means devising a prognostics tool capable of operating in the conditions of non-ergodicity and dynamism.

Problem Statement. To devise and implement a prognostics tool able to operate in the conditions of ergodicity and dynamism typical of industrial asset fleets.

A solution to this problem has been postulated to be the deployment of autonomous, asset-specific pieces of software able to provide individualised prognostics (see, for example, [17, 18]). The ideal properties of these pieces of software have been proposed simultaneously in several fields. Thus, they receive different names: holons, industrial agents, digital twins, etc. These pieces of software always form a distributed system, that provides an overall representation of the asset fleet[6]. Distributed systems, due to their flexibility and real-time capabilities should be capable of operating in the conditions of real industrial asset fleets, and thus this thesis' problem reduces to the task of developing a distributed system capable of successfully performing prognostics.

Prognostics so far has been largely based in a centralised approach, in which all the available data from the assets being operated is leveraged into a single prognostics model that is then used to predict failures. Centralised prognostics solutions have several drawbacks: they are not designed to adapt in real time to changes in the asset fleet, suffer from scalability problems due to the significant increase of the size of data-sets [21], and are not resilient to failure as the whole system depends on a single software component.

Successful implementations of distributed systems for asset management tasks exist, but are largely limited to diagnostics and e-maintenance [10]. There have been some attempts to implement distributed systems for real-time prognostics, but their components lack the properties of autonomy and communication required in agent theory [22]. This means

[6]Here Andrea Omicini's definition of a distributed system is used: "A collection of autonomous computational entities conceived as a single coherent system by its designer" [19]. Note how this differs from the stricter definition by Tanenbaum and Steen that implies that the system must be perceived as a single entity [20].

that existing solutions are still in an embryonic stage; either they are presented purely as theoretical constructs, as applications in scenarios comprehending few machines, or as solutions in which their components are not yet able to communicate and collaborate with each other. As a result of this, key properties of a successful industrial solution such as resilience to component failure, or scalability are yet to be demonstrated (see [23] and Chapter 2 for an in-depth discussion on the state-of-the-art).

There are two principal reasons why research on distributed prognostics solutions has been stalled: first, their cost and difficulty of implementation has limited their spread to scenarios featuring very high-value assets. These assets seldom fail, which means that although they are able to produce very large amounts of sensed data the number of recorded trajectories to failure remains scarce. Second, there are no established methods to quantify differences between individuals in a fleet of assets. Unlike biology, in which DNA can be used to code for each different individual, physical assets have yet to find a consistent metric to code their differences. All of this means that a framework in which real-time prognostics is determined for individual assets by independent and communicative system components, still falls far from the current approach both in academia and industry. This thesis bridges this gap by presenting a Multi-Agent System for real-time distributed prognostics able to operate in realistic industrial scenarios: Distributed Collaborative Prognostics.

1.6 Distributed Collaborative Prognostics

The idea of Distributed Collaborative Prognostics stems from the concept of collaborative software agents, which are pieces of software that can communicate with each other in order to learn an algorithm or a policy better than if they were to learn it by themselves [24]. In Distributed Collaborative Prognostics, an agent is assigned to each asset and learns a predictive model aimed at predicting its failure. Different agents, corresponding to different assets, share information with each other and use this shared information to refine their predictive models in real time. As discussed in depth in Chapter 2, several frameworks exist that give theoretical grounding to such a system of agents. This thesis uses Multi-Agent Systems (see Chapter 3 for the justification of this choice).

Multi-Agent Systems incorporate agents especially designed to adapt in real-time to different scenarios. In Multi-Agent Systems, collaborative learning allows agents with limited experience and capabilities to acquire knowledge that otherwise would be barred to them [25]. This strategy, based on sharing and processing information horizontally, has been successful in biological systems, and has been applied to engineering challenges such as energy optimisation and traffic control [26]. Agents can collaborate in three ways: by sharing

sensory information, by sharing the consequences of their decision making, and by sharing decision policies. By doing so, agents are able to converge to optimal policies significantly faster, without compromising their accuracy [25].

Inspired by multi-agent collaborative learning, the vision of Distributed Collaborative Prognostics is a distributed, real-time implementation that aims to estimate a machine's time to failure with high accuracy and without the need for extensive prior information. In this thesis, Distributed Collaborative Prognostics is used to determine failure prognostics and optimise maintenance plans. Fig. 1.3 shows a sketch of Distributed Collaborative Prognostics, in which different cars from the same fleet experience failure modes and are part of an interconnected network of assets that collaborate in order to perform prognostics.

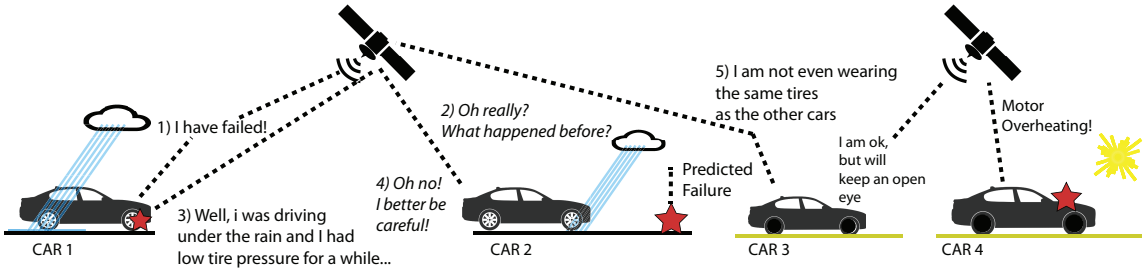


Fig. 1.3 Sketch of Distributed Collaborative Prognostics. In this example, cars predict failure events using past data and inputs from similar cars in the fleet.

1.7 Research questions

The research gaps outlined in the literature review of this thesis led to three different research questions. These questions are focused on achieving the general objective of this thesis, which is to *develop Distributed Collaborative Prognostics*: a real-time solution able to predict asset failures with high accuracy and without the need for extensive prior information.

1. *Theoretical:* How can Multi-Agent Systems be used for prognostics in asset fleets? This research question aims to extend existing approaches in order to enable Distributed Collaborative Prognostics.

2. *Technical:* How can Distributed Collaborative Prognostics be used in conjunction with predictive maintenance? This research question aims to link the proposed collaborative learning approach with maintenance policies.

3. *Practical:* Under which circumstances does Distributed Collaborative Prognostics outperform traditional approaches?

1.8 Theoretical position

It is traditional in PhD theses to outline the ontological and epistemological foundations on which the thesis lays on. The ontological base refers to how the author defines reality, for example whether the author adheres to an idealist view of the world (reality only exists as we think about it) [27], or rather to a realist framework (the belief that nature exists independent of consciousness) [28]. In this case, the author believes strongly in a realist framework, to which this work will also adhere.

Epistemology studies the nature of knowledge; a researcher might be convinced that the physical world is independent of consciousness, but still believe that our understanding of nature is highly subjective. This thesis is founded on a combination of pragmatism (something is true enough if it works) [29], and empiricism (empirical data is paramount) [30].

1.9 Methodology

This thesis follows three phases: an initial exploratory phase, an implementation phase and a validation phase (see Fig. 1.4).

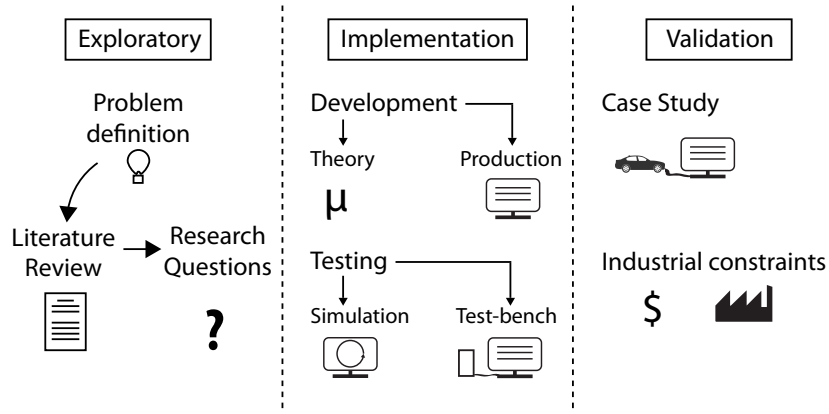


Fig. 1.4 Diagram showing all the phases of the methodology used in this thesis.

In the exploratory phase, a novel solution to an existing problem is proposed. Then, a literature review is undertaken to assess its actual novelty and relevance. Subsequently, the solution is refined further. Research gaps are identified and used to narrow the scope of the research, which is written as a set of research questions. Finally, the tasks and objectives needed in order to address these questions are set up (see Fig. 1.5).

In the implementation phase, the tasks identified in the exploratory phase are performed, and experimental data is used to benchmark whether the objectives connected to these

tasks have been achieved. Normally the implementation phase is focused on developing the engineering solution pertinent to the research problem. Thus, it does not consist of validation using industrial case studies. Empirical testing during this phase is limited to readily-available test-bench cases. For example, public data sets or standard simulation frameworks used to validate solutions in the field.

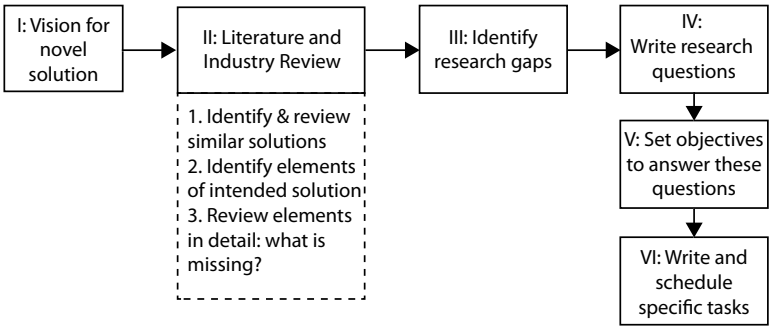


Fig. 1.5 Block diagram showing the initial exploratory phase of this PhD Thesis.

The validation phase consists in empirically validating and improving the engineering solution in real situations. This is done using data from an engineering scenario without reducing its complexity or especially preparing it for a positive outcome. Note that the validation phase is preceded by a test phase, in which the engineering solution is subjected to synthetic scenarios to test its response to unlikely situations. Fig. 1.6 shows the steps taken during the elaboration of this thesis during the implementation and validation phases.

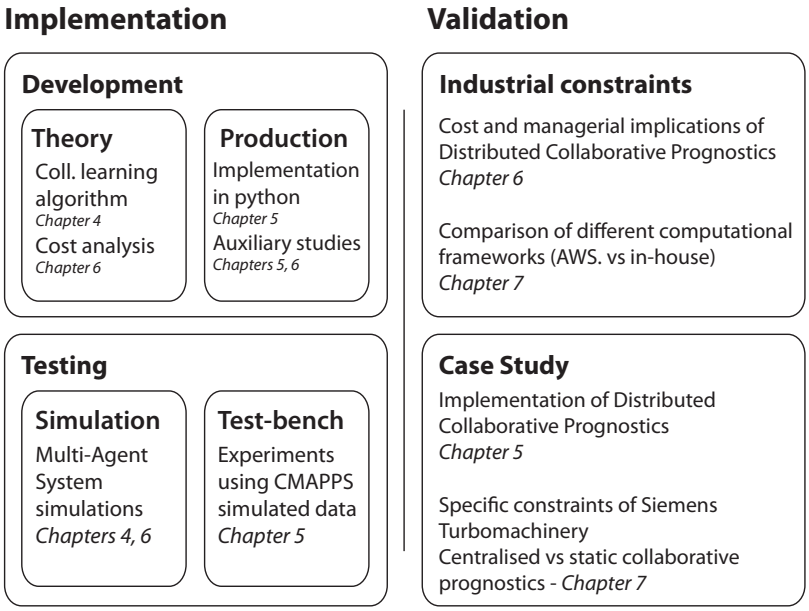


Fig. 1.6 Elements composing the implementation and validation phases of this thesis.

1.10 Thesis outline

This thesis consists of eight chapters. The first chapter motivates the purpose of this thesis, and presents the methodology and research questions that this thesis will address. The second chapter contains the research background of this thesis, including its literature review. The third chapter presents the characteristics of Distributed Collaborative Prognostics and justifies the choice of Multi-Agent Systems as the framework used for its implementation. This chapter also describes the multi-agent architectures used in the rest of the thesis. The fourth chapter studies the coupling of Distributed Collaborative Prognostics and a maintenance policy. This is followed by a chapter in which a deployable implementation of Distributed Collaborative Prognostics is presented. The two subsequent chapters, chapters 6 and 7, focus on the consequences of different implementations of the tool. Chapter 6 studies the cost implications of different architectures, and Chapter 7 contains the case study presented in this thesis. The last chapter presents the conclusions and future work.

A short synopsis of each chapter follows:

1. *Chapter 1, Introduction*: this chapter introduces the contents of this thesis. The chapter starts by introducing asset management, and moves on to describe the problems of maintenance and prognosis. After that, two important properties of industrial asset fleets are described: non-ergodicity and dynamism. This thesis' problem statement is then aimed to devise a tool able to provide prognostics under such conditions. Following the problem statement, Distributed Collaborative Prognostics is described as a possible solution. This leads to this thesis' research questions. This chapter concludes with a description of the methodology followed in this thesis, and with the thesis outline that the reader is reading right now.

2. *Chapter 2, Research Background*: this chapter presents the research background of this thesis. The chapter is divided into two large sections: the first section contains a comprehensive literature review, and the second section contains a description of the theoretical background. This thesis' literature review comprehends three fields of research and their overlap: asset management, machine learning for prognostics, and distributed systems for prognostics. The conclusion of the literature review is that although some distributed implementations for prognostics exist, their building blocks are not capable to collaborate with each other, hindering their performance in real industrial scenarios. The theoretical background section of this chapter is dedicated to describe the problem of prognostics, and the machine learning frameworks used in this thesis to solve it.

3. *Chapter 3, Distributed Collaborative Prognostics: properties and architectures*: this chapter describes the properties of Distributed Collaborative Prognostics, and justifies the use of Multi-Agent Systems as the framework where it is implemented. This justification relies on a particular type of Multi-Agent Systems: Advanced Multi-Agent Systems, that are shown to satisfy all the properties of Distributed Collaborative Prognostics. Additionally, the chapter includes two sections where the different multi-agent architectures used in this thesis and their building blocks are described.

4. *Chapter 4, Distributed Collaborative Prognostics with a maintenance policy*: the second research question of his thesis refers to the coupling of maintenance policies with Distributed Collaborative Prognostics. This chapter is largely dedicated to address this question. To do so, a Multi-Agent implementation of the Distributed architecture is presented, together with a time-based replacement policy. Results show that this policy approximates the least-costly policy, suggesting that it can be used in conjunction with Distributed Collaborative Prognostics. Additionally, results show that the number of collaborating agents, the noise of the system, and the weight given to data obtained from other agents have an important effect on the cost of the system.

5. *Chapter 5, An implementation of real-time Distributed Collaborative Prognostics*: this chapter presents an implementation of Distributed Collaborative Prognostics ready for industrial deployment. This implementation is programmed in python, and is able to provide real-time prognostics estimates using deep learning. This chapter shows that the presented implementation fulfils all the properties of Distributed Collaborative Prognostics presented in Chapter 3 by using a publicly available prognostics data set. On top of this, collaborative learning is shown to outperform fleet-wide learning with regards to its prognostics accuracy. The chapter concludes by describing how its results help addressing the first and last research questions of this thesis.

6. *Chapter 6, Cost implications of Distributed Collaborative Prognostics*: this chapter studies the cost implications of using different architectures for Distributed Collaborative Prognostics. It addresses all three research questions, as it includes a predictive maintenance policy, a Multi-Agent Simulation, and studies in which conditions this thesis' tool should be implemented in industry. This chapter's Multi-Agent implementation of Distributed Collaborative Prognostics extends the implementation presented in Chapter 4 to larger fleets of assets featuring agent failures. The chapter concludes that distributed architectures are better for the case of high-value assets or low communication and processing costs. It also concludes that distributed architectures are

beneficial in the presence of agent failures, as they are more resilient to them than centralised approaches.

7. *Chapter 7, Case study: the Siemens gas turbine fleet*: this chapter contains this thesis' case study, dedicated to implement Distributed Collaborative Prognostics in the Siemens gas turbine fleet. First, a description of the Siemens turbine fleet is given, together with a generalist description of a gas turbine. Second, a preliminary data analysis is presented in which the time distribution of the turbine's events is described, and the distribution of the different sensors in the gas turbines around the events is studied. This leads to the presentation of a data preparation framework, that is later used to reduce the dimensionality of the sensor data. The fourth section of this chapter presents prognostics results for the Siemens gas turbine fleet using a traditional centralised approach, that is later compared with a collaborative approach. The collaborative approach is found to outperform the traditional approach consistently. This chapter then demonstrates real-time Distributed Collaborative Prognostics in the Siemens fleet. The chapter concludes that collaborative prognostics outperforms non-collaborative prognostics, and that prognosis accuracies obtained from synthetic data sets overestimate the accuracies achievable in a real industrial scenario.

8. *Chapter 8, Conclusion and future work* this chapter presents the conclusions to this thesis. This chapter is composed of six sections. The first section presents the general conclusions, which are the explicit responses to this thesis' research questions. The second section presents the caveats of the presented tool: a higher operational cost and complexity, and a tendency for over-fitting. The third section reviews the contributions to academic knowledge stemming from the work presented in this thesis, focusing on the journal publications that have stemmed from it. Subsequently, the fourth section comments on the technological choices made during the design of Distributed Collaborative Prognostics. The fifth section describes this thesis' contribution to industrial practice. The last section presents the future work of this thesis. This can be summarised as the need for further studies concerning larger fleets of machines, the need for ad hoc clustering algorithms able to weight in data quality, and the importance of developing open-source code for Distributed Collaborative Prognostics.

Chapter 2

Research background

This chapter has two parts: a literature review that substantiates the novelty of the proposed solution, and a theoretical background section that introduces relevant mathematical concepts.

2.1 Literature review

This section presents the literature review performed for this thesis. Distributed Collaborative Prognostics is a solution that combines elements from three large fields of research: asset health and performance management, Multi-Agent Systems, and machine learning. The first field, asset health and performance management, provides the economical and managerial framework in which the presented solution operates, and sets its maintenance policies. The second field, Multi-Agent Systems, provides the theoretical and technological foundations of its software elements. The third field, machine learning, provides the methods used to predict asset failures and to compute inter-asset similarity (see Fig. 2.1).

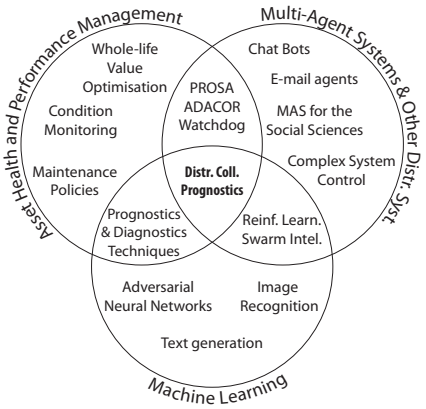


Fig. 2.1 Venn diagram showing the intersection of the three fields reviewed in this chapter. Distributed Collaborative Prognostics incorporates machine learning techniques within a Multi-Agent System to solve an asset health and performance management problem.

2.1.1 Methodology

The literature review that follows is not a systematic literature review. However, most of the papers cited in this thesis have been accessed following the procedure described below. Exceptions are writings that have been recommended by colleagues, or papers that have been found through references in the writings obtained using this procedure.

1. The relevant search terms (see Table 2.1) are imputed in *Scopus* (www.scopus.com) and *Google Scholar* (https://scholar.google.com). Conditional terms are used to restrict the search result.

2. The writings from each search are split in four lists: 1) review papers, 2) journal articles, 3) books, 4) conference papers.

3. For each list of writings with more than 70 entries, the 70 most cited items and the 20 most recent items are selected, and their abstracts are read. Lists that have less than 70 entries are selected in their entirety.

4. Only items relevant to this thesis' interest are chosen for further reading (about one of every five items selected in step 3) are finally included).

5. The selected writings are read and their knowledge incorporated into the literature review.

Field	**Machine learning**	**Asset management**	**Distributed systems**
Term 1	Regression	Prognostics	Multi Agent
Term 2	RNN / LSTM	Diagnostics	Holonic
Term 3	Dimensionality reduction	Maintenance (Policy)	Distributed
Term 4	Anomaly Detection	Condition Monitoring / IoT	Edge Processing
Conditional term 1	Prognostics	Real(-)time	Maintenance
Conditional term 2	Machine	Data(-)driven	Prognostics
Conditional term 3	Asset management	Health Management	Diagnostics
Writings included	34	42	45

Table 2.1 Terms used to search writings for the literature review of this thesis. Conditional terms indicate terms that were used to refine further the large number of results appearing from using the non-conditional terms as only input. The number of total papers included does not include other references used in the rest of the thesis.

This procedure has been repeated several times during the elaboration of this thesis (at least once each year and once for each of the several papers that have been written during its elaboration).

2.1.2 Asset health and performance management

Asset health and performance management[1] can be defined as the task of maximising the whole-life value of an asset [1]. This corresponds to the total value that it generates during the phases of deployment, operation, maintenance, improvement and removal. Typically, an asset goes through these phases whilst owned by different organisations (for example, often the organisation that manufactures the asset is not the same as the one that disposes of it). The period during which the value of an asset affects a given organisation is known as the responsibility period [1]. Distributed Collaborative Prognostics operates during the phases of operation and maintenance, and thus concerns organisations that have these phases in their responsibility period.

In order to maximise the value generated by assets during said period, managers typically focus on reducing failures and maintaining performance while minimising cost [31]. Failures and performance drops can be either measured or predicted, defining the two different problems that form the core of asset health management: diagnostics (including failure detection) and prognostics. Diagnostics refers to the assessment of the state of an asset, and prognostics refers to future predictions about its state [7].

Once diagnostics or prognostics have been performed, the asset manager decides on a maintenance policy, aimed at achieving the optimal balance between cost, risk and performance [32]. Maintenance policies are decision-making heuristics that determine the actions and schedule of an asset's maintenance team. In their most basic form, maintenance policies are designed to leverage the trade-off between the costs of corrective maintenance (maintenance of a failed asset), and preventive maintenance (maintenance of a working asset) [7]. In most occasions, when managing a failure mode, the preventive maintenance strategy is significantly cheaper than the corrective maintenance one. Maintenance policies are used to hedge this difference [7].

Maintenance policies

While diagnostics and prognostics tend to be a data-driven field, the design and optimisation of maintenance policies is often-times analytical. In order to determine these policies, researchers assume certain properties in a single-unit or multi-unit system and from these properties derive the optimal maintenance policy. This policy is normally described as a sequence of times in which maintenance actions are supposed to take place [33].

If no information is known about the failure time probability distribution of the assets, often the only reasonable maintenance policy is corrective: to wait until the asset has failed

[1] Often shortened as asset management.

and then replace it [34]. Fortunately, this is very seldom the case, as the large size of modern asset fleets allows for an estimation of the failure-time probability distribution of the fleet [35]. This must not be confused as a prognostics method: individual assets do not have predictions updated in real time. Instead, an asset failure time is assumed to be sampled from the fleet's failure time probability distribution, and maintenance policies are generated from this assumption [36, 33].

Optimal maintenance policies once the fleet's failure time distribution has been inferred are well-known in reliability engineering. Some of the most common are the "constant interval", and "age based" policies. The former finds an optimal interval to perform preventive replacements of an asset, irrespective of the age of the asset. The later considers the age of the asset itself, and samples its expected time to failure from the truncated distribution of the fleet's failure time probability distribution [33].

An influential improvement on maintenance policies came with the usage of the Proportional Hazard Model to measure instantaneous hazard at any time. This model assumes that the instantaneous hazard is the product of a positive function p dependent of the measured sensor values, and a hazard dependent on the age of the asset. The instantaneous hazard is then updated compared to the baseline Weibull hazard[2]. Usually, the instantaneous hazard is leveraged in the optimisation of the replacement policy, and an optimal hazard value at which the asset must be preventively repaired is obtained [37].

For the case of prognostics, the optimal maintenance policy for a given asset can be naively obtained by taking the computed probability distribution of time to failure as if it was the failure-time probability distribution for each asset. Then, the optimal "constant interval" policy can be calculated at each time-step [38]. This approximation assumes that the probability distribution of time to failure will remain constant, which is often not the case. In practice, the cost-efficiency of prognostics-based maintenance policies is compromised by two factors: (1) preventive maintenance policies have decreased the number of measured failures (see the discussion in [33]), (2) the uncertainty associated to prognostics is hard to estimate. The first factor can be addressed by experimental testing, in which machines are systematically run until failure, or by implementing risk-prone maintenance policies in assets with limited failure consequences. The second phenomena is an ongoing matter of discussion in the world of prognostics and regression techniques [39–41].

Diagnostics and condition monitoring

Diagnostics has a longer history than prognostics, with papers dating as far back as the late 70's [42]. Before the invention of remote sensing, diagnostics was based on inspection.

[2]Other probability distributions can also be used.

Typically a team was sent to investigate the state of the asset and maintenance was performed according to the reported state [43]. With the spread of sensor and communication technologies, monitoring assets in real time became feasible. This new approach to diagnostics was coined condition monitoring (CM) [44].

Condition monitoring provides the information needed to update maintenance plans as the assets deteriorate. This practice is known as Condition Based Maintenance (CBM), and has become common practice in industry [44]. CBM refers to all steps of asset management that are enabled by condition monitoring, summarised in data acquisition, data processing and maintenance decision making [7]. State of the art CBM research focuses on implementing the decision making as an integral part of the monitoring and diagnosis system [44].

The technologies employed for condition monitoring directly overlap with those used in the Internet of Things. The Internet of Things advocates for physical assets (things) having identities, operating intelligently and communicating within a social context [45]. In the IoT, assets share sensed data with an ever-growing network of devices, with the hope that this data will be used to improve understanding of asset behaviour [46, 47]. This paradigm has an industrial branch, the Industrial Internet of things (IIoT) [48]. Technological improvements linked to the IoT, like cheaper sensors and widespread connectivity have put condition monitoring in the center of many diagnostics architectures, reducing maintenance cost and machine downtime [49].

Much of this reduction has come to be thanks to the employment of automated diagnostics. Automated diagnostics of failure events makes use of many techniques, spanning different ranges of complexity [7]. The most basic and ubiquitous technique is the implementation of threshold-based alarms: a fault is detected when the values returned by a sensor or a group of sensors exceed a pre-set threshold [50]. The elements comprising this technique have been further standardised (see [51, 52]). In these standards, sensors are conceptualised as descriptors. Descriptors produce symptoms, qualitative indicators of the presence or absence of a fault. A descriptor is then a measure specifically linked to a symptom and not just a variable of the system. Threshold alarms can be understood as interpretation rules on these descriptors, quantitatively determining diagnostic rules (see [53] for a discussion of the ISO standards in the context of Condition Based Maintenance).

Threshold-based alarms are present in the assets used for this thesis' case study, and produce targets that are conducive to regression-based prognostics techniques (see Chapter 7). In many cases, machines are programmed so that once such a fault is detected they stop operating in order to avoid further damage [54]. A threshold-based alarm can be seen as a type of recurrent failure, in which a machine is put out of operation by means of its own control mechanisms.

Another popular method is to use Fourier or wavelet transforms to transform a time series signal to a frequency space, where faults are easier to identify [55, 56]. Automated classification is also used to separate failure sensor data from healthy data. In order to do so, most machine learning classification methods have been used [57], from decision trees [58] to Neural Networks [59], and logistic regression [60]. Another method, used mostly to improve threshold-based methods and avoid false positives due to machine transients or other misleading sensor readings is fuzzy-logic reasoning [61, 62]. Finally, dimensionality reduction methods are used in diagnostics to synthesise the data obtained from different sensors into an indicator of the state of the asset, for example a Health Indicator [63, 64].

Prognostics

Prognostics aims to provide quantifiable information regarding the future state of an asset: for example, the expected time left until the next failure. In fact, prognostics can be defined in connection with diagnostics: "predictive diagnostics which includes determining the remaining useful life or time span of useful operation for a component" [65].

Successful prognostics rely on a combination of reliable time to failure models with a good assessment of the current state of the asset [44]. Provided that the prognostics models are reliable, an asset manager is then able to optimise maintenance and operation policies according to this knowledge. This is typically done by using the predicted probability of failure in models that seek to optimise the economic output of the asset [66] (see Section 4.3 for an example).

There are two ways to describe prognostics: either as the estimated Remaining Useful Life (RUL), i.e. the time to failure, or as the probability of failure within a future time interval [44]. In reality, both methods often overlap as probability distributions of the time to failure allow for the determination of both parameters simultaneously [12].

Prognostics methods can be roughly divided in two big categories[3] *physical methods* and *data-driven methods* [11, 12]. In the first case, physics-based models are computed using the initial condition of the asset, and the future state of the asset is determined by the evolution of its governing differential equations. Such models can be very accurate when the deterioration process leading to failure is determined by non-chaotic equations or when the initial state of the asset is known with very small uncertainty. Unfortunately, physics-based models are very demanding computationally and reliable simplified versions only exist for a few applications [11, 67]. This reduces their applicability to a subset of well-understood processes and limits their scalability in large fleets of assets [11, 12]. Despite this, such models have been applied

[3]Methods in the intersection of the two categories also exist (see [67, 68]).

successfully several times and remain a central tool in machine prognostics (see [69–72] for several examples).

Data-driven methods for machine prognostics normally rely on machine learning algorithms. Given the extent and importance of these, they are reviewed in the next section and formally described in the Theoretical background (Section 2.2).

2.1.3 Machine learning for prognostics

Machine learning methods, based on the use of empirical evidence to train computer-generated models, have been used in prognostics since the early 2000s (see, for example, [73, 74]). Machine learning methods can be divided in two large categories: supervised, and unsupervised [75]. Supervised methods produce computer-generated models using a training set of input and prediction variables with the objective of the model generalising to a different set of data[4]. Unsupervised methods look for patterns in the data without the need of being provided with explicit prediction variables, its objective being to find a compact representation of the data set [75].

As discussed earlier in this chapter, prognostics essentially consists of predicting the time at which a machine will fail. Thus, the most popular machine learning methods used in prognostics are supervised methods for regression in which the prediction variable is the time to failure. Mathematical methods used for future trending vary from Gaussian processes [77, 78], to polynomial extrapolation and vector machines [79, 80]. Neural Networks can be used to predict classes within a sliding box model, or in order to estimate the time to failure directly [33, 81–83]. Classification is mostly used in diagnostics to detect anomalies in sensor readings [84]. However, it can also be adapted to prognostics through the aforementioned sliding box model which classifies multi-sensor time series in different categories corresponding to different intervals of time to event data (see Sec. 7.4.1).

Together with this, methods used to prepare asset data to make it more conducive to regression are also very popular [85]. Some examples of these methods are the wide use of principal component analysis and proportional hazards modelling [86, 87, 35]. Extended reviews of other prognostic models, including other Bayesian techniques, hidden Markov models and fuzzy systems can be found in [35, 78].

In this thesis, Recurrent Neural Networks (RNN) [88] are chosen for prognostics because they are designed to handle patterns with the characteristics encountered in industrial failure data: non-linearity, noise, and time-dependency. Theoretically, RNNs are Turing complete and thus can learn complex temporal patterns [89]. The caveats of using RNNs are that they

[4] If the test data is used to further refine the hyper-parameters of the machine learning model, an additional subset of data is needed for validation [76].

require more computational resources than other regression methods, and that they have a tendency towards over-fitting. The theoretical background for the machine learning methods employed in this thesis is presented in Section 2.2.

2.1.4 Distributed systems for prognostics

This thesis explores the advantages of using a distributed approach to perform prognostics in a fleet of assets. In the asset management *lingua*, such fleets are known as multi-unit systems. Other terms, such as multi-component systems are reserved to the study of assets formed by many inter-dependent components. A multi-unit system usually refers to a fleet of several similar assets that share common failure modes. Because a multi-unit system corresponds to a fleet of similar assets, units are often assumed to be essentially equal, thus reducing the complexity of the problem [14, 90, 91].

Multi-unit systems can be managed in a centralised way (a single software component is responsible for the full fleet), or in a distributed way (when several agents are used). Although both approaches have been implemented industrially, centralised approaches are more common due to their simplicity. In a centralised architecture, asset data is stored in a single database, which then is used to train prognostics models for the rest of the assets. Examples of this can be found in knowledge based systems [92]. Most machine learning implementations for prognostics are also centralised (see [88, 93, 94]).

Distributed approaches are usually employed when the assumption of equal assets is dropped, and individualized prognostic models become the preferred option. The many existing approaches that deal with multi-unit systems in a distributed way can be broadly classified in two categories: Multi-Agent Systems, and other distributed systems.

Multi-Agent Systems

Multi-Agent Systems are software systems composed of many independent agents that aid humans in taking decisions [95]. Their distributed and continuously adaptable nature makes them ideal candidates to deal with the non-ergodic and dynamic properties of industrial asset fleets [22]. Apart from their applications in prognostics, reviewed in detail later in this section, Multi-Agent Systems have been successful in solving problems in industrial production, traffic management, etc. [96].

Before diving into the specific applications of Multi-Agent Systems in prognostics, it is worth to clearly establish the definition of the word “agent" used in this thesis. Despite (or perhaps because of) its great popularity, the meaning of the word agent has remained contested over time [24]. In this thesis, a quite restrictive definition is chosen: agents are

autonomous, problem-solving, and goal-driven computational entities with social abilities [22]. According to this definition, a piece of software that, for example, is limited to collecting data from an asset and sending over this data to another piece of software is not considered an agent.

Another important defining element of any given Multi-Agent System is its architecture, that determines hierarchical relations among agents and their communicative heuristics. Broadly speaking, there are three[5] types of architectures in Multi-Agent Systems: Hierarchical, Heterarchical, and Distributed [22, 97]. In a Hierarchical architecture information follows a predetermined path across the hierarchy, from lower to higher levels. Decisions then follow the inverse path, with the higher level agents in the architecture taking priority, and often deciding, over the lower level agents [98]. Heterarchical architectures allow agents within the same level in a Hierarchical architecture collaborate with each other without the need of upper-level agent intermediation [99]. Purely distributed architectures take this approach to its ultimate consequences, and as such are formed by a single type of agent that performs all the tasks of the system. In this case, communications are peer-to-peer, and there is no hierarchy [22]. A detailed description of these architectures is given in Sec. 3.3.

Hybrid architectures have been proposed as a framework for holonic manufacturing systems. An example of such an architecture is ADACOR (and its evolution, ADACOR^2) [100, 101]. ADACOR has been postulated as a tool for diagnostics, and has as its core component its ability to switch from a Hierarchical to a Heterarchical architecture in presence of disturbances (for example, machine failures).

The study of Multi-Agent Systems is a mature field of research that in the last years has strived towards standardisation. This has produced, among others, standards regulating the way in which agents communicate with each other (known as agent communication languages (ACL)), standards dealing with agent security, etc. [102, 103]. Implementation of some of these standards is made easy by open source libraries dedicated to that purpose. For example, in python `pykqml` is a library defined to easily convert a message to the Knowledge Query and Manipulation Language (KQML), a widely used agent communication language [104, 105].

When it comes to prognostics, Multi-Agent Systems have been used in multiple occasions, although the roles of agents in these applications have varied. The first mentions of the potential of Multi-Agent Systems for prognostics date from 2007, when Liu et al. and Gonzalez et al. proposed the usage of Multi-Agent frameworks for prognostics in a fleet of electric ships [106], and as part of an architecture incorporating the OSA-CBM standards

[5]The centralised architecture is omitted because it is composed of a single agent.

[107] [6]. Implementation was first demonstrated a year after, when Ivan S. Cole and a team of scientists at Boeing successfully employed a Multi-Agent System to enrich sensor data in an aircraft [110]. In their work, each aeroplane hosts several agents that manage the data of several sensors. The enriched data produced by the agents is then used to produce prognostics in a centralised computer.

True distributed prognostics in a Multi-Agent System was not demonstrated until 2012, in a paper by Xavier Desforges in which each subsystem of a device is assigned a prognostics agent [111]. In Deforges' approach, distribution is still seen as a mean to increase computation speed and accuracy, rather than a way to handle dynamic and heterogeneous fleets. This is similar to Wei Wu's work on multi-agent based prognostics. In Wu's work, a multi-agent approach is taken mainly from an algorithmic point of view, in which agents are an initial set of weights to be tried in an artificial Neural Network [112].

Following Desforges work, a series of papers focused on detailing possible implementations of multi-agent prognostics, some of them describing new architectures. For example, Amy J. C. Trappey introduced in 2013 an architecture to enable collaboration between agents and humans in maintenance environments [113]. A year later, Luca Fasanotti, proposed an architecture to merge Multi-Agent Systems and artificial immune systems for machine prognostics [114]. Fasanotti would later go on to publish an implementation of the system in 2018 [115]. Other work in the same period falls in the middle between architectural description and implementation (see for example [116]).

Across 2018 and 2019, Multi-Agent Systems for prognostics have continued to enjoy research interest. Apart from the publications written by the author of this thesis, Ghita Bencheikh (working with Xavier Desforges) has used the prognostics capabilities of Multi-Agent Systems to solve scheduling of production and maintenance activities [117]. Multi-Agent Systems have also been used to perform cooperative prognostics for three proton-exchange membrane fuel cell stacks [118].

Multi-Agent Systems are not the only type of distributed framework used for prognostics. As shown in the pages that follow, many researchers choose other distributed approaches for their prognostics implementations and avoid mentioning Multi-Agent Systems altogether.

Holonic and other distributed systems

Many of the papers that research distributed prognostics do so without engaging with the field of Multi-Agent Systems. The reasons for this can be that researchers are using pre-existent implementations that don't focus on the Multi-Agent System paradigm, or that they use

[6] An influential set of standards for condition based maintenance [108, 109].

agents that do not fulfil stringent agent definitions such as the one discussed in the last section.

An example of a well-known agent architecture adaptable to distributed prognostics is the Watchdog Agent [17]. In this architecture, an agent which monitors the asset condition and provides diagnosis and prognosis services is assigned to each industrial asset. These services are not only based on the asset condition but also on expert knowledge. More nuanced decisions such as proactive maintenance decisions are done by a decision support tool, which is not installed in the agent. The Watchdog Agent is based on the OSA-CBM standards, is programmed entirely in LabView [119], and is sold as a proprietary software.

The Watchdog Agent has been used for prognostics in a variety of research studies (see for example [120–122]). However, collaborative prognostics in a real fleet of assets has remained unexplored, with the closest example being Shi's work [122] in which prognostics is calculated in parallel in a LabView program without communication between the agents. Due to the proprietary nature of the Watchdog Agent, a lot of its industrial implementations have likely remained unpublished.

Other existing implementations of distributed prognostics utilise a multiplicity of tools. To review just some, Hadden G. D. et al. proposed a distributed prognostics and diagnostics architecture especially tailored to manage ship monitoring systems [123]. Jinjiang Wang et al. focused on developing an affordable sensing and computing node able to perform prognostics independently. This node, that enabled communication between software components called mobile agents, was then validated on a fleet of six test induction motors [124].

Zhou J. et al. proposed what essentially is a Multi-Agent System composed by several Java agents featuring real-time data acquisition and agent-to-agent communication [125]. Interestingly enough, they chose to avoid any explicit references to Multi-Agent Systems. Another example of what is essentially a quasi Multi-Agent System is the influential paper by Chen C. et al. in which Microsoft's .NET framework is used to perform prognostics in independent system components [126]. Note that this paper shares some co-authors with the earlier work of Michael Roemer et al. that provided architectural and procedural instructions for distributed prognostics [127]. Another similar study by Wang J. et al, where distribution is based on mobile agents on the cloud, incorporates agent-to-agent messaging and has a real-time capability [124]. A more detailed review of how these distributed systems compare to Multi-Agent Systems can be found in [23] (a paper by the author of this thesis).

These *quasi* Multi-Agent Systems studies are not the only case in which distributed architectures are leveraged in prognostics. Some authors see distribution as a way to share the computational load of a centralised prognostics algorithm. A good example is Saha S. et al. [128] who envisaged distributed prognostics as a way to take advantage of the

computing power distributed in the fleet's machines. Other existing architectures that implement distributed health and performance management often deal with diagnostics (see, for example, [129–131]).

Shortcomings of existing approaches

Distributed real-time prognostics has been postulated for a variety of scenarios. However, actual implementations are scarce and often concern systems composed by few assets, or multi-component systems. Most of the papers reviewed above do not focus on providing a complete tool for distributed prognostics. Rather, they present a novel prognostics approach to solve a specific industrial problem (see, for example, [118, 123]). This means that existing solutions are often not transferable across industrial scenarios.

Apart from this lack of transferability, a frequent shortcoming of existing solutions is that some don't provide true real-time capabilities, and use distribution just as means of improving computational speed or accuracy. From those that provide real-time capabilities many concern multi-component systems and therefore operate in the intra-asset level instead of at the fleet level. A good example of this are implementations that focus on handling different sub-components in complex machines (see, for example, [110] and [111]).

Despite most published work falling in the categories described above, a few solutions do provide transferable real-time prognostics for multi-unit systems (for example, the Watchdog agent, and Jinjiang Wang's sensing and computing node). However, their experimental support is often based on synthetic data sets or fleets composed of a few assets. Most importantly, their design has not yet been adapted to inter-agent communication and collaboration.

In general, the agents that come closer to this thesis' vision lack the capabilities of collaborating with each other, thus missing a key property of modern agents. This thesis extends the idea of asset-specific agents to Multi-Agent System theory and provides them with inter-agent communication capabilities, separated data spaces, decision making faculties, and a system able to dynamically adapt to varying conditions in the asset fleet. The most significant addition to previous practice is to describe how collaboration can be used to significantly improve prognostics accuracy, and how a Multi-Agent System can be used to dynamically choose different subsets of collaborating assets.

2.2 Theoretical background

This section presents the theoretical background of the machine learning algorithms used in this thesis. The prognostics problem is presented mathematically, and the algorithms used to solve it are introduced. For the prognostics problem, two loss functions obtained from the time

to failure of a machine are presented. These loss functions can then be used in conjunction with different machine learning methods. The loss functions presented here are long-known results from the field of survival analysis and machine learning. Notwithstanding, they must be introduced as they form the backbone of the prognostics produced by the software agents in Distributed Collaborative Prognostics.

2.2.1 The prognosis problem

The prognosis problem can be formally stated as follows: for an asset i, find a function $f_i(T, x_{0:t})$ that gives the probability per unit time[7] that a failure event occurs at a time T. Here, $x_{0:t}$ is a multivariate $m \times t$ vector that contains the time series of all the m input variables (normally sensor values) recorded in the past (see, for example, [132]).

Failure trajectory

A failure trajectory (or trajectory to failure) is the sequence of feature values $x_{0:T_n}$ that precede a failure recorded at time T_n. This is denoted as $x^n_{0:T_n}$ where n indicates that this trajectory corresponds to the *nth* observed failure in the data set. If a trajectory has not yet reached failure, its feature values are $x^n_{0:t}$ (all the data available until the current time). A trajectory of this type is also known as a censored trajectory, while a trajectory for which T_n is known is an uncensored trajectory.

In order to clarify what is meant by trajectory, Fig. 2.2 shows a sketch of the matrix fed to the machine learning algorithm for training purposes.

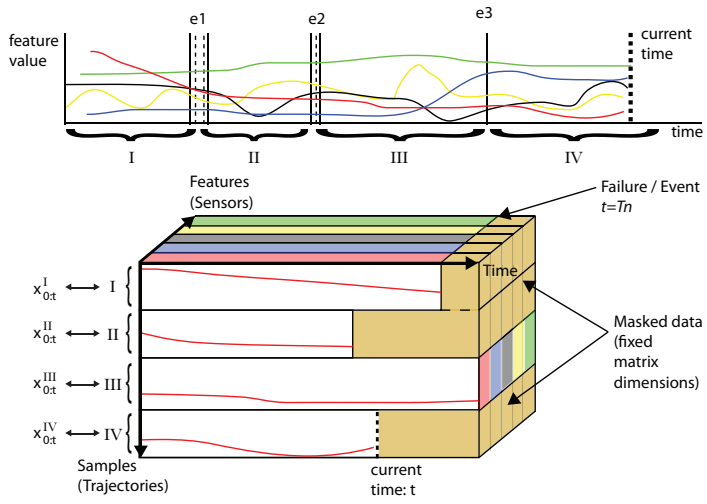


Fig. 2.2 Sketch showing the training data matrix fed to the machine learning algorithm.

[7] The probability density function.

2.2.2 Loss functions

Supervised machine learning methods are used to estimate a function that minimises a loss function, expressive of the difference between its predictions and a set of target values in the training set. A correct loss function choice is crucial because it is the metric that tells the machine learning method whether the function that it has generated fulfils the desired task.

For regression, a widely used loss function is the Mean Squared Error (MSE) between the predicted values and the target values:

$$l = \frac{1}{N}\sum_{n=1}^{N}\left(y_t^n - \mu\left(x_{0:t}^n\right)\right)^2. \tag{2.1}$$

In this equation $\mu\left(x_{0:t}^n\right)$ corresponds to the predicted time to failure. y_t^n is the target or real time to failure at time t (note that the time to failure decreases as t increases). N corresponds to the number of training pairs $(x_{0:t}^n, y_t^n)$. This loss function is derived from assuming that each target of the machine learning algorithm is sampled from a Gaussian distribution with mean $\mu\left(x_{0:t}^n\right)$ (more of this later).

According to the definition put forward in Section 2.2.1, the prognostics algorithm should return not only the predicted time to failure but also information of the probability of failure at any further time. Uncertainty estimation remains an open topic in deep learning [133]. However, the mean and variance of the target probability distribution can be estimated directly by maximizing the log-likelihood of the targets, providing a measure of the error of the Neural Network outputs [134, 135].

The likelihood function, $\mathscr{L}$ gives the probability that a set of observations is drawn from different parametrisations of a probability distribution [136]. Maximising the likelihood corresponds to estimating the parametrisation that is more likely to explain a set of observations. Therefore, the likelihood is often written as $\mathscr{L}(\theta|x)$ where θ are the parameters of the probability distribution from which the observations may have been drawn, and x are the observations. In this thesis, the observations are the times to event y_t^n, and the parameters vary depending on which probability distribution is chosen for $f_i(T, x_{0:t})$. For continuous distributions, maximising the likelihood function corresponds to maximising the density probability function[8]:

$$\max_\theta \mathscr{L}(\theta|y_t^n) = \max_\theta\left[\lim_{h\to 0^+}\mathscr{L}(\theta|y\in[y_t^n, y_t^n+h])\right] = \max_\theta\left[\lim_{h\to 0^+}\frac{1}{h}\int_{y_t^n}^{y_t^n+h} f(y|\theta)\,dy\right] = \max_\theta f(y_t^n|\theta). \tag{2.2}$$

[8]The probability of a given observation y_t^n is the integral over a very small interval of the density function.

Assuming that the features $x_{0:t}^n$ are independent and identically distributed, the log-likelihood can be written as a sum of the log-likelihoods of the examples ($\mathscr{L}_t^n$). For the observation pairs $(x_{0:t}^n, y_t^n)$, this reads:

$$\log(\mathscr{L}) = \sum_{n=1}^{N} \sum_{t=0}^{T_n} \log \mathscr{L}_t^n(\theta | y_t^n) = \sum_{n=1}^{N} \sum_{t=0}^{T_n} \log \left[\Pr_\theta \left(Y_t^n = y_t^n | x_{0:t}^n \right) \right]. \tag{2.3}$$

Maximising this equation means to 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 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 explicitly obtain $\Pr_\theta \left(Y_t^n = y_t^n | x_{0:t}^n \right)$, one needs to designate a parametric representation of the probability distribution of $f_i(t)$. One of the most convenient of these parametrisations is the Gaussian probability distribution.

The Gaussian-based loss function

The Gaussian distribution is an extremely popular distribution to estimate and represent uncertainties. Gaussian distributions fulfil a number of convenient mathematical properties, and are especially conducive for machine learning because they are the only family of probability distributions that have a variance that is independent from its mean (once obtained from a set of independent measurements) [137]. The log-likelihood of a Gaussian with the parametrization $f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x-\mu)^2)}{2\sigma^2}}$ is [134]:

$$\log(\mathscr{L}) = \sum_{n=1}^{N} \sum_{t=0}^{T_n} \left[-\frac{1}{2} \log \sigma^2(x_{0:t}^n) - \frac{\left(y_t^n - \mu(x_{0:t}^n) \right)^2}{2\sigma^2(x_{0:t}^n)} \right]. \tag{2.4}$$

In this log-likelihood, constant terms have been dropped because they do not have any effect on the optimisation procedure. The loss function to minimise is then the negative of the log-likelihood $l = -\log(\mathscr{L})$. Minimising this loss function gives the parameters $\left(\mu(x_{0:t}^n), \sigma(x_{0:t}^n) \right)$ for each sequence $x_{0:t}$ that fully determine the function $f_i(T, x_{0:t})$, thus providing a solution to the prognostics problem. Here, T is defined as the time from t, which is the time at which prognostics is computed.

The problem of using a Gaussian distribution to estimate time to failure is that this distribution admits failures that occur in negative times. In other words, this distribution always contains non-physical probabilities (the probability that a failure event happened in the past despite the asset having already survived until the present time is not zero).

The Weibull distribution

Similar to the Gaussian case, a Weibull distribution can also be used as the parametrisation for $f_i(t)$. This is the main assumption behind a machine learning approach known as Weibull Time To Event - Recurrent Neural Networks (WTTE-RNN) [83]. This approach has the benefit that it allows to use both censored and uncensored trajectories in the training of a Recurrent Neural Network.

From a theoretical perspective, WTTE-RNN is important because it provides the appropriate mathematical framework to use reliability-inspired loss functions in Recurrent Neural Networks, and because it elegantly links the loss function with survival analysis allowing to train the Recurrent Neural Networks using censored data. For the sake of completeness we provide a brief description here.

In survival analysis, one usually fits a probability distribution $f(t)$ to useful life time data (here useful life time is understood as the time that a machine has lasted before failure). The useful life time of a machine is then assumed to be a random variable to be sampled from this distribution. From these simple assumptions, the survival function is defined [9]:

$$S(t) = 1 - F(t) = 1 - \int_0^t f(s)ds. \tag{2.5}$$

$F(t)$ is the cumulative distribution function (CDF). $S(t)$ is the probability of an event (or failure) not occurring before time t, also known as the probability of survival. $S(t)$ is also known as reliability. From the reliability and the probability of failure we can obtain the failure rate $\lambda(t)$, also known as hazard or instantaneous hazard:

$$\lambda(t) = \frac{f(t)}{S(t)}. \tag{2.6}$$

Note the following relation:

$$\lambda(t) = -\frac{d}{dt}\log S(t) = -\frac{d}{dt}\log\left[1 - F(t)\right] = -\frac{F'(t)}{1 - F(t)} = \frac{f(t)}{S(t)}. \tag{2.7}$$

Traditional reliability methods rely on obtaining the conditional probability distribution given the knowledge that a machine had already survived until time t_{surv}. This can be done by taking the original probability distribution, $f(s)$ so that when $s = 0$, $f_s(0) = f(t_{\text{surv}})$, and renormalising it to the probability density for $s > t_{\text{surv}}$ (which is the survival function $S(t_{\text{surv}})$):

$$f_s(t) = \frac{f(t+t_{\text{surv}})}{1-\int_0^{t_{\text{surv}}} f(s)ds} = \frac{f(t_{\text{surv}}+t)}{S(t_{\text{surv}})}. \tag{2.8}$$

The link between this classical survival approach and the WTTE-RNN method is substantiated in the following paragraphs. In WTTE-RNN, the proposed log-likelihood function to be maximised by the Neural Network is [83]:

$$\log(\mathscr{L}) = \sum_{n=1}^{N}\sum_{t=0}^{T_n} u_t^n \log\left[\Pr\left(Y_t^n = y_t^n | x_{0:t}^n\right)\right] + (1-u_t^n)\log\left[\Pr\left(Y_t^n > y_t^n | x_{0:t}^n\right)\right]. \tag{2.9}$$

Where u_t^n indicates whether the observation at time t is censored (if the real failure time has not yet been observed, then $u_t^n = 0$). The first contribution to the log-likelihood, $u_t^n \log\left[\Pr\left(Y_t^n = y_t^n | x_{0:t}^n\right)\right]$, means that if the real time to failure has been observed ($u_t^n = 1$, uncensored), one simply reverts to eq. (2.3). The second term, $(1-u_t^n)\log\left[\Pr\left(Y_t^n > y_t^n | x_{0:t}^n\right)\right]$ addresses how to train the algorithm with censored data. If the real time to failure has not been observed ($u_t^n = 0$, censored), the algorithm is set to 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 probabilities appearing in eq. (2.9) can be obtained by means of survival analysis (derivation from [83]). For the continuous case, the likelihood (for each component) can be rewritten:

$$\mathscr{L}_t^n = f(y_t^n)^u \Pr(Y_t^n > y_t^n)^{1-u} = f(y_t^n)^u S(y_t^n)^{1-u} = \lambda(y_t^n)^u S(y_t^n). \tag{2.10}$$

Note how here, eqs. (2.6), (2.2), and the definition of $S(t)$ have been used. Taking the logarithm:

$$\log(\mathscr{L}_t^n) = u\log(\lambda(y_t^n)) + \log(S(y_t^n)) = u\log(\lambda(y_t^n)) - \int_0^{y_t^n} \lambda(s)ds \equiv u\log(\lambda(y_t^n)) - \Lambda(y_t^n). \tag{2.11}$$

In here, the integral of eq. (2.7) has been used with the condition $S(0) = 0$. u has the same role as u_t^n. $\Lambda(t)$ is known as the cumulative hazard function, defined as the integral of the hazard function ($\lambda(t)$):

$$\Lambda(t) = \int_0^t \lambda(w)dw. \tag{2.12}$$

If one assumes that $f(t)$ conforms to a Weibull distribution[9], the cumulative hazard is:

$$\Lambda(t) = \int_0^t \frac{f(w)}{1 - \int_0^w f(s)ds} dw = \int_0^t f(w) \exp^{\left(\frac{w}{\alpha}\right)^\beta} dw = \int_0^t \frac{\beta}{\alpha} \left(\frac{w}{\alpha}\right)^{\beta-1} dw = \left(\frac{t}{\alpha}\right)^\beta. \quad (2.13)$$

Where α is the scale parameter and β is the shape parameter. Combining eqs. (2.11) and (2.13) the continuous log-likelihood (added over all trajectories and all time-steps, and using the concept of Recurrent Cumulative Hazard Function as shown in [83]) is:

$$\log(\mathscr{L}) = \sum_{n=1}^{N} \sum_{t=0}^{T_n} \left(u_t^n \left\{ \beta_t^n \log\left(\frac{y_t^n}{\alpha_t^n}\right) + \log\left(\beta_t^n\right) \right\} - \left(\frac{y_t^n}{\alpha_t^n}\right)^{\beta_t^n} \right). \quad (2.14)$$

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. The unconstrained optimization problem to be solved by the Neural Network can be then summarised in finding the weights w that maximise $\log(\mathscr{L})$. This corresponds to minimising the negative of the log-likelihood $l = -\log(\mathscr{L})$.

The dependency of the shape of a Weibull distribution (and more specifically its variance) with its defining parameters, α and β, make this optimization problem often difficult to solve. Eq. (2.14) features some opportunities for numerical instabilities: negative values of the logarithm's argument and exploding gradients being the most common. To solve this, α and β must be carefully constrained [83].

2.2.3 Neural Networks

Neural Networks with non-linear activation functions are machine learning frameworks introduced in the 1960s that profit from the large amount of possible combinations emerging from stacking layers of interconnected elements known as "neurons" [138]. The underlying idea of a Neural Network is simple: a directed acyclic graph represents a model that applies a series of concatenated operations to an input, generating an output. Neural Networks are in fact derivations of a very old idea, as they can be reduced to multiple linear regression methods known since at least the early 1800s by Legendre and Gauss [139]. Neural Networks, however, have benefited from at least two important improvements since the nineteen century: non-linearity (given by non-linear activation functions), and a much faster training speed facilitated by Stochastic Gradient Descent and modern computers. Neural Networks have

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

been successful in solving many practical problems, from image classification [140] to Natural Language Processing tasks such as speech recognition [141].

To understand the basics of how Neural Networks work, it is useful to start with a simple network consisting only of three layers: an input layer, an intermediate layer, and an output layer. In this example, the output layer outputs a mean μ and a standard deviation σ. As seen

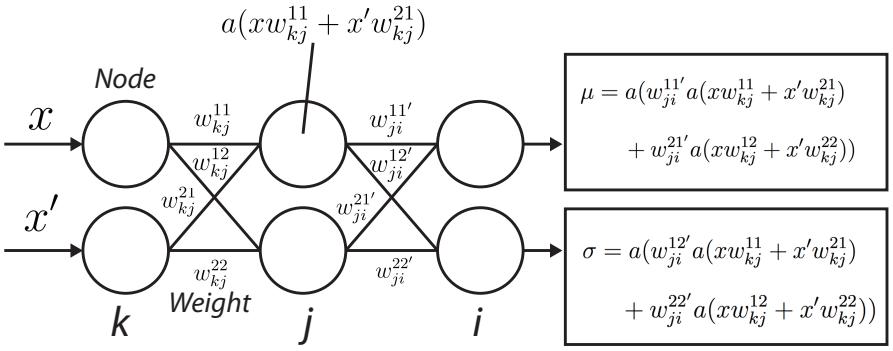


Fig. 2.3 Diagram of a three-layer Neural Network.

in Fig. 2.3, the input to the Neural Network (formed by the pair (x,x')) is transformed into an output μ, σ through the weights w_{ij} and the activation functions a. Note how from a very simple network with just six neurons, the output expressions are already quite complicated. This gives a good idea of the expressiveness of Neural Networks, that relies on the large number of inter-neuronal connections.

In prognostics, the non-linear model represented by the Neural Network is the function $f_w(x_{0:t})$ giving the parameters of the function $f_i(T, x_{0:t})$ at each time-step t. Once the activation functions (transformations performed at each neuron of the Neural Network) and network architecture are chosen, $f_w(x_{0:t})$ is fully determined by the edge weights $w_{ij} \in w$.

The weights in a Neural Network can be updated by different algorithms, normally based on a concept known as gradient descent. Gradient descent takes its name from the property of the gradients of a continuous, derivable function γ, for which local minima, maxima, and saddle points fulfil the following condition:

$$\nabla\gamma(x) = 0. \tag{2.15}$$

In which ∇ is the gradient operator. As described in the previous section, supervised machine learning methods are usually set up as minimisation problems. Thus, Neural Networks are frameworks set up to minimise a loss function l so that:

$$\nabla l(w_{jk}) = 0. \tag{2.16}$$

Where we have used w_{jk} to make clear that the optimisation objective of a Neural Network framework is to find the appropriate edge weights. The primary target of any gradient descent algorithm is to locate the direction in which l decreases the fastest. This can be achieved by finding the unitary vector $\vec{u}$ across which the directional derivative minimises. From multivariate calculus, we know that the directional derivative is:

$$\nabla_{\vec{u}} l = \vec{u} \nabla l. \tag{2.17}$$

To find the direction in which l *decreases* the fastest, we have to obtain the *minimum* gradient.

$$\min_{\vec{u}} \nabla_{\vec{u}} l = \min_{\vec{u}} |\vec{u}||\nabla l| \cos\theta = |\nabla l| \min_{\vec{u}} \cos\theta. \tag{2.18}$$

Where θ is the angle between the gradient and the vector $\vec{u}$, we know that its cosine minimises when they oppose each other. Thus, the direction $\vec{u}$ that minimises l is the opposite direction to the gradient, hence the term gradient descent. This method iteratively updates the weights of the neural network using the following formula:

$$\vec{w}' = \vec{w} - \mathrm{lr} \nabla l(\vec{w}), \tag{2.19}$$

in which lr is known as the learning rate. Training a Neural Network comprehends two steps: first, l is calculated as a function of the predictions of the Neural Network $f_w(x_{0:t})$. This is known as forward propagation.

Second, the gradients of the loss function l with respect to the weights $\vec{w}$ are calculated ($\nabla l(\vec{w})$). In Neural Networks, this is known as back propagation. The purpose of back propagation is to translate changes in the loss function into changes in the weights of the Neural Network. The derivation of back propagation is one of the best ways to give further context to two important parameters: the learning rate lr, and the momentum η. The derivation included here is inspired by the publicly available derivation by Dr. J. G. Makin [142].

First, let's revisit Fig. 2.3. Note how the layers in the Neural Networks are named in inverse alphabetical order, in this case: k, j, i. From now on, w_{kj} will refer to the edge weights between the layers k and j in the Neural Network. Note that the super-indexes, referring to the specific pairs of interconnected neurons have been dropped for simplicity. The input to any neuron of the layer j of the Neural Network is:

$$x_j = \sum_{k \in K_j} w_{kj} y_k, \qquad y_k = a(x_k). \tag{2.20}$$

Where a is the activation function (for the sake of this derivation it will be assumed to be a unitary sigmoid across all layers: $f(z) = \frac{1}{1-e^{-z}}$). The sigmoid is chosen because it has a simple derivative, but this derivation can be extended to any activation function.

Let's come back to what we want to calculate: how to connect weight changes with changes in the loss function. For the sake of simplicity we choose the mean square error loss function, eq. (2.1). Assume that j is the output layer of the Neural Network. In this loss, we rename $y_j \equiv \mu\left(x^n_{0:t}\right)$ and $y \equiv y^n_t$. We take the chain rule (remember that we are focusing on the layer j):

$$\frac{\partial l}{\partial w_{kj}} = \frac{\partial l}{\partial y_j}\frac{\partial y_j}{\partial x_j}\frac{\partial x_j}{\partial w_{kj}} = (y_j - y)\frac{\partial y_j}{\partial x_j}y_k = (y_j - y)y_j(1-y_j)y_k. \tag{2.21}$$

Some of the partial derivatives in eq. (2.21) are trivially obtained from eqs. (2.1) and (2.21), as in this case the loss function directly depends on the output of the last layer. If one calculates the change on the weight produced by only one training sample, then $\frac{\partial l}{\partial y_j} \propto (y_j - y)$ (constant terms are dropped as they do not matter for the optimisation). If we recall that $y_k = a(x_k)$, and that the activation function is the sigmoid function, which has the property $a' = a(1-a)$. It follows that $\frac{\partial y_j}{\partial x_j} = y_j(1-y_j)$.

To implement this term into an algorithm one must define how will this gradient change will be used to approach the loss minima. From eq. (2.19):

$$\Delta w_{kj} = -\mathrm{lr}\frac{\partial l}{\partial w_{kj}}. \tag{2.22}$$

The learning rate $\mathrm{lr} \in (0,1]$ is a crucial parameter when training Neural Networks. A large learning rate means that a small change in the loss function translates on a large change in the weights of the Neural Network, and a small learning rate means the opposite. Normally, one starts with large learning rates, and switches to smaller learning rates as the training of the Neural Network progresses.

Up until now, j has been assumed to be the output layer of the Neural Network, and as such the result that has been obtained only applies to the weights of the layer immediately before it. To generalise to the whole depth of the Neural Network we must assume that j is one of the intermediate layers in the Neural Network, also known as *hidden* layers.

The first equality in eq. (2.21) still holds. However, the first partial derivative $\frac{\partial l}{\partial y_j}$ must be re-calculated. In this case, we calculate the change in the loss function produced by a small change in y_j (the output of one of the neurons of our layer). To do so, we need to propagate *onwards* from the j layer to the next i_{th} layer. Thus, we need to propagate the changes that

this output produces on the inputs of all the neurons of the i_{th} layer: I_i:

$$\frac{\partial l}{\partial y_j} = \sum_{I_i} \frac{\partial l}{\partial y_i} \frac{\partial y_i}{\partial x_i} \frac{\partial x_i}{\partial y_j} = -\sum_{I_i} \delta_i w_{ji}. \tag{2.23}$$

Where δ_i is the error term, defined as $\delta_i := \frac{\partial l}{\partial y_i} \frac{\partial y_i}{\partial x_i}$, a product of the two first partial derivatives, already calculated before (just change j for i, as now i is the output layer). The term w_{ji} comes from deriving eq. (2.20). Recovering eq. (2.21), the following equation is obtained:

$$\frac{\partial l}{\partial w_{kj}} = -\sum_{i \in I_j} \delta_i w_{ji} y_j (1 - y_j) y_k. \tag{2.24}$$

An attentive reader will notice that this differs from the output layer result eq. (2.21). However, both differences can be "buried out" by using the definition of the error term within eq. (2.21):

$$\frac{\partial l}{\partial w_{kj}} = -\delta_j y_k. \tag{2.25}$$

Which combined with eq. (2.22) gives the change in the weight.

$$\Delta w_{kj} = \mathrm{lr} \delta_j y_k. \tag{2.26}$$

To implement this algorithmically, one normally wants to iteratively update the changes in the weights of the Neural Network. This is done by means of *epochs*, loops through all the training examples in between of which one usually updates the learning rate lr. The following equation is often used to update the weights transferring information from one epoch to another.

$$\Delta w_{kj}(n) = \mathrm{lr} \delta_j y_k + \eta \Delta w_{kj}(n-1). \tag{2.27}$$

Here n is the epoch, and $\eta \in [0,1)$ is a parameter called momentum, that signals how much of the change in the weight in this epoch will be influenced by the change that it underwent in the last epoch.

Calculating δ_j, especially in the case of hidden layers, is often a computationally demanding task. A key approach to reduce this computational demand (and thus increase training speed) has been the Stochastic Gradient Descent algorithm. The main insight of Stochastic Gradient Descent is that the gradient of a function is an expectation, and that this expectation can be estimated by using just a subset of the samples in the training data-set [143].

As shown in Section 2.2.2, the loss function of a machine learning algorithm can often be written as a sum over training examples. If this is the case, its gradient can also be

de-composed in the same way. Stochastic Gradient Descent proposes that at every step of the algorithm (every update of the weights) a subset of the training examples is used to calculate the average gradient. If l is the per-example loss $l(x^n_{0:t}, y^n_t, w) = -\log \Pr_w \left(Y^n_t = y^n_t | x^n_{0:t}\right)$ of the output y given the input $x^n_{0:t}$ and the weights w, the gradient of the log-likelihood is:

$$\nabla_w \log(\mathscr{L}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_w l(x^n_{0:t}, y^n_t, \vec{w}). \tag{2.28}$$

This subset m, known as minibatch or batch[10], is drawn uniformly from the training set. If $m = 1$, this procedure is known as *online* training.

2.2.4 Back propagation through time

The derivations included in the last section were given in the context of Artificial Neural Networks, non-linear functions that map a feature input into an output. This thesis focuses on predicting when a machine will fail from a sequence of sensor values, and thus the computational tools of interest are those that are designed to learn sequences of values. Neural Networks can be adapted to do so, in the form of what is known as Recurrent Neural Networks.

This section presents a summarised derivation of back propagation through time in Recurrent Neural Networks. Among other things, this derivation is important to understand the dimensionality of their free parameter space. Most of the derivation is adapted from [143].

A Recurrent Neural Network stores its sequential information in a hidden state $\vec{h}(t)$ that depends on the hidden state at $t-1$, on the input variable $x^n_{0:t}$, and on the weights of the Neural Network $\vec{w}$:

$$\vec{h}(t) = f(\vec{h}(t-1), x^n_{0:t}, w). \tag{2.29}$$

It follows that $\vec{h}(t-1)$ will depend on $\vec{h}(t-2)$, dependency that can be extended recursively. This means that $\vec{h}(t)$ depends on the values of the feature $x^n_{0:t}$ at all times smaller than t. This is exactly what one wants when learning sequences: for the sequence to depend on its values in the past. The output of a RNN then reads from the hidden state through an output layer. In Recurrent Neural Networks, the dimensionality of $\vec{h}$ is fixed, and thus we are building a mapping from all the previous values of the feature space, $x^n_{0:t}$, to a fixed-dimension vector $\vec{h}$. A higher dimensional hidden space can store more information from the original features, but also increases the number of free parameters in the network.

[10] In this thesis, batch is used, although batch sometimes refers to the full training set [143].

A Recurrent Neural Network has essentially three kinds of weights: the weights parametrising the connection of the hidden state with the input features, represented by the matrix U, the weights parametrising the connection between hidden states (carrying temporal information), represented by the matrix W, and the hidden-to-output weights parametrised by the matrix V (see Fig. 2.4). U and V have the same dimensionality than the weights connecting two dense layers in an artificial Neural Network.

If the hidden state has N_u neurons and the input has K features, the number of free parameters added by dense layer would be KN_u. If we add here the weights of W connecting each element in the hidden state with its previous component and the bias weights, a Recurrent Neural Network layer will add $KN_u + N_uN_u + N_u = N_u(K + N_u + 1)$ free parameters to the complexity of a Neural Network[11]. N_u is the number of bias weights (one for each hidden state neuron). Bias is an added scalar value to the output of a neuron that allows for the non-linear function represented by the neuron to be displaced from origin.

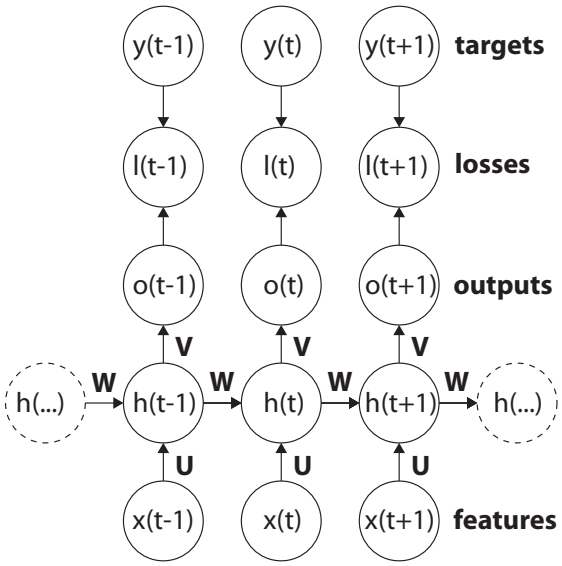


Fig. 2.4 Sketch of a Recurrent Neural Network and its weight matrices (modified from [143]).

To understand how a Recurrent Neural Network is trained, it is useful to derive first the forward propagation (the propagation from the training examples to the loss). This derivation assumes that the outputs are discrete (for example, words or discrete numbers). A hyperbolic tangent is assumed as the activation function for the hidden units. As often done for discrete predictions, the output $\vec{o}$ is regarded as giving the log probabilities of each possible value of the discrete target variable. Then, this is smoothed through a softmax operation to get a

[11] Or $MKN_u + N_uN_u + N_u$ if the dimensionality of the output layer, M, is considered.

vector $\vec{y}$ of normalised probabilities. In a Recurrent Neural Network, forward propagation is done from an initial hidden state $\vec{h}(0)$. For each time step from $t = 1$ to $t = T_p$ (the time at which the prediction is done), the following update equations are iteratively computed:

$$\vec{a}(t) = \vec{b} + W\vec{h}(t-1) + U\vec{x}(t), \tag{2.30}$$

$$\vec{h}(t) = \tanh(\vec{a}(t)), \tag{2.31}$$

$$\vec{o}(t) = \vec{c} + V\vec{h}(t), \tag{2.32}$$

$$\vec{y}(t) = \text{softmax}(\vec{o}(t)). \tag{2.33}$$

Here, notation is simplified. $\vec{x}(t)$ is x_t^n, and $x_{0:t}^n = \vec{x}_{0:t}$ (for a given n,t). Note how the temporal information is saved in the hidden space, and how the weight matrices W,V, and U are shared across the temporal dimension (W is the same for every t). $\vec{b}$ and $\vec{c}$ are bias vectors. The loss for a sequence of $\vec{x},\vec{y}$ pairs is then simply the addition of the temporal losses. Note how for each time (t), the entire sequence of $\vec{x}_{0:t}$ is incorporated into the model:

$$l\left(\vec{x}_{1:T},\vec{y}_{1:T_p}\right) = \sum_{t=1}^{T_p} l(t) = -\sum_{t=1}^{T_p} \log \Pr_{\text{model}}\left(y(t)|\vec{x}_{1:t}\right). \tag{2.34}$$

Here $y(t)$ is used as it corresponds to the entry for $\vec{y}(t)$ corresponding to the desired output in the training example (recall that each discrete output is coded into a binary vector as in a logistic regression problem). This means that the Neural Network goes over the sequence as if it would be discovering new data in each time-step.

Calculating the gradient in a Recurrent Neural Network is computationally demanding but simple from a theoretical perspective. In essence, the same back propagation procedure described before can be used. Here, back propagation is derived using the network's computational graph[12].

The nodes of the RNN computational graph include the bias vectors $\vec{b}$ and $\vec{c}$ as well as the weight matrices U, V and W, and the sequential nodes $\vec{x}(t)$, $\vec{h}(t)$, $\vec{o}(t)$ and l (the loss given a sequence). For each of these nodes γ, one needs to calculate the gradient of the loss function with respect to them; $\nabla_\gamma l(t)$. Two nodes do not need further calculation: the gradient with respect to the loss $l(t)$ (which is always 1), and the gradient with respect to the inputs $\vec{x}(t)$ as they have no parameters preceding them in the computational graph. This derivation assumes that the loss is the negative log-likelihood of the target $y(t)$ given the input $\vec{x}_{0:T_p}$.

[12]To propagate gradients in Recurrent Neural Networks one often uses computational graphs: a tool to graphically formalise a set of computations. Computational graphs are mathematical representations in their own right, and should not be interpreted as just schematics [143].

Let's start with the gradient on the outputs $\vec{o}(t)$, for each output component o_i:

$$\begin{aligned}\left(\nabla_{\vec{o}(t)} l\right)_i &= \frac{\partial l}{\partial o_i(t)} = \frac{\partial l}{\partial l(t)}\frac{\partial l(t)}{\partial o_i(t)} = 1\frac{\partial l(t)}{\partial o_i(t)} = -\frac{\partial \log \Pr_{\text{model}}\left(y(t)|\vec{x}_{0:t}\right)}{\partial o_i(t)} \\ = -\frac{\partial \log \text{softmax}(\vec{o}(t))}{\partial o_i(t)} &= -\frac{\text{softmax}(\vec{o}(t))}{\text{softmax}(\vec{o}(t))}\left(\delta_{i=y(t)} - \text{softmax}(o_i(t))\right) = (\hat{y}_i(t) - \delta_{i=y(t)}).\end{aligned} \tag{2.35}$$

Here $\delta_{i=y(t)}$ can be understood as: if y is actually the correct discrete number, the Kronecker delta collapses (remember that the target for each training example is a vector with 1 signalling the correct word or discrete number). $\hat{y}_i(t)$ is used to signify an element of the output vector $\vec{y}(t)$ (not to be confused with $y(t)$, representing the true value of the target). To back propagate through time, one must work its way from the end of the sequence to its origin. At the end of the sequence, the hidden state $\vec{h}(t)$ only has $\vec{o}(t)$ as a descendent, which simplifies the gradient:

$$\nabla_{\vec{h}(T_p)} l = \left(\frac{\partial \vec{o}(t)}{\partial \vec{h}(t)}\right)^{\top} \nabla_{\vec{o}(T_p)} l = V^{\mathsf{T}} \nabla_{\vec{o}(T_p)} l. \tag{2.36}$$

In this equation, the rule of chain for gradients has been used. Here $\frac{\partial \vec{o}(t)}{\partial \vec{h}(t)}$ is the Jacobian matrix for $\vec{o}$ with respect to the components of $\vec{h}$, that can be obtained trivially from eq. (2.32). Now, if one iterates a further time step backwards $t = T_p - 1$, the hidden space vector has both a hidden space descendent $\vec{h}(t+1)$ and the output $\vec{o}(t)$. Its gradient is thus given by the chain rule:

$$\nabla_{\vec{h}(t)} l = \left(\frac{\partial \vec{h}(t+1)}{\partial \vec{h}(t)}\right)^{\mathsf{T}} \nabla_{\vec{h}(t+1)} l + \left(\frac{\partial \vec{o}(t)}{\partial \vec{h}(t)}\right)^{\mathsf{T}} \nabla_{\vec{o}(t)} l. \tag{2.37}$$

To calculate the first summand of this gradient, one needs to take into account that the hidden units feed to each other through a hyperbolic tangent activation function (and the weight matrix W, see eq. (2.32)). Once computed, this gradient gives:

$$\nabla_{\vec{h}(t)} l = W^{\mathsf{T}} \text{diag}\left(1 - (\vec{h}(t+1))^2)\right) \nabla_{\vec{h}(t+1)} l + V^{\mathsf{T}} \nabla_{\vec{o}(t)} l, \tag{2.38}$$

where $\text{diag}\left(1 - (\vec{h}(t+1))^2)\right)$ indicates the diagonal matrix with diagonal components $1 - (h_i(t+1))^2$. Now that the gradients on the internal nodes of the Recurrent Neural Network have been obtained, the gradients with respect to weights and other parameters can be

obtained following a similar procedure. A list of expressions for the gradients with respect to all different weight matrices and bias vectors $\vec{b}$ and $\vec{c}$ can be found in [143].

2.2.5 Long Short-Term Memory variant

The Long Short-Term Memory (LSTM) variant is designed to adapt Recurrent Neural Networks to long time-dependences [144]. Simple Recurrent Neural Networks such as the one shown in the last section include a recurrent multiplication of the matrix W to the sequence of previous hidden states, so that for no input $\vec{x}$ the hidden state $h(t)$ depends on $h(0)$ as:

$$\vec{h}(t) = (W^t)^\top \vec{h}(0). \tag{2.39}$$

Here we assume a linear activation function. It is easy to see through eigenvalue decomposition that for large t the eigenvalues of $(W^t)^\top$ will either decay to zero (for eigenvalues smaller than 1), or explode [143]. This is known as the vanishing/exploding gradient problem. Long Short-Term Memory Recurrent Neural Networks are specifically designed to address this problem. In order to do so, LSTMs include a memory cell that maintains its state over time. This cell is accessed through additional hidden units (known as gates) with non-linear activation functions [144]. These gates are specially conceived to maintain the state of the memory cell along the training sequences.

There are different variants of the LSTM architecture, normally defined as whether some of the elements of the memory cell are missing (for example, the input gate, the output gate, the forget gate, etc. [144]). This section introduces what is known as the *vainilla* variant. This corresponds to a recurrent block with input, output and forget gates, all of them accessed by the input features and the output of the previous LSTM block (see Fig. 2.5). At first sight, Fig. 2.5 can appear a bit overwhelming. To understand it better it is helpful to discuss how it compares to a Recurrent Neural Network. U_i are weight matrices that multiply the input akin to matrix U in Fig. 2.4, thus they have dimension $N_u \times K$ where N_u is the number of LSTM blocks. R_i are weight matrices that recursively multiply the outputs similar to matrix W in Fig. 2.4, they also share its dimensionality: $N_u \times N_u$. The difference here is that there are four matrices of each type, and that there are time-recursive connections inside the LSTM block that allow for it to address the problem of exploding and vanishing gradients.

Technically, LSTMs do not have simple hidden units as RNN have. When in this thesis the size of the hidden state is referred to what is meant is the number of LSTM blocks (dimensionality N_u in the equations below). Vainilla LSTM layers have four times more free

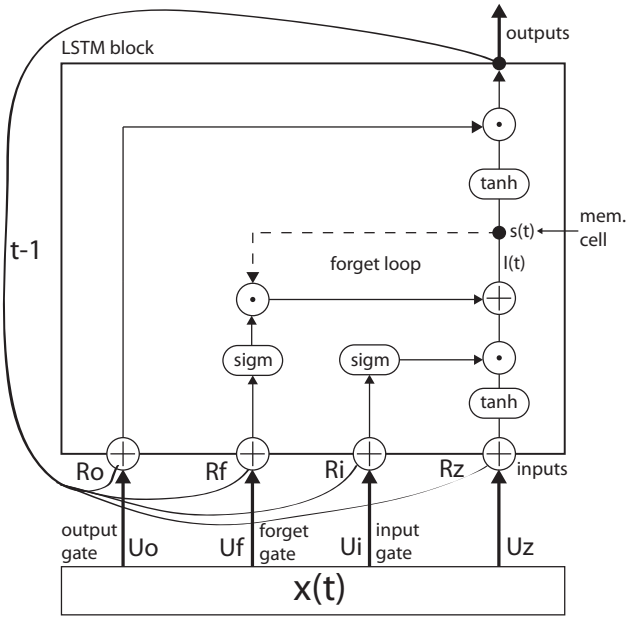


Fig. 2.5 Diagram of a LSTM Recurrent Neural Network and its weight matrices. Inside the block, dashed lines signify time-delayed connections

parameters than a RNN. Therefore, the number of free parameters added by the LSTM layer is $4N_u(K+N_u+1)$.

Effect on the gradients

Once RNNs are understood, the LSTM case conforms to similar mathematical derivations. Like in RNNs, computational graphs are used to calculate the gradients, that then can be used in a conventional Stochastic Gradient Descent optimiser. A full description of the equations for forward and back propagation is included in [144]. This section is limited to show the most important property of LSTMs: the solution of the vanishing/exploding gradient problem (taken from [145]).

To understand how LSTMs are able to learn long sequences, the one-dimensional case is chosen as it allows for a more direct comparison with Recurrent Neural Networks. Assume no input and no bias. This time, a sigmoid activation function σ is used. In a conventional Recurrent Neural Network, the hidden space at $h(t)$ depends on $t-1$ as:

$$h(t) = \sigma\left(wh(t-1)\right). \tag{2.40}$$

Where w is a scalar weight (remember, this is the one-dimensional case). The gradient at a time step t' with respect to an arbitrary $t < t'$ is:

$$\frac{\partial h(t')}{\partial h(t)} = \prod_{k=1}^{t'-t} w\sigma'\left(wh(t'-k)\right) = w^{t'-t}\prod_{k=1}^{t'-t}\sigma'\left(wh(t'-k)\right). \tag{2.41}$$

This is essentially a one dimensional version of eq. (2.39) for a non-linear activation function and once the derivative has been taken. Again, the term $w^{t'-t}$ appears, representing an exponential decrease (or increase) of the gradients.

In a one-dimensional LSTM case with no inputs and biases, the derivative of the memory cell state with respect to itself depends only on the input to the forget gate. From Fig. 2.5, it is easy to see that the input to the cell is:

$$s(t) = s(t-1)\sigma(v(t)) + I(t). \tag{2.42}$$

Where $I(t)$ is the input to the memory cell from the input gates. $v(t)$ is the input to the forget gate. When it is derived with respect to a previous time, for example $(t-1)$, the multiplying term $s(t-1)$ vanishes, and one obtains:

$$\frac{\partial s(t')}{\partial s(t)} = \prod_{k=1}^{t'-t}\sigma'\left(v(t'+k)\right). \tag{2.43}$$

Thus, the exponential term has disappeared (compare with eq. (2.41)). Granted, any set of weights differing from $\sigma'\left(v(t'+k)\right) \approx 1$ will also make the gradients vanish or explode for sufficiently long sequences. However, this variation is much slower than in the case of Recurrent Neural Networks. This means that LSTMs suffer from vanishing/exploding gradients but their effect appears at much larger time-scales.

With this, the theoretical foundations of this thesis terminate. Much of what has been left out can be consulted in Godfellow's book on deep learning [143].

2.3 Conclusion

This chapter presents the literature review and theoretical background upon which this thesis is based. In the literature review, three fields of research are reviewed: asset health and performance management, and the applications of machine learning and Multi-Agent Systems for prognostics. In the theoretical background, the prognostics problem is defined and the statistical methodology used to obtain prognostics estimates is presented.

The literature review leads to a clear conclusion: the technologies needed to implement Distributed Collaborative Prognostics exist, but they have not yet been combined into a comprehensive solution. Real-time machine learning frameworks able to convert multi-sensor readings into time-to-failure estimates are available. Similarly, the theoretical and architectural foundations of Multi-Agent Systems and other distributed systems have been developed, and there is no shortage of papers claiming their potential benefits in prognostics and asset management in general. What is missing is a combination of these technologies that effectively solves the prognostics problem, and enables assets with collaborative and dynamic capabilities.

Some work in the literature has attempted to bridge this gap. Much of it concerns small fleets of assets (on the order of less than ten individuals), fails to experimentally demonstrate its usefulness, or can't be transferred across different industrial scenarios. What is more important, most of the published work does not incorporate inter-agent collaboration thus falling short from a true multi-agent or holonic framework (see [23] by the author for an in-depth comparison of existing implementations and Distributed Collaborative Prognostics). This is a crucial shortcoming, as real-time collaboration is one of the key properties of an agent, and is important for it to operate in situations of dynamism and non-ergodicity.

Leaving aside the literature review, the theoretical background available to produce high-quality prognostics draws from regression analysis. This chapter describes some well-established models, based on parametrising the probability distribution of the regression targets. Additionally, this chapter presents an introduction to the frameworks used to perform regression: Neural Networks. Special attention is put into Recurrent Neural Networks and their LSTM variant, designed to learn long sequences of values.

Chapter 3

Distributed Collaborative Prognostics: properties and architectures

This chapter presents the principal functionalities and characteristics of Distributed Collaborative Prognostics deemed necessary to address this thesis' problem statement, and discusses how they compare with the properties of Advanced Multi-Agent Systems. In doing so, this chapter addresses the first of this thesis' research questions by drawing a link between the properties of Distributed Collaborative Prognostics and those of Multi-Agent Systems.

This chapter is composed of four sections. In the first section, the properties of Distributed Collaborative Prognostics are presented, justifying the use of Multi-Agent Systems for its implementation. In the second and third sections the building blocks and architectures used later in this thesis are described, based on four different canonical multi-agent architectures. The last section contains the conclusions of the chapter.

3.1 Properties

This section describes the functions that Distributed Collaborative Prognostics must be able to perform, together with the characteristics that it has to have in order to successfully address this thesis' problem statement: to devise and implement a prognostics tool able to operate in the conditions of ergodicity and dynamism typical of industrial asset fleets.

The properties of Distributed Collaborative Prognostics will be divided in two concepts: functionalities, referring to what the tool must be able to *do*, and characteristics, referring to what the tool must *be*.

Functionalities

The functionalities of Distributed Collaborative Prognostics are:

- *Prognostics*: some of the tool's components must be able to provide prognostics in real time for each of the assets in the fleet.
- *Collaboration*: some of the tool's components must be able to communicate with each other, and improve prognostics thanks to this communication.
- *Smartness*: some of the tool's components must be adaptable to the execution of other functionalities besides prognostics and collaboration.

Collaboration here is defined as any of the cooperation techniques defined in Ming Tan's prominent paper on cooperation [25], i.e. sharing data, models, or the combination of the two. Collaboration is important for several reasons. Without collaboration, data from one asset cannot be leveraged into the prognosis of another asset. This would render the training of machine learning algorithms in a distributed system impossible. Collaboration also allows for dynamically weighting data from different assets in the prognostics algorithms as the conditions of the assets and the environment change. This is currently not possible in a centralised approach, and is crucial in order to adapt to the dynamic and non-ergodic properties of real asset fleets.

Prognostics is important for obvious reasons, as it is the ultimate purpose of the tool. What requires more comment is why diagnostics is not included as one of the tool's fundamental functionalities. The reason for this is that the tool presented in this thesis is based on a rigid definition of diagnostics: for machine-learning based regression to be possible, failures (training examples) have to have a consistent definition across the assets. Therefore, it is not clear that the same framework that works for distributed prognostics should be extended to diagnostics. Rather, a centralised static rule that determines when an asset has failed is preferable.

Smartness refers to the possibility of extending the tool to other asset management tasks such as maintenance policy optimisation, operations control, inventory decisions, etc. This thesis will demonstrate this property by extending the tool to maintenance policy optimisation.

Characteristics

In order to operate in dynamic and non-ergodic systems, Distributed Collaborative Prognostics must have a specific set of characteristics. The characteristics of Distributed Collaborative Prognostics are:

1. *Distribution*: the components forming this tool must be distributed. Each asset of the fleet has to be assigned to at least one of the tool's components.

2. *Scalability*: the tool must scale to as many independent components as the number of assets in the fleet that it operates.

3. *Transferability*: the tool must be designed in such a way that it can be easily implemented in different kinds of assets.

4. *Resilience*: the tool must be resilient to failures in its components.

Distribution is crucial in order to provide each asset with a certain degree of autonomy. Systems featuring distributed components such as holons, Multi-Agent Systems, etc. are designed to operate in situations in which many independent units have to be managed (see Chapter 2). Thus, they all can be considered as candidate frameworks to implement Distributed Collaborative Prognostics.

Scalability is motivated by the fact that the size of asset fleets has increased steadily in the last years, as fewer corporations control larger portions of the market share [146]. This means that any solution that does not scale is not prepared to function in many realistic modern-day industrial scenarios.

This thesis aims to provide a solution applicable to different industrial fleets of assets. Therefore, the solution must be easy to transfer across industries. Transferability benefits from components designed in such a way that they can be adapted to different problems with minimal or no change in their internal structure.

Resilience is often a forgotten piece of asset management solutions. Sometimes, it is possible that a tool devised to improve the reliability of an asset actually ends up generating a whole set of reliability problems itself. A good example is the emerging problem of sensor faults in some Condition Monitoring systems [147]. Distribution Collaborative Prognostics aims to avoid this problem by creating a tool that is resilient to the failure of its components.

Connection to Multi-Agent Systems

The properties described so far are encompassed by the properties of an existing distributed framework: Multi-Agent Systems. There are many definitions of the properties of Multi-Agent Systems, most of them including the characteristics described above either explicitly or implicitly. In this thesis, Advanced Multi-Agents Systems are chosen as the theoretical framework in which the tool must be developed.

This choice is made for two reasons: 1) Advanced Multi-Agent Systems have been proposed as an improved version of Multi-Agent Systems, in which the capabilities of

modern agents can be leveraged [22, 23]. This means that the elements in the tool (agents) are able to collaborate, and perform prognostics and other analytical tasks [22]. 2) The properties of Advanced Multi-Agent Systems overlap near perfectly with the characteristics of Distributed Collaborative Prognostics introduced in the last section.

An Advanced Multi-Agent System can be understood as a Multi-Agent System that fulfils the following properties [22, 23].

- Distribution: the Multi-Agent System architecture must have a decentralised structure. Optimisation, decision-making and control are performed by the agents in the system.
- Flexibility: the system must be able to adapt in real-time to changes in the agents and the environment.
- Adaptability / System learning: thanks to the real-time learning capability of the agents, the output of the system is constantly updated and improved in reaction to human demands and changes in the environment.
- Scalability: the system must be easily scalable. New devices must be able to join the system without important changes in the architecture or the agents composing the Multi-Agent System.
- Leanness: an Advanced Multi-Agent System should always strive towards minimizing the number of different categories of agents that it features. Differences between the agents should be kept small to obtain a swarm-like[1] behaviour.
- Resilience: the system should be designed so that, as much as possible, is resilient to agent faults.

Note how the functionalities of Distributed Collaborative Prognostics are not included in this description. This is because two of these functionalities (Collaboration and Smartness) are already included in the definition of the word agent. Agents in this thesis are autonomous, problem-solving (smart), and goal-driven computational entities with social abilities (able to communicate).

The connection between the desired characteristics of Distributed Collaborative Prognostics and Advanced Multi-Agent Systems is evident in all cases except for the case of transferability. A Lean and Adaptable system is very often a Transferable system, as it is designed so that it can be operated in different scenarios. This connection is explicitly shown in this thesis, as the same Advanced Multi-Agent System is used for different industrial scenarios with little modification in its components.

[1] See [148] for a definition of swarm-like robot cooperation.

This section justifies the choice of Multi-Agent Systems as the framework in which Distributed Collaborative Learning will be developed. The following sections are dedicated to describe the multi-agent architectures that will be used in this thesis, together with their building blocks.

3.2 Building blocks

This section describes the building blocks of the multi-agent architectures used in this thesis. From hereupon, when a building block is referred to without further explanation, the reader can assume that its definition can be found here.

3.2.1 Virtual Asset

Virtual Assets are passive software elements designed with the sole purpose of processing the data coming from an industrial asset, standardising it, and sending it to higher-order agents. As such, each Virtual Asset is assigned to a physical asset, and is responsible to feed standardised real-time data to agents in higher layers of the architecture. This data is formed by three components: (i) time-series sensor data, (ii) failures or warnings with their corresponding time, and (iii) a unique asset identifier, indicating the asset that they are dedicated to.

Because of their simplicity and one-way communications, Virtual Assets fail to satisfy the definition of an agent provided in Section 2.1.4 [2]. However, they are a crucial building block of many of the experiments presented in this thesis, and thus they are described here.

Virtual Assets are formed by two building blocks: a Communications Manager, and a Standardiser (see Fig. 3.1). The first block manages communications with higher-order agents, and the second one is dedicated to standardise the data coming from the asset so that it can be understood by them. The specific manner in which these building blocks operate will vary depending on whether they are used in simulations or physical implementations (compare, for example, the Virtual Assets in Chapter 4 with the ones used in Chapter 5). Thus, a detailed description of the agents is reserved for each implementation.

An interesting property of Virtual Assets is that they can be used to emulate real machines in experimental scenarios. For example, say that one has access to a large data set of many industrial machines that have been operating for several years. Virtual Assets can be used to read the data set at constant time intervals, and feed data to their corresponding Digital Twins

[2] Autonomous, problem-solving, and goal-driven computational entities with social abilities.

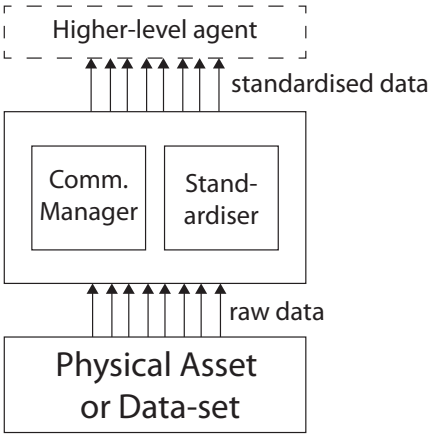


Fig. 3.1 Block diagram of a Virtual Asset, featuring its principal components.

such as if the machines were operating in real time. This allows to replicate the real-time behaviour of an industrial fleet and test different prognostics strategies in it.

3.2.2 Agents

This section describes the agents employed in this thesis. Each agent is composed of different building blocks, and has a specific position in each different architecture. In general, all the implementations presented in this work deal with at most three kinds of agents: Digital Twins, Mediator Agents, and a Social Platform.

High-level descriptions of each agent follow:

Digital Twin

Digital Twins are agents placed one level higher in the hierarchy than Virtual Assets. Like Virtual Assets, Digital Twins have a one-to-one correspondence with the assets in the fleet. Digital Twins are smart agents able to communicate with each other, and perform prognostics and data preprocessing. They also have the capability of computing differences between the assets that they are assigned to (more of this in Sections 5.3 and 6.3.1).

Digital Twins are composed of three building blocks: an Analytics Engine, a Communication Manager and a Data Repository (see Fig. 3.2). The Analytics engine runs the machine learning algorithms that the Digital Twin has been programmed to execute. The Communications Manager performs communication with other agents in the Multi-Agent System (including the agent capability to choose other agents to communicate with). The Data Repository hosts the data that the machine learning algorithms are trained upon.

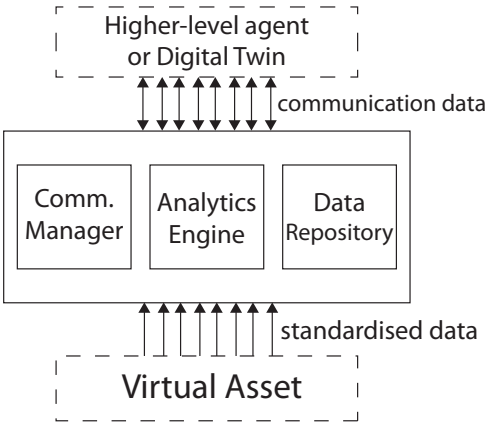


Fig. 3.2 Block diagram of a Digital Twin, featuring its building blocks.

Mediator Agent

As an intermediate agent, a Mediator Agent does not have a one-to-one correspondence with the assets in the fleet. In essence, this means that each Mediator Agent is in charge of several assets simultaneously. As such, Mediator Agents are able to determine maintenance policies for these assets, much like a Digital Twin does so for a single asset.

The composing blocks of a Mediator Agent are the same as those of a Digital Twin (see Fig. 3.3). The only difference being that their Communication Manager is more limited. Unlike Digital Twins, Mediator Agents cannot independently choose which other agents to communicate with, as this is decided by the Social Platform. Additionally, Mediator Agents are not allowed to perform direct peer-to-peer communications. Instead, they share data with each other through the Social Platform.

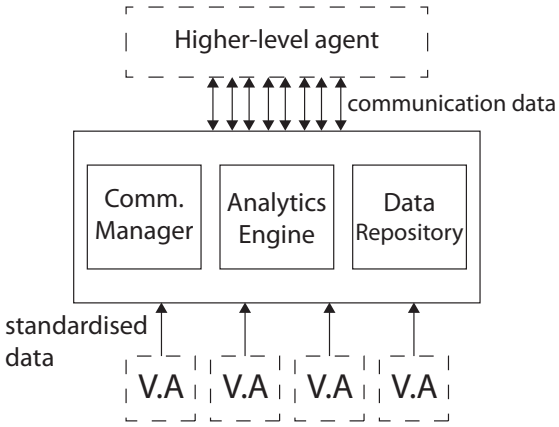


Fig. 3.3 Block diagram of a Mediator Agent, featuring its building blocks.

Social Platform

The Social Platform lies on top of the Multi-Agent System's hierarchy. Its role is to process information coming from the rest of the agents in the system. Using this information, it runs algorithms aimed at (1) forming groups of similar assets for collaborative learning, and (2) plotting whole-fleet information in a way that it can be understood by human operators. The Social Platform is also the agent that must be directly programmed to set parameters that concern the rest of the fleet of agents. This includes hyper-parameters for the training of the prognostics models in the Digital Twins, prediction limits, etc. In the case of a centralised architecture formed by a single agent, it can also calculate prognostics and give maintenance recommendations.

Apart from an Analytical Engine, the Social Platform also hosts a Communication Manager, responsible for the communications between the Social Platform and the rest of the assets in the fleet. In some cases, the social Platform is used to channel communication between agents that do not have peer-to-peer capabilities. Another reason to use the Social Platform as a communication platform is to retain control of the data flow within an enterprise (see Sections 5.2 and 7.5.1 for an industrial example for which this is required).

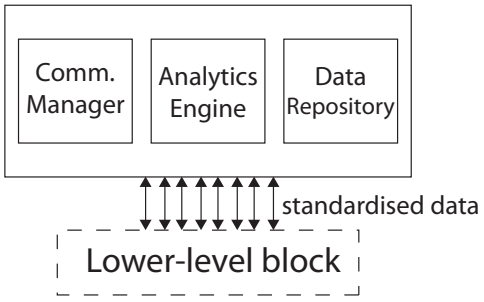


Fig. 3.4 Block diagram of the Social Platform, featuring its building blocks.

To manage the data related to the functions described above, the Social Platform has a Data Repository, that due to its privileged position in the hierarchy has to be endowed with substantially better capabilities than its Digital Twin's counterpart.

The building blocks of the Social Platform are shown in Fig. 3.4.

3.3 Architectures

This section addresses in part this thesis' first research question by exploring multi-agent architectures in a prognostics context (see Sec. 1.7 for the research questions). This PhD thesis focuses on four canonical Multi-Agent System architectures: Centralised, Hierarchical,

Heterarchical and Distributed. This section is dedicated to describe them, and how the agents featured in the preceding section fit within each of the architectures.

Table 3.1 shows a summary of the properties of each building block, the architectures that it belongs to and its position within the architecture.

Name	Agent	Layer	Components	Centr.	Hier.	Heter.	Distr.
Virtual Asset	N	3	Communication Manager Standardiser	Y	Y	Y	Y
Digital Twin	Y	2	Communication Manager Analytics Engine Data Repository	N	N	Y	Y
Mediator Agent	Y	2	Communication Manager Analytics Engine Data Repository	N	Y	N	N
Social Platform	Y	1	Communication Manager Analytics Engine Data Repository	Y	Y	Y	N

Table 3.1 Table summarising the Multi-Agent System building blocks and their position in the architectures. In here, Layer refers to the deepest layer in the hierarchy that any of the building blocks occupies in any of the architectures (see Fig. 3.5).

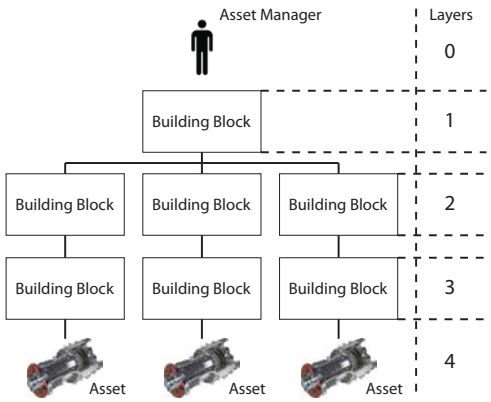


Fig. 3.5 Sketch of architecture layers and building blocks.

It is worth clarifying that in “Distributed Collaborative Prognostics", the term Distributed does not refer to any specific multi-agent architecture. Hierarchical and Heterarchical architectures are considered valid architectures for “Distributed Collaborative Prognostics", as they all portray a certain degree of distribution.

3.3.1 Centralised

Technically, the Centralised architecture is not a Multi-Agent Architecture, as it is composed only by one agent: the Social Platform. However, due to its simplicity and the high degree of data control that it offers, it is one of the most prolific architectures when it comes to implementation. Therefore, this architecture represents the minimum benchmark that any alternative architecture has to surpass if it wants to be considered as a viable solution.

In its strictest form, the Centralised architecture is formed only by the Social Platform and a set of Virtual Assets that send information to the Platform (see Fig. 3.6). In this case, the Centralised architecture does all the tasks of the system except data standardisation. This means, prognostics, maintenance policy recommendation, etc.

In this thesis, a Centralised architecture is defined as a set of Virtual Assets sending data to the Social Platform that then processes this data. In many centralised implementations, this is not the case and prognostics is performed by a piece of software that is not an agent.

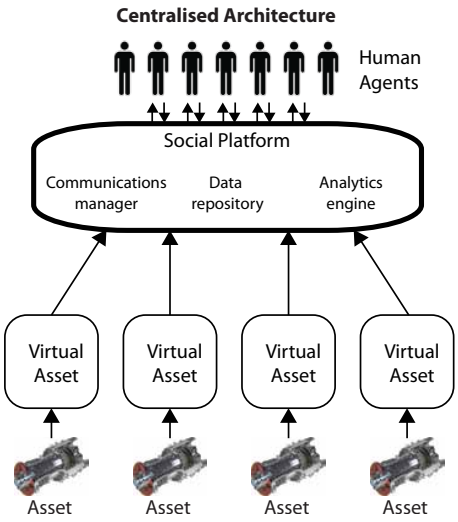


Fig. 3.6 Block diagram of the Centralised architecture. Black arrows indicate communication between components. The Social Platform is represented by a thicker block to indicate that it is the agent responsible for prognostics.

3.3.2 Hierarchical

A Hierarchical architecture is usually described as an architecture in which Mediator Agents perform the smart tasks of the system [149]. In this architecture, the lower level the agents are in the hierarchy, the simpler the tasks that they are assigned to perform.

In Distributed Collaborative Prognostics, hierarchical architectures are formed by a Social Platform, Mediator Agents, and Virtual Assets. Maintenance policy decisions and prognostics

are left to the Mediator Agents, while communications are managed by the Social Platform, that serves as a communication link between Mediator Agents.

In this kind of architecture, the Social Platform assigns groups of collaborating assets to each Mediator Agent, and the Mediator Agents compute prognostic models for the assets assigned to them. The Social Platform can create and delete Mediator Agents if the number of asset groups that it computes for the fleet varies (see Fig. 3.7).

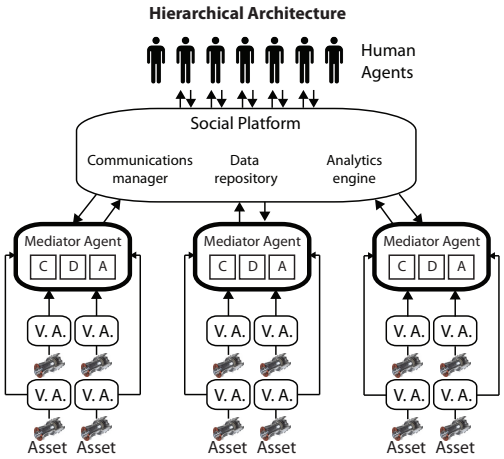


Fig. 3.7 Block diagram of the Hierarchical architecture. Black arrows indicate communication between components. The Mediator Agents are represented by thicker blocks to indicate that they are the element responsible for prognostics. C, D, A are used to indicate the Communications manager, the Data Repository and the Analytics engine.

3.3.3 Heterarchical

A Heterarchical architecture is formed by Virtual Assets, Digital Twins and a Social Platform. In this case, prognostics and maintenance recommendations are performed by the Digital Twins, who are also allowed to communicate directly with each other (see Fig. 3.8).

In this architecture, the role of the Social Platform is limited to forming groups of collaborating assets, processing fleet-wide data, and conveying human requirements to the rest of the Multi-Agent System.

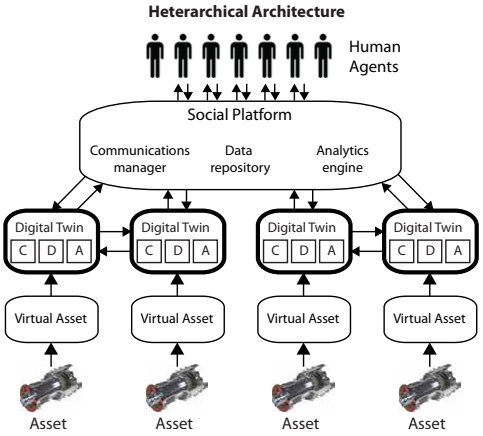


Fig. 3.8 Block diagram of the Heterarchical architecture. Black arrows indicate communication between components. The Digital Twins are represented by thicker blocks to indicate that they are the agents responsible for prognostics. C, D, A are used to indicate the Communications manager, the Data Repository and the Analytics engine.

3.3.4 Distributed

A Distributed architecture is formed only by agents that have a one-to-one correspondence with the assets in the fleet. In other words, it exclusively employs Virtual Assets and Digital Twins (see Fig. 3.9). Communications in this architecture are limited to peer-to-peer communications between the Digital Twins. The Digital Twins perform all the functionalities of Distributed Collaborative Prognostics: collaboration, prognostics, and maintenance recommendations (if desired). In this case, Digital Twins communicate with human agents directly without the need of an intermediate agent.

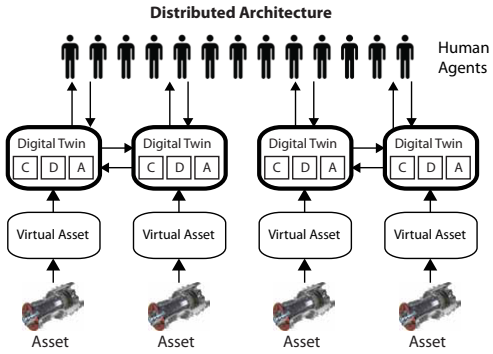


Fig. 3.9 Block diagram of the Distributed architecture. Black arrows indicate communication between components. The Digital Twins are represented by thicker blocks to indicate that they are the agents responsible for prognostics. C, D, A are used to indicate the Communications manager, the Data Repository and the Analytics engine.

3.4 Conclusion

This chapter describes the basic properties of Distributed Collaborative Prognostics, and justifies the idoneity of using Multi-Agent Systems as its distribution framework. Among Multi-Agent Systems, Advanced Multi-Agent Systems are chosen as the ideal framework to implement Distributed Collaborative Prognostics, as they display all its properties.

After justifying the choice of Multi-Agent Systems, this chapter moves on to describe all the architectures and agents used in this thesis. In the following chapters, it will be shown that the architectures and agents presented here are sufficient to study Distributed Collaborative Prognostics in several scenarios. This includes simulation and industrial implementation.

From the high-level descriptions provided here, it becomes clear that each architecture has its advantages and disadvantages. For example, architectures that feature a higher degree of decentralisation (such as the Distributed and Hierarchical architectures) are more resilient to agent failure, but they are also more complex and costly to operate. Similarly, Heterarchical and Centralised architectures are simpler to implement and operate but can be sensitive to the failure of higher-order agents. Chapter 6 is dedicated to study these effects in more detail.

The nomenclature and components of the architecture's building blocks are consistent across this thesis. As a consequence of this, the agents used in Chapters 4, 5, 6, and 7 can all be categorised as one of the agent types described in this chapter. This means that Digital Twins, Mediator agents and all the other building blocks of the architecture can be programmed in different programming languages, and attain different complexity levels whilst still conforming to the definitions presented here.

As for the case of building blocks, architectures can also be implemented in different programming languages. The rest of this thesis includes several slightly different realisations of the architectures presented in this chapter. A good example of this is the mechanics of inter-agent communications: in a real industrial implementation, communication is done through the internet. In a multi-agent simulation communication is done by sharing variables in the internal memory of the simulation.

The next chapter presents an implementation of a Distributed architecture in a multi-agent simulation software. This multi-agent implementation is used to study the dynamics of the coupling between Distributed Collaborative Prognostics and a maintenance policy.

Chapter 4

Distributed Collaborative Prognostics with a maintenance policy

This chapter studies the coupling of Distributed Collaborative Prognostics with a conventional maintenance policy, thus addressing this thesis' second research question[1]. By providing a Multi-Agent System implementation of Distributed Collaborative Prognostics, this chapter also covers part of the first research question of this thesis, referring to the use of Multi-Agent Systems for prognostics. In here, a simple prognostics algorithm based on non-linear regression from a one-dimensional Health Indicator is used. Prognostics is dealt with in more depth in the subsequent Chapters 5 and 7, in which regression from multi-dimensional time series is implemented.

This chapter features a Multi-Agent System able to perform Distributed Collaborative Prognostics, implement maintenance decisions, and monitor maintenance costs. This means that this system can be used to test whether collaborative learning has the potential to reduce maintenance costs using a Multi-Agent System approach. The system is programmed in the Multi-Agent System simulation software NetLogo, in which its dynamics are put to test [150]. NetLogo is chosen because of its simplicity, and because it can be linked to MATLAB, a well-known mathematical computing software. MATLAB is used to compute the prognostics algorithm in the agents of the system.

This chapter contains results published by the author in Future Generation Computer Systems in a paper aimed to demonstrate the general idea of Distributed Collaborative Prognostics [38].

[1] See Section 1.7.

4.1 Description of the system

For this implementation a Distributed multi-agent architecture is chosen (Fig. 4.1). However, the results presented here also apply to a Heterarchical architecture as the collaborative learning algorithm used in this chapter is not affected by their architectural differences (this stops being true once agent failures are incorporated in Chapter 6). In this system, Virtual Assets and Digital Twins are grouped together in the same NetLogo agent, as synthetic data is used. In the remainder of this chapter, the word agent is used to designate Digital Twins only.

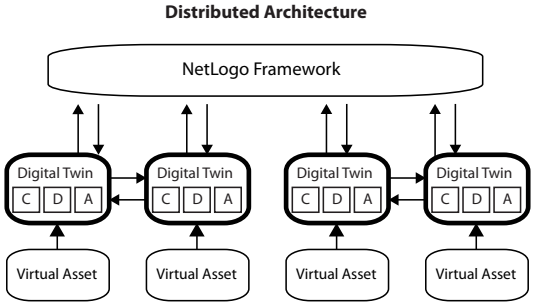


Fig. 4.1 Diagram of the Distributed architecture implemented in NetLogo. Note how human agents have been replaced by NetLogo, and how no physical machines are used.

In this implementation, information about the prognostic model is the only data shared among the Digital Twins. Thus, communication costs are deemed negligible compared to maintenance costs. Similarly, the computational demand on the Twins is assumed constant. The balance between communication, maintenance, and processing costs is studied in detail in Chapter 6.

Here, an exponentially decreasing Health Indicator is chosen as an approximate representation of the deterioration of the assets in the system. Such indicators are commonly used in prognostics to synthesise the information received from several sources of information (see, for example, [91, 151–154]). Health Indicators are one-dimensional scalars that represent the state of assets. For the Health Indicator chosen here, an asset i is assumed to fail if the Health Indicator HI_i is smaller or equal than 0: $\mathrm{HI}_i \leq 0$. The equation describing the evolution of this Health Indicator value with respect to the local time t_{li} of the asset i follows:

$$\mathrm{HI}_i(t_{li}) = a_i \left(1 - e^{-b_i(t_{fi}-t_{li})}\right) + \varepsilon_{0,\sigma}, \tag{4.1}$$

in this equation, t_{li} corresponds to the time since its last repair or installation. $\left(a_i, b_i, t_{fi}\right)$ are the parameters that define the evolution of the Health Indicator. b_i is a curvature parameter

(the smaller b_i is, the sharper the deterioration). a_i approximates the expected value of HI_i at $t_{li} = 0$. t_{fi} is the average (or expected) time of failure. $\varepsilon_{0,\sigma}$ is a random term with standard deviation σ and 0 mean, conforming to a Gaussian distribution. This random term is added to the equation to model the noise that is always present in asset sensors. In the experiments presented in this chapter, a_i will be normalised to 1. This normalisation gives significance to the noise parameter, as its standard deviation can be directly compared to the maximum value of the Health Indicator.

The objective of the Digital Twins is to determine the time of failure of the asset with minimal error. In this implementation of Distributed Collaborative Prognostics, it is assumed that they know the function followed by the Health Indicator, and their prognostics task is limited to figuring out its parameters (in this case (a_i, b_i, t_{fi})). This is a very generous assumption, as in reality the dynamics of the Health Indicator are often unknown and prognostics is done with less prior knowledge on the form of the prognostics function.

At any given time, each Digital Twin selects a list of N_i collaborating Twins. Collaborations that are represented in the Multi-Agent System by directed links. Each of these links is identified by a duplet of integers (i, j). In this duplet, i is used to identify the agent that receives information and j the agent that sends information (see Fig. 4.2).

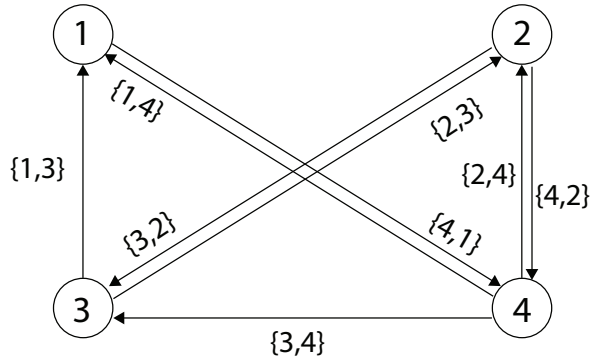


Fig. 4.2 Diagram of the communications among Digital Twins in this implementation. In this diagram, each Digital Twin is linked to Ni = 2 collaborating agents. The links representing these collaborations are represented visually and labelled using their duplet (i, j) .

It is assumed that the value of the Health Indicator is known at all times by the Digital Twins through a set of sensors implemented in each asset. The Digital Twins have to estimate the values of $\left(a_i, b_i, t_{fi}\right)$, that remain unknown. The triplet of estimated values is marked as $\left(\bar{a}_i^e, \bar{b}_i^e, \bar{t}_{fi}^e\right)$. This triplet is obtained through the prognosis algorithm implemented in the Twins, described in the following section.

4.2 A basic collaborative prognosis algorithm

To test Distributed Collaborative Prognostics in the basic form proposed in this chapter, a rudimentary collaborative prognosis algorithm is employed. This should not be confused with the final version of Distributed Collaborative Prognostics, presented in Chapter 5. In this algorithm, each agent in the implementation uses its Analytics Engine to estimate (a_i, b_i, t_{fi}) from the values of its Health Indicator, HI_i. To do so, the analytic engine of the Digital Twins executes the following operations:

1. The Digital Twins are provided with the latent time n that they must wait after asset installation before starting to perform prognostics. This hiatus of n time-steps is chosen in order to accumulate enough data to perform a non-linear fit of eq. (4.1) (see step 3).

2. The Twins' set of collaborating Digital Twins is determined by selecting the assets producing the N_i smallest distances d_{ij}. The indexes corresponding to these assets are saved in the vector $\vec{N}_i$. The distances d_{ij} are computed as follows[2]:

$$d_{ij} = \frac{1}{\sqrt{3}} \sqrt{\frac{(\bar{a}_i^e - \bar{a}_j^e)^2}{\bar{a}_{\max}} + \frac{(\bar{b}_i^e - \bar{b}_j^e)^2}{\bar{b}_{\max}} + \frac{(\bar{t}_{fi}^e - \bar{t}_{fj}^e)^2}{\bar{t}_{f\max}}}. \tag{4.2}$$

   In this equation, $\left(\bar{a}_{\max}, \bar{b}_{\max}, \bar{t}_{f\max}\right)$ are the maxima of the values of $\left(\bar{a}_i^e, \bar{b}_i^e, \bar{t}_{fi}^e\right)$, used to normalise the distance metric. During the first n time steps, a vector of collaborating assets formed entirely by random integer numbers is stored in the agent's Data Repository. The integers forming this vector are composed by the indexes j of the randomly chosen assets in the fleet.

3. Once the time n has been reached, the Digital Twins fit eq. (4.1) without the noise term to the Health Indicator data available to them (which corresponds to the data coming from their corresponding asset). In this experiment, this is done using MATLAB's `lsqnonlin`, a trust-region-reflective nonlinear least squares fitting algorithm (see, for example, [156]). This returns a triplet of floats $\left(a_i^e, b_i^e, t_{fi}^e\right)$.

4. Each Digital Twin receives the triplets $\left(a_i^e, b_i^e, t_{fi}^e\right)$ estimated by its nearest neighbours (obtained from the vector $\vec{N}_i$). Each Twin then re-evaluates its estimates using a weighted average from its own parameters and those of its neighbours. This is akin to what in the literature is known as sharing learning policies [25]. Each agent's estimates

[2] This metric has been obtained from [155].

are re-evaluated as follows:

$$\bar{a}_i^e = a_i^e w_{ii} + \sum_{\vec{N}_i} w_{ij}\bar{a}_j^e, \quad \bar{b}_i^e = b_i^e w_{ii} + \sum_{\vec{N}_i} w_{ij}\bar{b}_j^e, \quad \bar{t}_{fi}^e = t_{fi}^e w_{ii} + \sum_{\vec{N}_i} w_{ij}\bar{t}_{fj}^e. \tag{4.3}$$

The weights w_{ij} are set to depend exponentially with the inverse of the distance,

$$w_{ij} = \frac{e^{-d_{ij}\gamma_j}}{\sum_{\vec{N}_i} e^{-d_{ij}\gamma_j}}. \tag{4.4}$$

Weights are used to exert control on the influence of the data from collaborating Twins in the estimates of each Digital Twin. An exponential function is chosen because of its asymptotic behaviour at large γ_j, and its simple form at $\gamma_j = 0$. In this way, γ_j sets how strict the weighting is with respect to the calculated differences d_{ij}. $\gamma_j = 0$ would correspond to all Twins contributing equally. As γ_j increases, the data from the asset assigned to each particular Digital Twin also gains increasing importance (as opposed to the data coming from other Twins).

5. The Twins send the triplets $\left(\bar{a}_i^e, \bar{b}_i^e, \bar{t}_{fi}^e\right)$ to all the other Digital Twins. Every Digital Twin then calculates the distance d_{ij} to the rest of the assets using eq. (4.2).

6. The Twins are connected to their N_i nearest neighbours holding the smallest values of d_{ij} (the identities of whose have been stored in the vector $\vec{N}_i$).

7. Digital Twins recommend a maintenance action to the asset managers according to the policy described in Section 4.3.

8. Steps 2-6 are repeated at each time step.

Trough this implementation, NetLogo conveys the system configuration desired by the asset managers to the Digital Twins, thus setting fleet-wide parameters such as γ_i, N_i, n, etc.

4.3 Maintenance policy

The maintenance policy employed in this implementation is a simple predictive maintenance policy in which agents recommend preventive repairs when the asset local time t_{li} surpasses the estimated time of failure multiplied by a positive scalar η_i. The condition of preventive maintenance is:

$$\frac{t_{li}}{\bar{t}_{fi}^e} \geq \eta_i, \qquad 0 \leq \eta_i \leq 1. \tag{4.5}$$

Apart from preventive repairs, assets are repaired correctively immediately upon failure ($\mathrm{HI}_i \leq 0$). Assets are assumed to be repaired *as new* in both cases, thus the local time (or age) t_{li} is set to 0, which resets the value of HI_i.

The accumulated maintenance costs per unit time can be minimised in real time if $\bar{t}^e_{fi}$ is assumed to be an estimate of the real value of t_{fi}. Under convergence of the parameters $\left(\bar{a}^e_i, \bar{b}^e_i, \bar{t}^e_{fi}\right)$, the problem's conditions resemble a time-based replacement policy problem, in which optimisation for one maintenance cycle corresponds to the long-term solution [157]. This is, however, not the case in our system, as the estimated parameters are continuously updated with increasingly precise predictions as the asset comes closer to failure and gathers more data about itself. Thus, maintenance recommendations should also be updated, and the uncertainty associated with these recommendations should be appropriately leveraged.

The derivation that follows illustrates a time-based replacement policy implementable in a Distributed Collaborative Prognostics system similar to the one described above. In it, it is assumed that the probability distribution of the Health Indicator at each time-step is given by a normal distribution, resulting only from propagating the term $\varepsilon_{0,\sigma}$, representing sensor noise[3]. For each asset with a triplet $\left(\bar{a}^e_i, \bar{b}^e_i, \bar{t}^e_{fi}\right)$, the total expected cost per unit of time is:

$$C(\eta_i \bar{t}^e_{fi}) = \frac{C_p + C_c H(\eta_i \bar{t}^e_{fi})}{\eta_i \bar{t}^e_{fi}}. \tag{4.6}$$

In here, $H(\eta_i \bar{t}^e_{fi})$ is the expected number of failures between asset deployment and maintenance $(0, \eta_i \bar{t}^e_{fi})$. C_p is the cost of preventive repair and C_c is the cost of corrective repair. Normally, the cost of preventive repair C_p is much lower than the cost of corrective repair C_c (otherwise, preventive maintenance would be fruitless). The minimum cost corresponds to the value of η_i in which the first derivative of eq. (4.6) vanishes. Derivation leads to the following expression:

$$\eta_i \bar{t}^e_{fi} h(\eta_i \bar{t}^e_{fi}) - H(\eta_i \bar{t}^e_{fi}) = \frac{C_p}{C_c}. \tag{4.7}$$

h is the derivative of H. Assuming that the time step is arbitrarily small, the expected number of failures per cycle equals the probability of failures during one cycle. First, the probability of failure at time t is obtained:

$$P(t) = \int_{-\infty}^{0} N(\bar{\mathrm{HI}}_i(t), \sigma, x)dx = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{-\bar{\mathrm{HI}}_i(t)}{\sigma\sqrt{2}}\right)\right). \tag{4.8}$$

[3]Note: this means that uncertainty from the non-linear fit is not propagated.

Here $\bar{\text{HI}}_i$ is:

$$\bar{\text{HI}}_i(t_{li}) = \bar{a}_i^e \left(1 - e^{-\bar{b}_i^e(\bar{t}_{fi}^e - t_{li})}\right), \tag{4.9}$$

and $N(\bar{\text{HI}}_i(t), \sigma, x)$ is the Gaussian function with mean $\bar{\text{HI}}_i(t)$ and variance σ. In one cycle, the expected number of failures is calculated by integrating $P(t)$ over the time cycle and normalising by the time period $\eta_i \bar{t}_{fi}^e$:

$$H(\eta_i \bar{t}_{fi}^e) = \frac{1}{\eta_i \bar{t}_{fi}^e} \int_0^{\eta_i \bar{t}_{fi}^e} P(t)dt = \frac{1}{2\eta_i \bar{t}_{fi}^e} \left(\eta_i \bar{t}_{fi}^e + \int_0^{\eta_i \bar{t}_{fi}^e} \text{erf}\left(\frac{-\bar{\text{HI}}_i(t)}{\sigma\sqrt{2}}\right) dt\right) = \frac{1}{2\eta_i \bar{t}_{fi}^e} \left(\eta_i \bar{t}_{fi}^e + \int_0^{\eta_i \bar{t}_{fi}^e} \text{erf}\left(\frac{-\bar{a}_i^e \left(1 - e^{-\bar{b}_i^e(\bar{t}_{fi}^e - t)}\right)}{\sigma\sqrt{2}}\right) dt\right). \tag{4.10}$$

$h(\eta_i \bar{t}_{fi}^e)$ can be obtained using the fundamental theorem of calculus:

$$h(\eta_i \bar{t}_{fi}^e) = P(\eta_i \bar{t}_{fi}) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{-\text{HI}_i(\eta_i \bar{t}_{fi}^e)}{\sigma\sqrt{2}}\right)\right). \tag{4.11}$$

Eq. (4.7) then follows:

$$\eta_i \bar{t}_{fi}^e h(\eta_i \bar{t}_{fi}^e) - \frac{1}{2\eta_i \bar{t}_{fi}^e} \left(\eta_i \bar{t}_{fi}^e + \int_0^{\eta_i \bar{t}_{fi}^e} \text{erf}\left(\frac{-\bar{a}_i \left(1 - e^{-\bar{b}_i(\bar{t}_{fi}^e - t)}\right)}{\sigma\sqrt{2}}\right) dt\right) = \frac{C_p}{C_c}, \tag{4.12}$$

which can be solved for η_i numerically.

4.4 Simulation parameters

The system presented in this chapter is simulated using NetLogo. In this initial set of experiments, NetLogo was connected to MATLAB using MatNet, an open source extension of NetLogo [158]. MATLAB is a numerical computing environment, used in this section to calculate the non-linear fit of the Health Indicator, and to solve eq. (4.12). In the NetLogo simulations studied here the following parametric choices are made:

- n (the number of time steps in which there is no prognostics) is set to 10.
- γ_i is constant for all Digital Twins, $\gamma_i = \gamma$.
- Only three families of assets are simulated, with the following triplets $F : (a, b, t_f) \rightarrow \{(0.25, 0.025, 25), (0.5, 0.05, 50), (0.75, 0.075, 75)\}$.

- In this first run of experiments, a small fleet of 27 assets (9 per each family) is simulated (much larger fleets are simulated in Chapter 6).
- The cost of corrective repair is 100 times larger than the cost of preventive repair, that is set to 1.
- Numerically solving eq. (4.12) in real time slowed down multi-agent simulations. Instead, it is chosen to give the asset fleet a fixed value of $\eta_i = \eta$, and compute several simulations with different values of η. This allows for examining the behaviour of the system under suboptimal maintenance policies, and checking the validity of eq. (4.12) simultaneously.
- Experiments are run until convergence, defined by the absolute difference between the real time of failure and the estimate time to failure averaged over all the Digital Twins in the system. Concretely, convergence is achieved if $\left\langle |\bar{t}^e_{fi} - t_{fi}| \right\rangle < \kappa$. Where κ is chosen to be the smallest time step of the system, $\kappa = 1$.

In this implementation, the variables of interest will be the total maintenance cost, and the cost per time K. Results show their dependency on the number of collaborating assets N_i, and the Health Indicator noise σ. Additionally, their dependency on the replacement policy η, and the weighting of the data coming from different agents γ is also studied. The maintenance cost per time is:

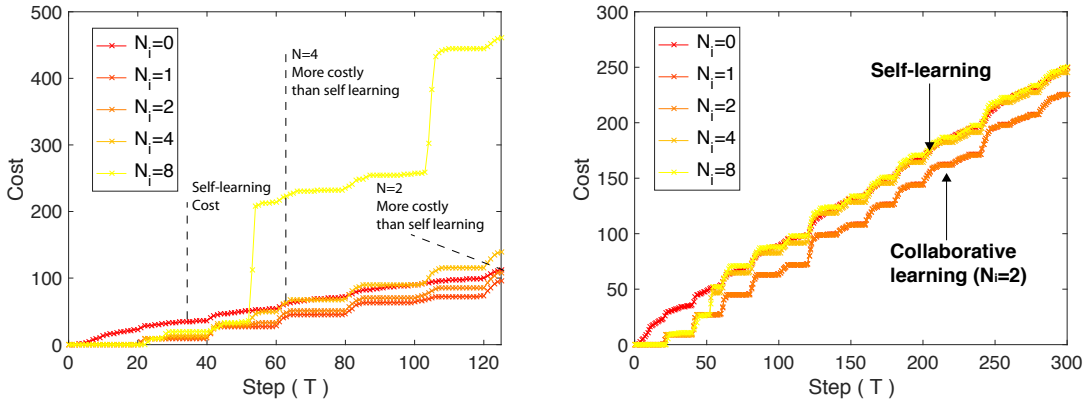
$$K = \frac{\sum_{t=0}^{T} C_t}{T}. \tag{4.13}$$

Where C_t is the total cost of maintenance actions occurring at a time t. $\sum_t C_t$ is the total cost of the simulation.

4.5 Results

Firstly, it is observed that K has a non-trivial dependency on the studied variables [38]. This means that the total cost per unit time has several local minima, and that when designing Distributed Collaborative Prognostics, care has to be taken to perform an appropriate choice of system parameters (such as, for example, the number of collaborating Digital Twins N_i).

Secondly, due to the absence of other cost contributions, K is used as a proxy for model accuracy. For a constant η, less accurate prognostics will produce larger costs. This allows for a clear interpretation of variations in K with respect to the other variables of the system. An example of this is the case of $\sigma = 0.01$, $\gamma = 0.1$ and $\eta = 0.8$ (see Fig. 4.3). In this case, increasing the number of Digital Twins that each Twin collaborates with decreases



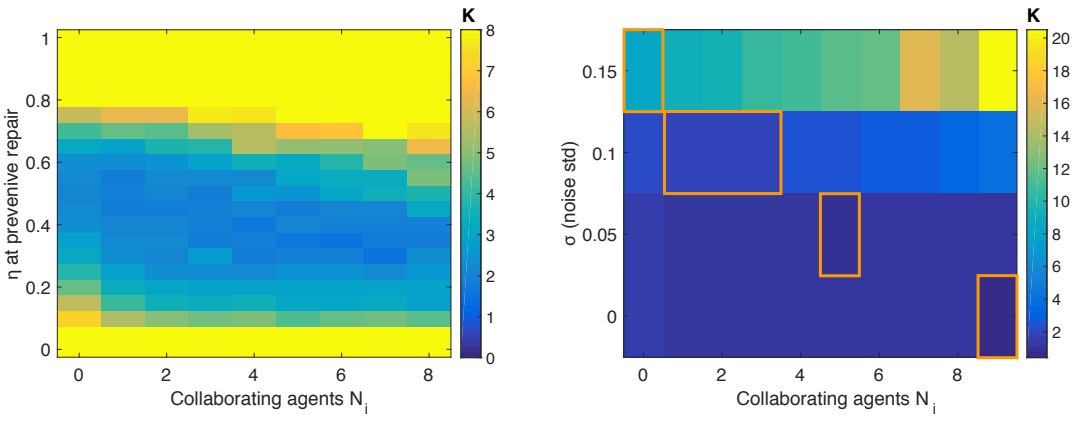
(a) Fix γ. (b) Dynamic γ.

Fig. 4.3 Cost of the system before and after convergence with collaborative learning (a), and restricting to self learning upon convergence, as suggested in Section 4.5.1 (b). Cost averaged over 50 experiments for $\sigma = 0.01$, $\gamma = 0.1$ and $\eta = 0.8$.

the associated costs until $N_i = 3$. This tendency reverses for $N_i \geq 4$, because Digital Twins representing assets that are too different from each other start collaborating, leading to less accurate prognostics and a costlier solution (see Fig. 4.3).

The dependency of cost on η and N_i is shown in Fig. 4.4a. This figure shows that given a relatively small noise term of $\sigma = 0.1$, the optimal number of collaborating Digital Twins is $N_i = 7$. As expected, too cautious (small η) and too risky (large η) maintenance policies are not cost effective. This is because at large η, preventive repairs are taken too late, and unpredicted failures occur, increasing corrective maintenance costs. At lower η the opposite effect happens: too frequent preventive maintenance actions and the impossibility of gathering enough close-to-failure data end up causing large costs.

Another interesting cost-dependency is on the number of collaborating Twins and the noise in the system, shown in Fig. 4.4b for $\eta = 0.5$ and $\gamma = 0.1$. This figure shows the expected monotonic increase of costs with noise σ. It also shows that the dependency of cost on the number of collaborating Digital Twins N_i happens not to be monotonic: only if there is no noise, higher N_i consistently means less cost. Once noise is introduced into the system, learning from more Digital Twins is not always beneficial and local minima appear. This is consistent with the idea of collaborative learning: it is very important to be sure to be learning from assets similar to oneself (recall the thought experiment of Chapter 1).



(a) Global minimum at $N = 7$, $\eta = 0.3$.

(b) Local minima marked in orange squares.

Fig. 4.4 Cost per unit time, K for different values of N_i, η and σ averaged over 20 trajectories. Fix values (a) $\sigma = 0.1$, (b) $\eta = 0.5$, both with $\gamma = 0.1$. Note how in (b) for $\sigma > 0.1$, increasing the number of collaborating assets, N_i, is detrimental for the system cost.

4.5.1 Behaviour after convergence

A second round of experiments was performed to study the behaviour of the system after convergence. The aim of these experiments was to test whether the cost benefits of collaborative learning are long-lasting or vanish with time.

Interestingly, when the system is run for a sufficiently long time, for small values of γ, collaborative learning is asymptotically costlier than self-learning ($N_i = 0$). This is shown in Fig. 4.3 for the cases of $N_i = 2$ and $N_i = 4$. This again, can be understood from the thought experiment done at the beginning of this thesis (see Section 1.4). If there is enough time, and data, it does not pay off to learn from different machines and risking obtaining data that is not pertinent to the asset itself.

This caveat is avoidable by using dynamic values of γ: the more information an agent has from the asset that it is assigned to, the less data it should use from other assets. This is tested experimentally by adding the following rule to the collaborative learning algorithm: if $|\bar{t}^e_{fi}(t_{li}) - \bar{t}^e_{fi}(t_{li}+1)| < \varepsilon$, during five consecutive time steps, then $\gamma_i \to \infty$ (equivalent to $N_i = 0$). Fig. 4.3 shows that this modified collaborative learning algorithm is always less costly than self-learning (here, $\varepsilon = 1$ is used).

4.5.2 Optimisation of the maintenance policy

Due to the dynamic nature of the failure predictions returned by the agents in the system, it is unclear whether the preventive maintenance policy can be optimised as described in Section 4.3. The following test is performed: the optimised weighted η value is calculated using eq.

(4.12), and plotted against the optimal η value obtained numerically by running experiments for a wide span of η values. Note that here η is shared for the whole fleet, thus the optimal value of η is approximated by the weighted average of η for each sub-fleet in the experiment. Results show that theory and simulation are similar, especially for low values of σ (see Fig. 4.5). This result can be interpreted as follows: even if the prognosis is not perfectly accurate, a maintenance policy as given in eq. (4.12) is still close to to the least-costly maintenance policy measured for the fleet.

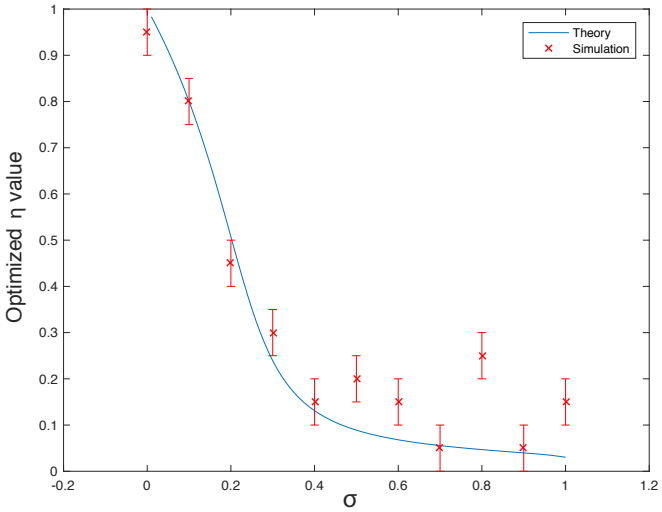


Fig. 4.5 Optimal (minimum cost) η value, compared to the optimal value of η predicted by eq. (4.12). Note how higher values of σ imply higher variance in the system as the estimated values of $\left(\bar{a}_i^e, \bar{b}_i^e, \bar{t}_{fi}^e\right)$ fluctuate due to noise.

4.6 Conclusion

The following findings can be derived from the results presented in this chapter:

- **Effects of noise**: the system reacts with noise in the Health Indicator as expected: more noise generally means more prediction error, and thus higher costs. At relatively low values of σ ($\sigma \lesssim 0.1$), collaborative learning outperforms self-learning. However, at large enough values of σ, it becomes more cost effective for the Digital Twins to use data only from their pertinent asset. The reason for this is that for large values of σ, Twins start learning from other Digital Twins that do not belong to the same asset kind. This induces large prognostics errors that ultimately overcome the reduction of

uncertainty given by sharing the estimates from different agents. It is thus crucial to develop collaborative algorithms resilient to sensor noise.

- **On data weighting**: in the presented system, γ has the role of determining the weighting of data coming from collaborating Digital Twins. It is observed experimentally that γ can be used to restrict negative effects of sensor noise on the system's cost. Thus, higher values of γ are advisable in systems where the noise σ is also higher.

- **On the maintenance policy**: a real-time optimisation based on the age based replacement policy problem can be used as long as the system is believed to have converged to accurate enough prognostics models. Although such convergence is hard to estimate in practice, the results from this study suggest that even when the prognostics models are not very accurate, such a theoretical approach at least approximates the optimal maintenance policy. This is an important result as it provides a strategy to implement real-time maintenance policies in Distributed Collaborative Prognostics, thus addressing this thesis' second research question.

- **On the number of collaborating Digital Twins**: as long as sensor noise[4] is kept sufficiently low, an increase in the number of collaborating Twins implies a reduction of total costs. As sensor noise increases, the optimal number of collaborating Digital Twins reduces, until to a point in which the minimum cost corresponds to the case of self-learning. Fleet-wide learning (learning from all agents in the fleet regardless of their differences) is significantly costlier than any of the alternatives.

The small size of the fleet, the simplicity of the cost function and the assumption that agents know the fundamental equation of the deterioration process are known weaknesses of this study. These weaknesses are addressed in Chapter 6, in which a sophisticated cost analysis accounting for all the other costs of the system is presented. Besides this, a much larger fleet of assets is introduced and implemented for all the architectures presented in Chapter 3.

This chapter presents an initial study of Distributed Collaborative Prognostics coupled to a maintenance policy, and sets the foundations for further study. It is concluded that, under stringent assumptions, collaborative learning can be used to significantly reduce maintenance costs. In the next chapter, an implementation deployable in realistic industrial scenarios is presented, which is used to test the properties of Distributed Collaborative Prognostics presented in Chapter 3.

[4]Sensor noise and Health Indicator noise are assumed to be linearly related.

Chapter 5

An implementation of real-time Distributed Collaborative Prognostics

This chapter presents an implementation of Distributed Collaborative Prognostics deployable in a real industrial scenario. This implementation uses synthetic data from the C-MAPSS engine degradation data set and the PHM08 prognostics data challenge, consisting of hundreds of high-dimensional trajectories to failure, designed to emulate the sensors in turbofan engines. Thus, the results shown here are physically more realistic than in the preceding chapter, in which a one-dimensional Health Indicator was used. In this chapter, no specific model is assumed for the deterioration of the machines. Instead, Recurrent Neural Networks are used to obtain an estimate of the time to failure probability density function.

All the code presented here is programmed in `python` and `bash`, two programming languages with a widespread presence in industry. Effort has been made to adhere to agent communication standards, such as the Knowledge Query and Manipulation Language (KQML) [105]. A big part of the work presented in this chapter has been published in the form of a journal paper to Engineering Applications of Artificial Intelligence [23].

Apart from presenting a deployable Distributed Collaborative Prognostics solution, this implementation also presents the first prognostics tool shown to fulfil all properties of an Advanced Multi-Agent System: Distribution, Flexibility, Adaptability, Scalability, Leanness, and Resilience. Advanced Multi-Agent Systems can be seen as an archetypical version of Multi-Agent Systems, in which the system leverages the capabilities of modern-day agents [22, 23]. As discussed in Chapter 3, Advanced Multi-Agent Systems provide an optimal framework for Distributed Collaborative Prognostics as they fulfil all of its properties.

Hitherto, this thesis' research questions have been addressed within NetLogo simulations. In this chapter, research questions 1 and 3, referring to the use of Multi-Agent Systems,

and to the study of the appropriate conditions for Distributed Collaborative Prognostics are addressed from a system design and experimental perspective[1].

5.1 Advanced Multi-Agent Systems

As discussed in Chapter 3, Advanced Multi-Agent Systems constitute an existing framework that fulfils all the requirements of Distributed Collaborative Prognostics. The definition of an Advanced Multi-Agent System is repeated here for completeness [22, 23].

- Distribution: the Multi-Agent System architecture must have a decentralised structure. Optimisation, decision-making and control are performed by the agents in the system.
- Flexibility: the system must be able to adapt in real-time to changes in the agents and the environment.
- Adaptability / System learning: thanks to the real-time learning capability of the agents, the output of the system is constantly updated and improved in reaction to human demands and changes in the environment.
- Scalability: the system must be easily scalable. New devices must be able to join the system without important changes in the architecture or the agents composing the Multi-Agent System.
- Leanness: an Advanced Multi-Agent System should always strive towards minimizing the number of different categories of agents that it features. Differences between the agents should be kept small to obtain a swarm-like behaviour.
- Resilience: the system should be designed so that, as much as possible, it is resilient to agent faults.

5.2 Architecture description

The architecture used in this implementation was chosen taking into account restrictions from this thesis' industrial partner, described in detail in Chapter 7 (as this implementation would be later used for the case study presented in said chapter). It was determined that the most convenient architecture was a modified Hierarchical architecture with three layers: Virtual Assets, Digital Twins, and a Social Platform. Note that in this version of the Hierarchical

[1] See Section 1.7.

architecture Digital Twins are used instead of Mediator Agents because each Twin only deals with one physical asset (see Fig. 5.1). The difference between this and a purely Distributed architecture is that Twins communicate with each other through the Social Platform instead of directly. Virtual Assets, Digital Twins, and the Social Platform are implemented as described in Section 3.2.2.

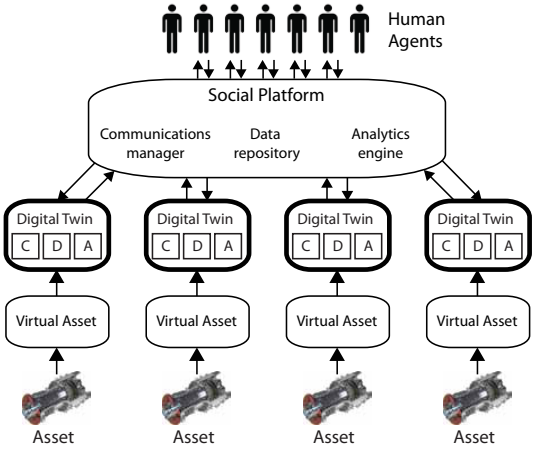


Fig. 5.1 Sketch of the architecture used in this chapter. Black arrows represent communications.

5.3 Real-time collaborative prognostics

Similar to what is done in Chapter 6, collaboration is enabled by sharing data between similar machines. The difference lies in that, in this case, a fully-mature prognostics algorithm based on real-time training and predicting using a Recurrent Neural Network is used, together with a realistic prognostics data set (see Chapter 2 for the mathematical foundations). This means that in this case the agents do not know the underlying equation of the deterioration process, and that a Health Indicator is not used.

Unlike in Chapter 4, the prognostics algorithm used here converts readings from multi-variable time-series sensor data into time to failure estimates. In order to collaborate, Digital Twins share multi-variable data with each other through the Social Platform. This data is incorporated directly into the training of the neural network as training examples $(x^n_{0:t}, y^n_t)$, also known as training trajectories (see Fig. 5.2 and Section 2.2 for details). These training examples are weighted in the loss function of the prognostics algorithm as follows:

$$L = -\sum_{n=1}^{N_i}\sum_{t=0}^{T_n} W_n \log\left[\Pr\left(Y^n_t = y^n_t | x^n_{0:t}\right)\right]. \tag{5.1}$$

Here W_n is defined as w_{ij} in eq. (4.4) in Chapter 4: $w_{ij} = \frac{e^{-d_{ij}\gamma_j}}{\sum_{N_i} e^{-d_{ij}\gamma_j}}$. where d_{ij} is the scalar distance calculated between assets i and j. For this implementation γ_j is chosen to be 0 and therefore $W_n = 1/N_i$, where N_i are the number of trajectories available to the Digital Twin. These trajectories are determined by means of a real-time clustering algorithm that determines the vector $\vec{N}_i$ of collaborating assets. This means that all trajectories, as long as they belong to the same cluster of assets, are weighted equally.

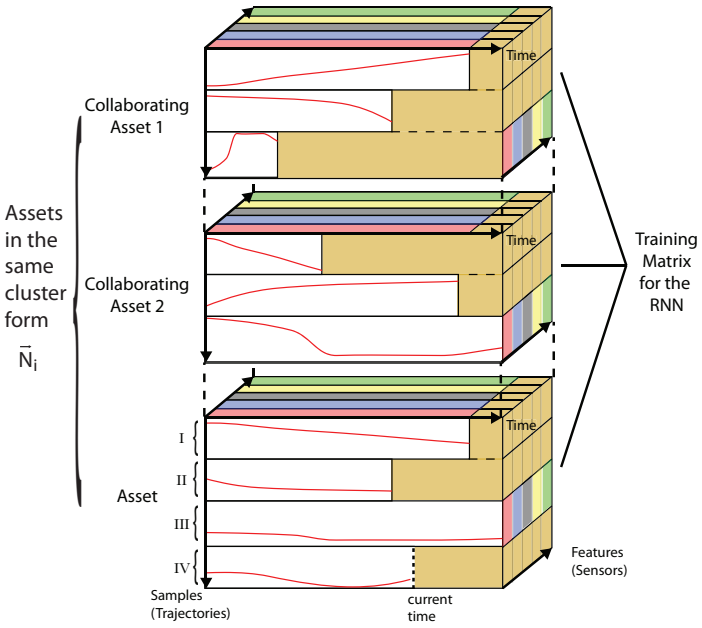


Fig. 5.2 Sketch showing the training data matrix fed to the asset's Recurrent Neural Network. Note how several matrices similar to the one shown in Fig. 2.2 are appended.

As mentioned in the introduction of this chapter, this Distributed Collaborative Prognostics implementation is programmed almost entirely in Python. For the machine learning implementation, `Tensorflow` [159], `Keras` [160] and the `wtte-rnn` library are used. `Keras` and `Tensorflow` are open-source deep learning libraries, and are chosen because of their spread and maturity in the field. `wtte-rnn` is a library programmed by Egil Martinsson that enables prognostics based on a Weibull loss. Python was chosen over multi-agent software frameworks such as NetLogo to ensure the system's deployability in a real industrial scenario, and because of its access to well-tested machine learning libraries. Fig. 5.3 shows a UML[2] diagram of the Distributed Collaborative Prognostics algorithm. Fig. 5.4 shows a non-standardised diagram that includes the data flow of the system.

Communication between agents is performed via the `websocket` library [162], a python library that allows for sending and receiving messages using the `websocket` protocol, a

[2]A standardised way to visualise the design of software systems [161].

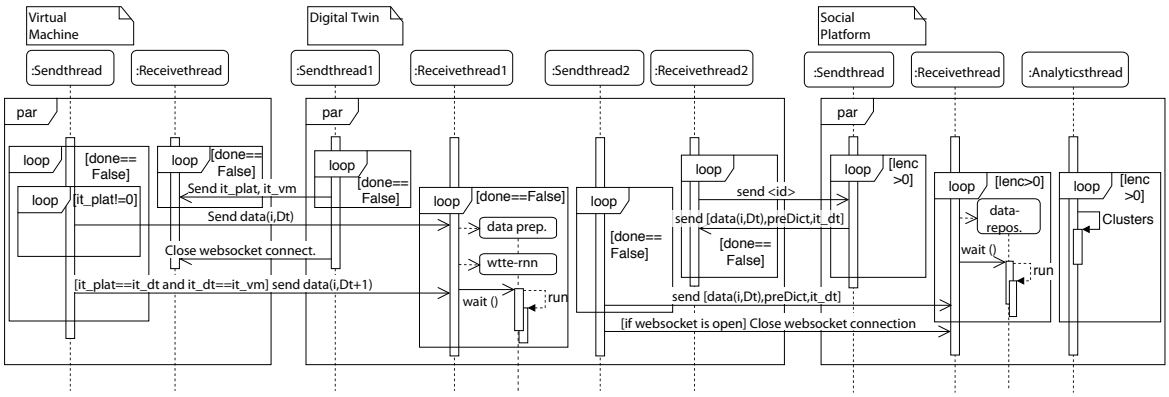


Fig. 5.3 UML diagram of the Multi-Agent System, including the major tasks performed by each block of the architecture (see Fig. 5.1). Arrows are used to indicate communication using websockets. lenc indicates the amount of machines connected to the Social Platform.

widely used protocol that allows for two-way interactive communications between a client and a server. Asynchronous processes are enabled by the `asyncio` library [163]. Parallel processes within the agents are computed using `threading` [164]. This Distributed Collaborative Prognostics implementation is devised so that it can be run in a physically distributed way, in a cluster, or in a single terminal through a bespoke `bash` script (see Appendix B.2.1). In any of these cases, the memory spaces of the agents are always strictly separated.

Typically, communication in Multi-Agent Systems is done using an Agent Communication Language [165]. In this implementation `pykqml` [104] is used, a python package that converts messages to the Knowledge Query and Manipulation Language (KQML) [105]. To do so, the agents can be programmed to transform python dictionaries into a KQMLList before sending them through the `websocket` protocol. This allows for a straightforward implementation of agent communication standards.

One of the biggest challenges identified during the development phase of this implementation was a bug caused by the loss of coupling between each agent's internal clocks (as different agents were subjected to different computational loads). This was solved by separating critical tasks of the agents into parallel threads. For example, within the communication manager the send and receive loops were separated and assigned different ports (see Fig. 5.4). This means that each agent pair is assigned two ports, with the exception of the Social Platform that despite being connected to a multiplicity of Digital Twins, only requires two ports to manage the entire fleet. This is due to the Social Platform acting as a server, with the Digital Twins as its clients.

The synchronization challenges mentioned above take place because the experiments go through years of physical data in a few hours by means of using the Virtual Assets to source the data. In a real-world implementation, the mean time between asset failures would be

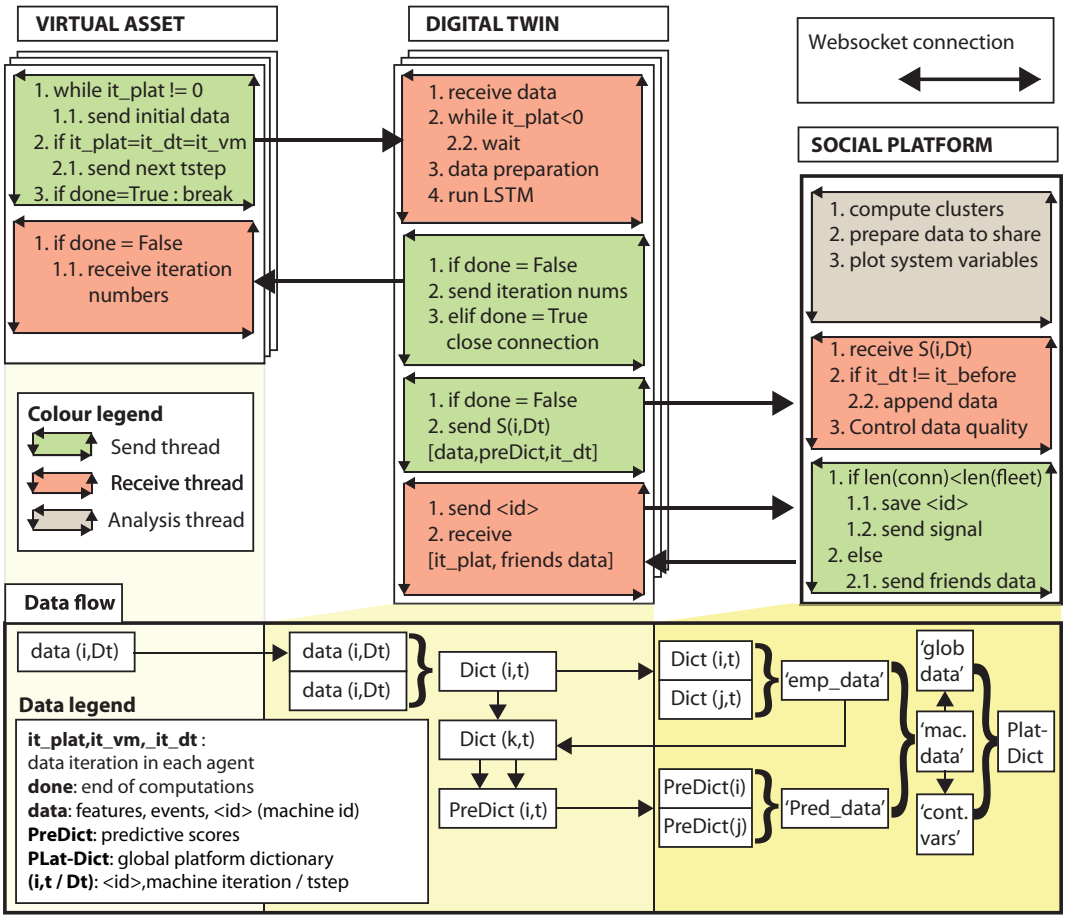


Fig. 5.4 Sketch of the Multi-Agent System and its data structures, including the major tasks performed by each block of the architecture (see Fig. 5.1).

much larger than the time required to train and execute the prognostics algorithm. Instead, in the experiments presented here, data generation is much faster than training. This increases demand on deep-learning algorithms and causes synchronisation problems.

In order to overcome this challenge, the analytics engine of the Digital Twin has been incorporated as an asynchronous process within the communication manager of the Digital Twin. This forces the Digital Twin to stop processing data until the training of the prognostics algorithm is completed, thus imitating the way in which a real-time implementation would operate.

Distributed Collaborative Prognostics is based on the exchange of data between agents. Thus, it is important to describe the data structures used in any of its implementations. In this case, inset python dictionaries are used so that the amount of data available to the system's agents increases as it travels upwards the hierarchy of the Multi-Agent System. From bottom to top: the Virtual Asset's most important data structure is the multi-variable sensor data

recorded at time interval Δt for the asset i. This data point is named "data$(i,\Delta t)$". At each Δt, data$(i,\Delta t)$ is sent to the corresponding Digital Twin together with an asset identifier, and the Digital Twin's data manager concatenates it into a `python` dictionary: Dict(i,t). This dictionary stores all the data sent by the Virtual Asset between times 0 and t, where $t = \sum \Delta t$ is the lifetime of the Digital Twin. Through its prognostics algorithm (run by the analytics engine), the Digital Twin produces a separate python dictionary, PreDict(i,t), that stores prognostics data (time to event prediction, probability density function, and accuracy data). PreDict and data$(i,\Delta t)$ are iteratively sent to the Social Platform, that stores them in its data repository together with the data coming from the other Digital Twins in a dictionary called 'machine data'. The Social Platform stores two other dictionaries: a global data dictionary called 'glob data', and the control dictionary 'control variables'. 'glob data' stores variables pertinent to the whole fleet, for example the output of the clustering algorithm (stored in a matrix known as the Friendship Matrix), and the global accuracy of the prognostics algorithm. 'control variables' is the smallest dictionary of the three, and contains human inputs and information about which agents are connected to the Social Platform.

5.3.1 Clustering

At each time-step the Social Platform executes a clustering algorithm to determine groups of similar assets. Until now, algorithms with a fixed number of collaborating assets N_i have been introduced (for example the collaborative learning algorithm introduced in Chapter 4). In a real fleet of industrial assets, the number of collaborating assets *and* the number of clusters is likely to vary depending on changes on the assets operational conditions, failure modes, etc. Thus, a clustering algorithm that automatically finds the optimal number of clusters k fits better the purpose of a flexible Multi-Agent System.

Selecting a clustering algorithm for a fleet of industrial assets is not always an easy task. For example, in the experiments presented in this section, the fleet of assets is relatively small compared to the dimensionality of the feature space used for clustering (sensor data). This is known as the curse of dimensionality [166]. After several experiments, it was found that DBSCAN, k-Means clustering with a variable number of clusters k and Hierarchical clustering were the three existing clustering algorithms better suited for the task. Ultimately, DBSCAN was chosen because (1) its superior accuracy in predicting the number of clusters, (2) its consistency across experiments, and (3) scalable extensions able to handle large populations were available [167].

For the case of DBSCAN, one must choose a metric to calculate distances between the assets in the population. In this case, a standard euclidean distance is found to give good results. Notwithstanding, distances such as the fractional distance metric or the Manhattan

distances should also be considered for cases in which the number of features representing every individual is of the order of the number of assets, as they are known to behave better under the curse of dimensionality [168].

The clustering algorithm implemented in the Social Platform follows:

Algorithm 1 Clustering algorithm

1: Obtain uncensored trajectories from each asset in the fleet;
2: **for** last N trajectories **do**
3: Take the temporal mean of each feature;
4: **end for**
5: Append the temporal means in a matrix featuring all assets in the fleet;
6: Normalise the matrices features using sklearn's MinMax scaler;
7: Compute euclidean distances between each element in the matrix;
8: Retrieve the maximum distance D;
9: Obtain the clusters using sklearn's DBSCAN with $\varepsilon = 5D/N$, in which N is the number of assets in the fleet, min_samples $= 1$;

5.4 Design of the experiments

This section presents a set of experiments designed to test whether the proposed system fulfils each of the properties introduced in the definition of Advanced Multi-Agent Systems: Leanness, Distribution, Scalability, Adaptability, Flexibility and Resilience. Together with real-time prognostics and communication between agents, the experiments presented here show that this implementation satisfies the properties of Distributed Collaborative Prognostics described in Chapter 3.

To test this, two standard prognostic data sets are chosen: the C-MAPSS Turbofan Engine Degradation Simulation Data Set [169], and the PHM08 prognostics data challenge data set [170]. These data sets are chosen because of two reasons: first, their popularity and longevity in the prognostics research community allow for readily interpretation of the results. Second: both data sets feature multi-variable time-series such as the ones present in most modern prognostics scenarios, and thus can be adapted to represent the conditions of real-time distributed prognostics.

The tool used to generate both data sets, C-MAPSS, is a MATLAB-based software able to reproduce the behaviour of modern-day turbofan engines [171]. Among other functionalities, C-MAPSS allows for replicating the effect of environmental and operational parameters in the sensors' readings preceding machine failures. For example, C-MAPSS allows to vary the Mach number (constrained to sub-sonic regimes), the altitude (between 0 and 40000 ft),

and the temperature at sea level. C-MAPSS replicates the behaviour of engines composed by several interrelated sub-systems, including limiters, control systems, and regulators. The limiters included in C-MAPSS resemble the control mechanisms often present in industrial machinery, designed to prevent machines from exceeding pre-set tolerances. In the turbines included in this data set there are limiters for the core speed, the high pressure turbine exit temperature, the engine-pressure ratio, and the static temperature at the High-Pressure Compressor (HPC)[3].

C-MAPSS can be used to simulate the failure of any of the machine's rotating components. However, the data set used in this chapter only features degradation in the HPC and fan modules. The time-series sensor values included in the data set represent fan and core speeds, engine pressure ratios, sub-system temperatures, bleed enthalpy, etc.

Distributed collaborative prognostics is especially suited for machines that experience recursive soft failures (such as recursive triggering of limiters that render the machine inoperative). Instead, the C-MAPSS data set employed here includes only one trajectory to failure for each machine in the data set. To overcome this issue, multiple trajectories to failure are treated as if they correspond to the same asset by appending them one after the other in the same time-series. This can be done as long as the C-MAPSS parameters used to generate these trajectories are known.

In order to prepare this chapter's experiments, it is crucial to have information about the operational condition and failure type of each trajectory, as this information can be leveraged in simulating machines that dynamically change their operational setting or failure type. Unfortunately, the data set employed here does not provide detail of all these parameters, as it was originally prepared to be used in a competition. Thus, some information is inferred from the data set in the data preparation step.

Next section analyses the data set and determines the properties of each trajectory to failure, so that they can be used to prepare the experiments.

5.4.1 Data preparation in the C-MAPSS data set

The data set used in this section features two kinds of failures: one related to fan degradation, and another one related to High Pressure Compressor degradation. This data set is divided into four subsets: FD001, FD002, FD003, and FD004. For FD003 and FD004, there is no explicit information regarding which assets experience each kind of failure. However, for machines operating in sea-level conditions (corresponding to the FD003 data set) their failure type can be inferred from the sensor values.

[3] An in-length description of C-MAPSS is included in [170].

To figure out which of the two failures correspond to each trajectory, it is noted that the values of sensor 7 show a monotonic positive trend in fan degradation failures, and a negative trend for the case of HPC degradation. A rolling average[4] combined with a majority rule in its first derivative is used to automatically discern between the two. A positive majority corresponds to fan degradation, and a negative majority to HPC failure. Manual classification compared with automatic classification returns a near-perfect accuracy (Fig. 5.5 shows the clear difference between the two failures).

Once the failure types have been classified, it is important to do the same for the operational settings of the machines in the dataset. An initial look into the data indicates that machines in the C-MAPSS data set operate in six different operational settings. However, upon further investigation, one realises that in reality machines can be classified in two types: those staying in a single environment (sea level) and those flying across varied environments (see Fig. 5.6).

Thanks to this realisation, enough is known about the characteristics of each trajectory to failure to design a set of experiments aimed to test the properties of Distributed Collaborative Prognostics. These experiments are described in the following section.

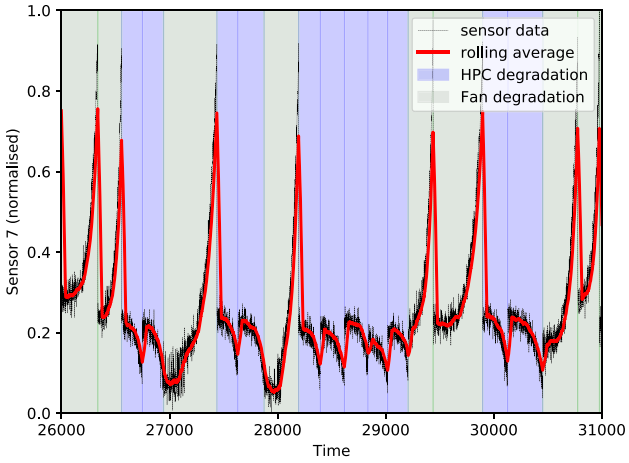
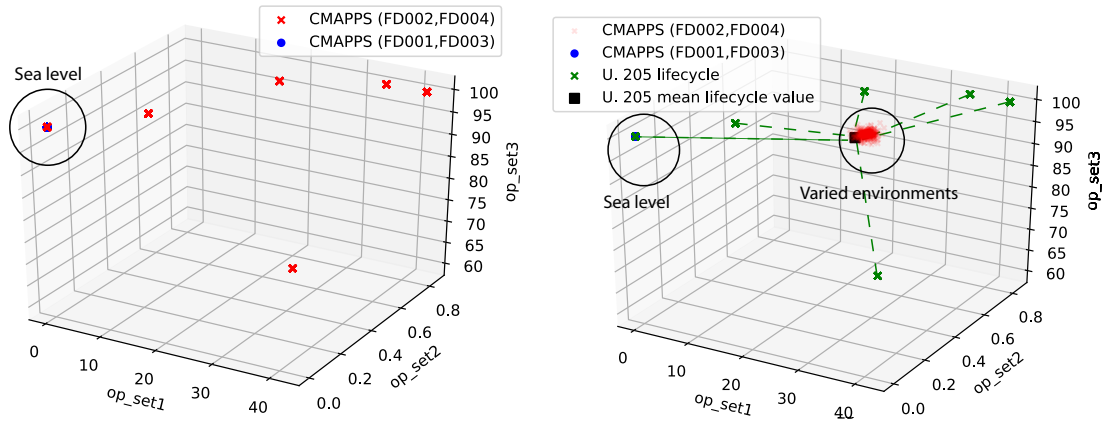


Fig. 5.5 A set of trajectories to failure extracted from the FD003 section of the data set, showing raw values of the s7 sensor (black line), its rolling average (red line), and their classification as fan degradation or HPC degradation according to a majority rule.

5.4.2 Experiments

In this section, the experiments designed to check whether this implementation fulfils the properties of Advanced Multi-Agent Systems are described. Fig. 5.7 shows how the C-

[4]Window size = 40.



(a) (b)

Fig. 5.6 Figure showing the different operational regimes in the C-MAPSS dataset (a) shows a scatter plot of the regimes (b) shows a scatter plot of the temporal mean of the regimes, clearly showing that only machines in sea-level maintain their operational setting constant.

MAPSS engine degradation data set and the PHM08 data set have been used in each of the experiments.

1. **Distribution** is a design characteristic of this implementation. As such, it is not necessary to design experiments explicitly targeted to show it. This implementation achieves distribution by executing parallel python scripts, acting as agents that communicate with each other using the websocket protocol. Agents use this protocol regardless of whether the system is run within a single or several computers. As an initial proof of concept of the implementation, an experiment is designed featuring two fleets of assets with stable environmental and failure properties. In this experiment, only sea-level trajectories are used, divided into ten sets of twenty trajectories, each set representing a Virtual Asset. Clustering is made possible because some of these machines include failures due to HPC degradation, and some other due to fan degradation (there is no mix up) (see Fig. 5.7).

2. **Flexibility**: to test whether the presented implementation is able to continuously adapt to changes in the environment and in the agents, two experiments are devised: one featuring assets with changing failure types, and one with assets with varying operational settings (see Fig. 5.7). In the first experiment, machines are made to alternate HPC failures and fan degradation failures. In the second one, assets operating at sea level are brought to higher altitudes. Apart from flexibility with regards to asset and environmental properties, the system is automatically adaptable to changes in the sensor readings, as these are immediately leveraged in the agent's prognostics.

3. **Scalability**: to test for scalability, the C-MAPSS engine degradation data set and the PHM08 prognostics data set are combined to produce a fleet to 46 assets that undergo recurrent failures (see Fig. 5.7). The system is then tested on this fleet.

4. **Resilience**: to test whether the system is resilient, faults are artificially introduced in the functioning of the agents, and in their data quality. To do so, the data generated by one or more Virtual Assets is corrupted with flat readings, outliers, and drift errors, chosen because of their prominence in industrial applications [172–174]. It must be mentioned that C-MAPSS data is by default contaminated with noise. To test for agent fault, agents are abruptly stopped without properly closing their websocket connections.

5. **Leanness**: the leanness of the proposed implementation is investigated by studying the changes that must be done in it in order to apply it to a different prognostics scenario. Similar to distribution, leanness is a design property of the system.

Experimental parameters

The experiments are performed by executing the algorithms shown in Figs. 5.3 and 5.4 with the following parameters:

- Each Virtual Asset starts with six failures as previous experience. Digital Twins train their LSTM Neural Network every time a new trajectory to failure is received.
- The architecture of the LSTM Recurrent Neural Network is $26 \times 24 \times 10 \times 2$, with the layer with 24 neurons being the only recurrent layer. `tanh` is used as activation function for all layers except the custom output layer. No Dropout of Regularisation are used.
- For Stochastic Gradient Descent, the Adam optimiser is used with a learning rate of 0.01 (see eq. (2.19)). Norms are clipped to 10 to avoid exploding gradients.
- The number of epochs for the Neural Network is 400, with a batch size of 5. Early stopping with patience 50 and minimum delta 0.05 is used through the validation loss to restrict over-fitting. The split is 5/6 training and 1/6 validation.
- For the clustering algorithm, only the last two uncensored trajectories are considered.

To assess the accuracy of the prognostics algorithm, the mean square error of the predicted time to failure is used. In this implementation, the analytics thread of the Social Platform

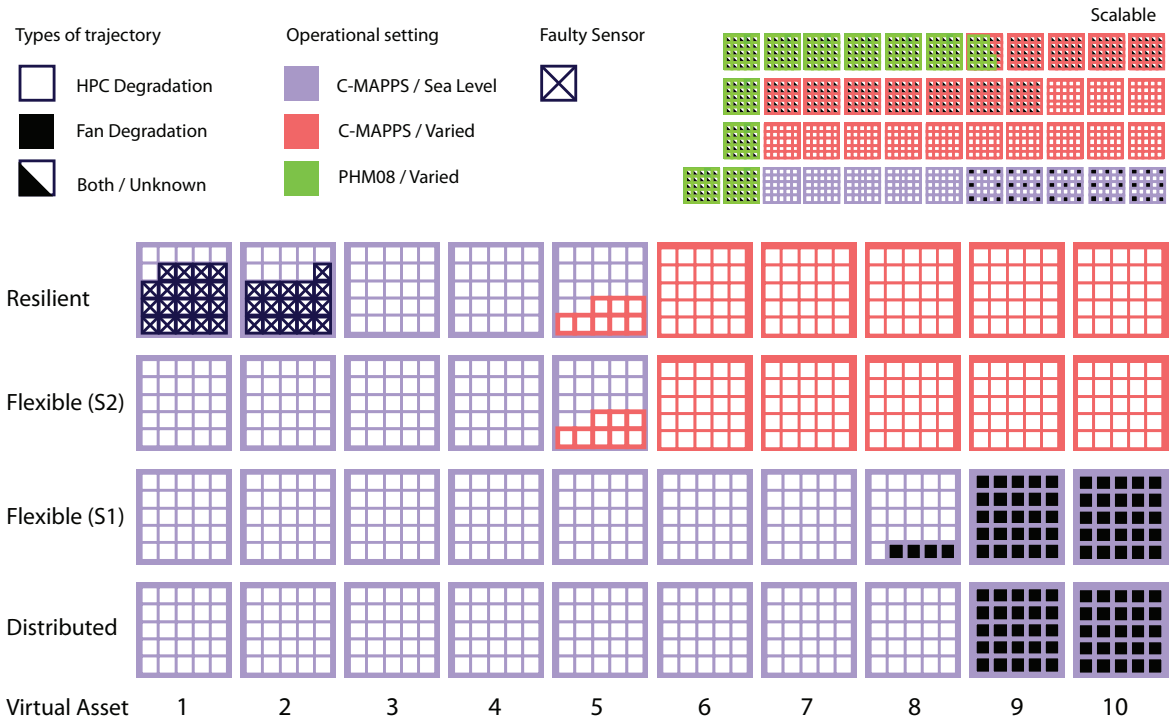


Fig. 5.7 Sketch of the re-structuring to the C-MAPSS and PHM08 data sets to conform to the experiments described in this section. Each cluster of squares represents an asset going through 20 recursive failures (each square representing a failure). Different colours represent different operational settings. Filled squares represent a machine failing due to fan degradation and empty squares due to HPC degradation. Half-filled squares represent trajectories with an unknown failure type (only featured in the scalable experiment). Squares with a cross represent trajectories where a sensor failure has been induced.

averages in real time over fleet scores and plots it to the asset manager. Notwithstanding, the results presented in this chapter are plotted using bespoke plotting algorithms. The agents are programmed so that if running as an experiment, they continuously store their variable space in the server memory, allowing for subsequent processing of experimental data. Experiments are performed on DIAL's high performance computer 'optimusprime'.

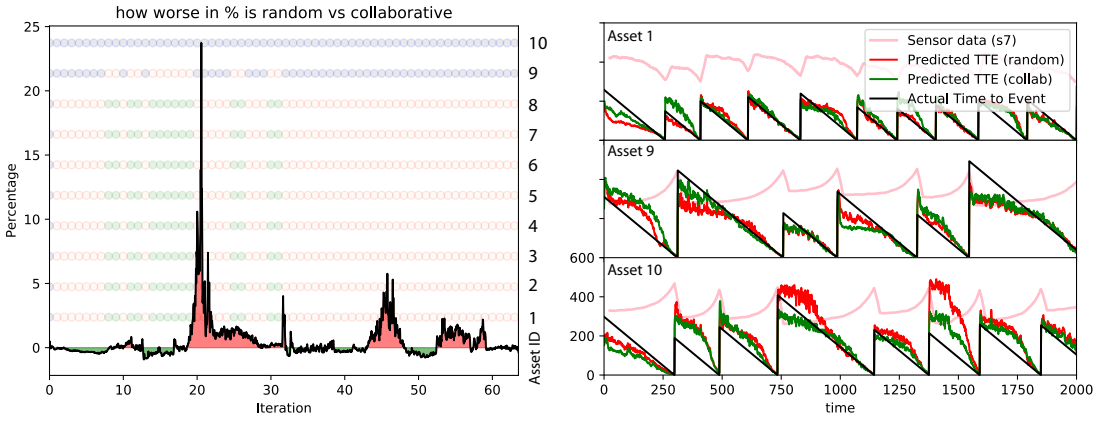
5.5 Results

This section presents the results of the experiments described in Section 5.4.2. These experiments have been designed to test whether the implementation presented in this chapter satisfies the properties of Distributed Collaborative Prognostics.

5.5.1 Distribution

To test the distributed property of the system, the corresponding experiment described in Section 5.4.2 is performed. Distribution is a property of the system by design, so this experiment is simply a validation of Distributed Collaborative Prognostics, in which the fitness of the clustering and prognostics algorithms are tested.

To test the idea of real-time distributed prognostics, an experiment in which assets learn only from similar assets (determined by the clustering algorithm) is compared to an experiment in which assets learn from a random subset of assets. The collaborative strategy is found to outperform the non-collaborative strategy (see Fig. 5.8). The system manages to cluster the assets properly for most of the time-steps: assets 9 and 10 form a cluster for 80% of the time (these are the two assets featuring fan degradation failures in Fig. 5.7).



(a) (b)

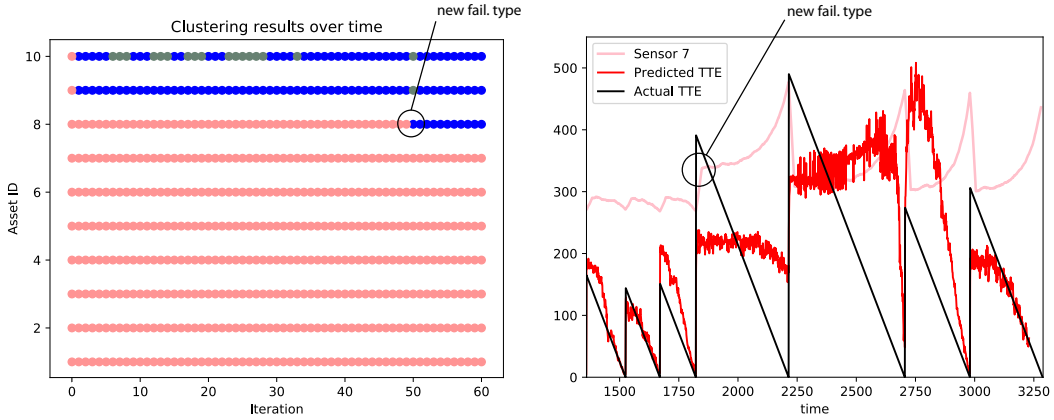
Fig. 5.8 (a) Percentile difference in accuracy between random and collaborative learning. Red shaded, time when random learning underperforms collaborative learning. Overlayed, cluster assigned to each asset. Assets 9 and 10 share a cluster as they are the only ones featuring fan degradation. (b) Data for three of the Digital Twins showing predicted time to failure for collaborative learning (green) and random learning (red). Overlayed, scaled measurements of their seventh sensor (pink). Especially for the Digital Twins corresponding to assets enduring fan degradation failures (assets 9 and 10), collaborative predictions often outperform non-collaborative predictions. True time to failure shown in black.

5.5.2 Flexibility

To test flexibility, two scenarios have been devised: one in which the failure type of some assets changes across time, and another in which the operational setting varies.

Scenario 1: varying failure

In this scenario, an asset in the fleet that has been failing only through HPC degradation is made to suddenly start experiencing fan degradation failures. The system reacts by (1) quickly clustering the asset with the rest of assets that feature a similar failure, and (2) by sharing with its Digital Twin data from the assets with which it is now clustered (see Fig. 5.9).



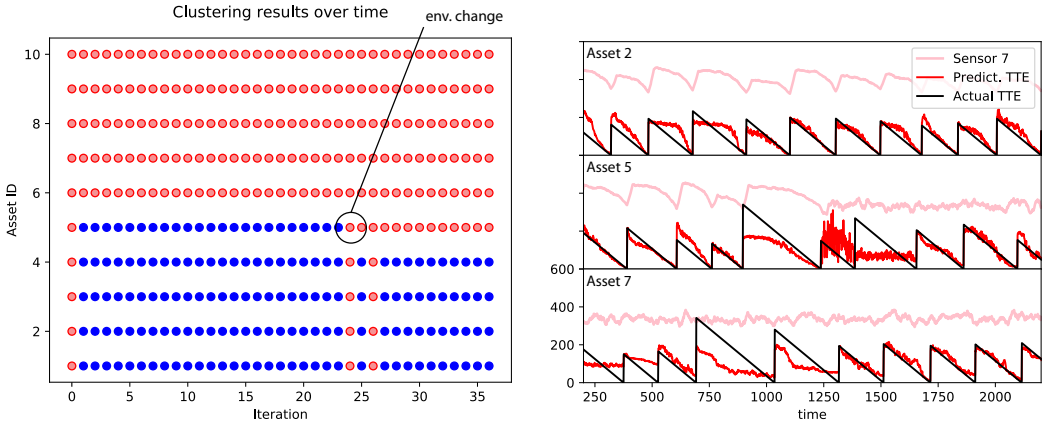
(a) (b)

Fig. 5.9 Scenario 1. (a) The cluster to which each asset belongs at each timestep is indicated by its colour. Shortly after asset number 8 starts experiencing a new kind of failures, it is clustered together with its peers. Clustering works well with the exception of short-term glitches. (b) Output from the Digital Twin of asset nr. 8. The Twin is able to learn the new failure type thanks to the data sent by its peers.

Scenario 2: varying operational setting

The operational setting of the assets is made to vary from sea level to varied environmental conditions, while making sure that their failure type is kept constant[5]. The system is able to readily react to environmental changes and cluster the assets with other assets operating in similar conditions (see Fig. 5.10). Note how the Digital Twin assigned to asset nr. 5 is able to react to the new kind of environment, predicting subsequent failures with a high degree of accuracy.

[5]The FD001/2 subsets of the data contain only fan degradation failures.



(a) (b)

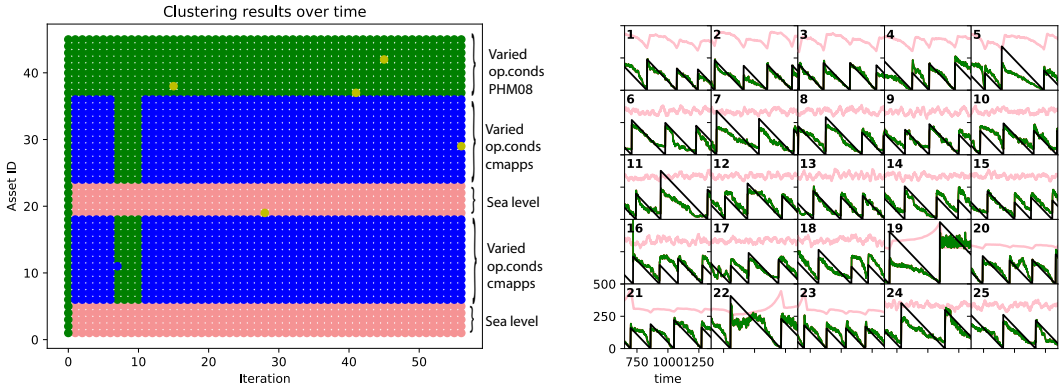
Fig. 5.10 Scenario 2. (a) The cluster to which each asset belongs at each timestep is indicated by its colour. Immediately after asset number 5 changes its environment, it is clustered together with the rest of the fleet. (b) Output from the Digital Twin of asset nr. 5: immediately after switching environment, the asset is able to react to its new environment thanks to data from its peers, even if its own data is scarce.

5.5.3 Scalability

To test whether the proposed solution is scalable, the available data sets are re-structured to represent as many assets as possible. To provide enough experimental support, each asset is made to hold a minimum of 20 trajectories to failure. The C-MAPSS data set, combined with the PHM08 prognostics data set feature 926 independent trajectories to failure that once divided by 20 yield a total of 46 assets (see Fig. 5.7). Fig. 5.11 shows a snapshot of the proposed system processing all assets simultaneously.

In theory, scalability could be considered a design property of the system; the websocket protocol is able to service tens of thousands of concurrent connections [175, 176], and as long as each asset has access to enough computational power to run its corresponding agent the system scales well. Notwithstanding this, when running the experiments in a central server as in this case, the computational demands of deep-learning pose some limitations. In this case, the number of epochs in the Recurrent Neural Network is limited to 100. However, computational demands do not pose a problem in a real industrial scenario, in which the training time of the Recurrent Neural Network is much smaller than the mean time between failures. This applies even if modest computational resources are available[6].

[6]See [82], by the author, for a more detailed discussion.



(a) (b)

Fig. 5.11 (a) Clusters formed for the assets in the data set. Clustering seems to be strongly dominated by the environmental condition, and as such the assets corresponding to FD001 and FD003 (sea level) are clustered together. For a few time steps, this makes the algorithm cluster together machines corresponding to the C-MAPSS and PHM08 data set. (b) Output from 25 assets in the data set. Shown in pink, the scaled value of sensor 7. In green, the predicted time to event, and shown in black the real time to event.

5.5.4 Leanness

To test the system for leanness, one must describe how much of the system's code and architecture has to be changed to adapt it to a different prognostics case. The leanness of the presented solution is demonstrated by using it for this thesis' case study (presented in Chapter 7). In this forthcoming chapter, the system's leanness is made evident by the fact that there is no need for new agent types in the system.

As intended by design in this implementation, the system is adapted to a different industrial scenario by modifying the code of the Virtual Asset. This is necessary because Siemens data comes in a different format than C-MAPSS. The extent of the modification is of 140 lines of code, dedicated to execute the following tasks:

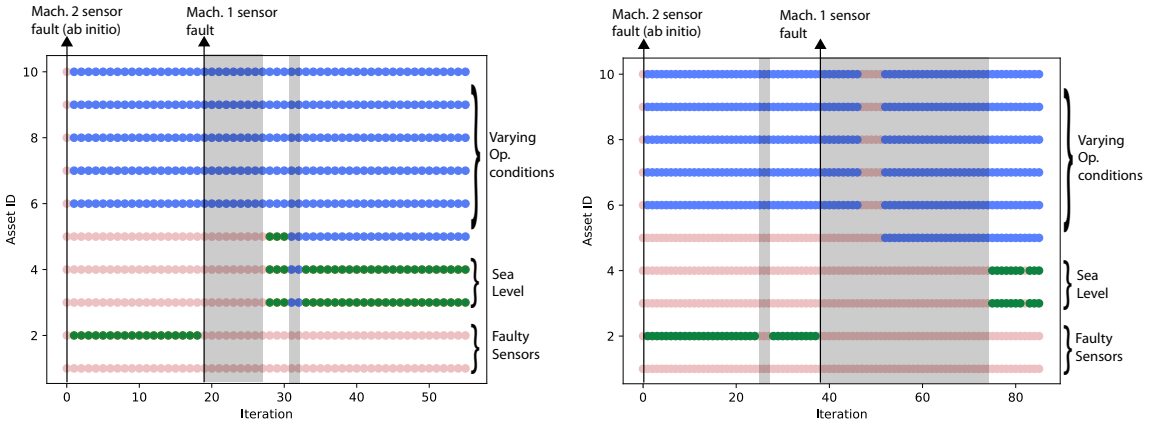
1. Import, sampling, and standardisation of gas turbine data.
2. Segmentation of data according to different events.
3. Making sure that the code works with turbines that had not yet recorded a failure event.

As argued in Chapter 3, leanness is important because it is normally a good guarantee for transferability: the ability of the solution to adapt to different industrial scenarios. Extended results for this experiment are shown in Chapter 7. The new lines of code needed to implement this thesis' case study are outlined in Appendix B.2.

5.5.5 Resilience

Resilience is tested by terminating agents at different layers of the Multi-Agent System hierarchy, and by contaminating sensor data with sensor faults. All resilience experiments are performed within Scenario 2 of Section 5.5.2 (corresponding to varying operational settings). The sensors that include faults are sensors 2, 5, 7, 9, 11, and 14 in assets 1 and 2.

1. Flat reading: flat reading faults are incorporated by making the sensors mentioned above flatten from the beginning for asset nr. 1, and from half of the experiment for asset nr. 2. Fig. 5.12a shows how once sensor faults start developing, assets are clustered away from the rest of the fleet and avoid contaminating it with their data. It is worth mentioning that despite the presence of sensor faults, the system is able to retain its flexible properties, and change the cluster of asset nr. 5 when it transitions from sea level to a varied operational regime.



(a) Sensor faults (b) Resident Outliers

Fig. 5.12 (a) Clusters for the case of assets with flat reading sensor faults. (b) Clusters formed for the assets in the case of assets with recurrent outliers. Assets containing sensor faults are separated from the rest of the fleet, and the normal fleet clustering is not affected. Shaded areas indicate times when the clustering algorithm fails to properly cluster some of the assets.

2. Outliers: large regular outliers are introduced every 30th time step. Fig. 5.12b shows how the Social Platform clusters assets with outliers together for most of the experiment. Despite not using any outlier-removal scaler, the prognostics algorithm operates well in their presence. This could be caused by their periodic nature, likely to be learnt by the Recurrent Neural Network.

3. Drift: a positive exponential drift with a cap at a large value is induced in the aforementioned sensors. A capped exponential drift is chosen because this kind of drift conforms

to the case in which sensors end up malfunctioning completely. The Social Platform is able to automatically locate the drift and cluster together assets that experience it (see Fig. 5.13).

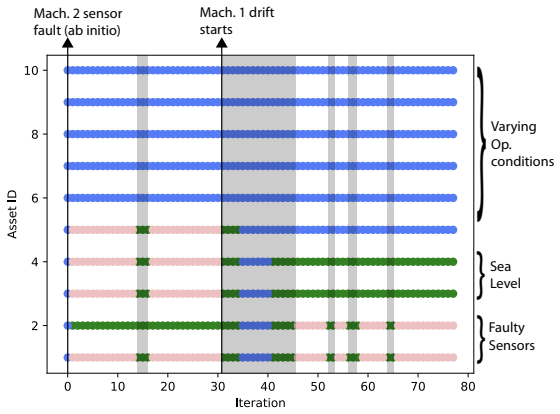


Fig. 5.13 Clusters formed for the case of assets with exponential drift. Assets containing sensor faults are automatically separated from the rest of the fleet. The normal fleet clustering is not affected. Shaded areas indicate times when the clustering algorithm fails to properly cluster some of the assets.

4. Agent failure: to test the system against sudden agent failure, agents are terminated across the Multi-Agent System by killing their corresponding python process. The results are similar to those of the multi-agent simulations presented in Chapter 6: terminating a Virtual Asset or a Digital Twin results in the halt of prognostics for the corresponding asset, without negatively affecting the overall operations of the system. Terminating the Social Platform, however, causes a total halt of the system until the platform is brought back to life.

5.6 Conclusion

This chapter presents an implementation of Distributed Collaborative Prognostics ready to be deployed in a real industrial scenario. This implementation consists of a real-time Advanced Multi-Agent System designed to operate under the conditions of non-ergodicity and dynamism typical of industrial asset fleets. This Multi-Agent System is entirely programmed in python, and uses Long Short-Term Memory Neural Networks to produce real-time individualised prognostics[7]. Communication between agents is done through the internet thanks to the websocket protocol, a widely-used protocol able to handle thousands of concurrent

[7]This implementation can be adapted to any other regression framework.

connections. These technological choices guarantee that the system is deployable in modern asset fleets endowed with IoT technologies.

Multi-Agent Systems have been postulated as a technology able to handle prognostics in dynamic and heterogeneous asset fleets. Despite several publications employing distributed systems for prognostics, this still lacks experimental support [23]. In this chapter, this possibility is shown through several experiments that realistically recreate situations that machines undergo during their operational life. Contrary to a centralised approach, agents compartmentalise the prognostics problem in multiple separated data spaces, impeding in this way the propagation of corrupt or non-relevant data.

This chapter studies different scenarios recreated using a standard prognostics data set (see Sec. 5.4.2). Perhaps the most interesting property of Distributed Collaborative Prognostics demonstrated here is its ability to adapt to unexpected conditions such as agent and sensor faults without the need of supervision. This adaptability is mainly given by a clustering algorithm run in the Social Platform that computes in real time the number of groups of collaborating assets (clusters), and the vector of collaborating assets within each cluster, $\vec{N}_i$.

Therefore, this chapter directly addresses the first and last of this thesis' research questions, as it describes in detail a Multi-Agent System with prognostics capabilities, and studies in which scenarios Distributed Collaborative Prognostics is likely to outperform traditional approaches. The presented solution conforms to the properties of Advanced Multi-Agent Systems and is capable of performing collaborative prognostics in real time. This means that it satisfies all the properties of Distributed Collaborative Prognostics described in the first sections of Chapter 3.

The main weakness of this study is that it uses computer-generated data for its experiments. This weakness is addressed in Chapter 7, in which a real fleet of gas turbines is implemented using the same multi-agent implementation.

Chapter 6

Cost implications of Distributed Collaborative Prognostics

In Chapter 4, a multi-agent implementation of Distributed Collaborative Prognostics incorporating a maintenance policy was studied. Among other caveats, the size of the asset fleet was too small, the cost equation was unrealistic, and agent failure was not considered.

This chapter is dedicated to present a study in which these caveats are solved. Here, the cost equation is more nuanced, agent failure is considered, and the size of the asset fleet is much larger. Additionally, all the multi-agent architectures presented in Chapter 3 are implemented and compared with each other[1].

The improved study presented here is used to address the third research question in this thesis, which is to study in which circumstances Distributed Collaborative Prognostics is cost-effective (see Section 1.7 for the research questions).

A significant part of the work presented in this chapter has been published as a paper in the “Journal of Intelligent Manufacturing" [149]. Maharshi Dadha (second author of the paper) coded the Distributed Clustering algorithm presented in this chapter.

6.1 Failures of the building blocks

One of the main differences between this chapter’s experiments, and the NetLogo experiments presented in Chapter 4 is that in this chapter the building blocks of the multi-agent architecture are assumed to fail. Considering component failure is important in order to study the resilience of the proposed solution, which is one of the required properties of Distributed Collaborative Prognostics (see Chapter 3).

[1] Distributed, Hierarchical, Centralised, and Heterarchical.

The effects of the failure of each of the architectural components used in this chapter follow.

- **Virtual Asset**: The failure of a Virtual Asset causes the immediate halt of communications between it and its assigned higher-level agent (for example, a Digital Twin). Thus, prognostics for the physical assets assigned to the Virtual Assets are immediately halted and only resumed once the Virtual Asset has been brought back into operation.
- **Digital Twin**: Upon failure, all communications between the Digital Twin and other agents in the architecture are severed. All the analytics performed by the Twin are also halted, which means that the Twin stops computing prognostics and giving maintenance recommendations. A secondary effect of this halt is that the Digital Twin stops consuming computational resources.
- **Mediator Agent**: As in the case of Digital Twins, the failure of a Mediator Agent supposes a halt of its communication and computation operations. This has a more dramatic effect than Digital Twin failure, as maintenance recommendations for several assets are stopped at once, potentially incurring in larger financial losses.
- **Social Platform**: As for the rest of the agents, communications and analytics are severed upon failure. In the case of a centralised architecture, a Social Platform failure corresponds to the halt of all maintenance recommendations.

6.2 Cost model

When considering whether a specific multi-agent architecture should be implemented in an asset fleet, it is crucial to consider its overall cost. To do so, it is important to take into account the value of the assets compared to each of the cost components of the system. For example, it intuitively makes sense to provide a fleet of expensive cars with sensors, wireless communications and processors in order to monitor their health state, but this may not be true for a fleet of light bulbs. Does each light bulb really need its own prognostics model, wireless connectivity and dedicated software agent? To figure out the answer to this question one must carefully account for all the costs in the system.

The cost incurred by Distributed Collaborative Prognostics can be divided into three components: maintenance, communication and processing. Eq. (6.1) gives the total cost of the system:

$$C_{\mathrm{T}} = C_{\mathrm{M}} + C_{\mathrm{C}} + C_{\mathrm{P}} = N_{\mathrm{C}}\Gamma + N_{\mathrm{P}}\gamma + C_{\mathrm{C}} + C_{\mathrm{P}}, \tag{6.1}$$

where C_T is the total cost of the system, C_M is the cost of maintenance, C_C is the communication cost, and C_P is the processing cost. Here it will be assumed that processing costs include the cost of installation of sensors and processors. The right hand side of the equation shows the decomposition of each cost component in its individual contributions. The maintenance cost is formed by the cost of a predictive maintenance action, γ, multiplied by the number of times that predictive maintenance has been performed, N_P, added to the cost of correctively maintaining one asset Γ multiplied by the number of corrective maintenance actions N_C.

This study assumes that corrective maintenance corresponds to the full replacement of a failed asset. And that its cost is proportional to the acquisition cost of the asset, $\Gamma \propto C_A$. In the text that follows, high values of Γ correspond to high-value assets.

As mentioned in the last chapter, predictive repairs are much cheaper than corrective repairs ($\gamma = \alpha\Gamma$ where $\alpha \ll 1$). Eq. (6.1) can be re-written as:

$$C_T = \Gamma(N_C + \alpha N_P) + C_C + C_P. \tag{6.2}$$

The exact amount contributed by each of the cost components depends on the specificities of the industrial implementation. However, the cost of processing a byte of data and sending it through the internet can be assumed to be constant across different industries. To compare between different asset values, it is useful to normalise eq. (6.2) to the corrective maintenance cost Γ:

$$C_t = N_C + \alpha N_P + N_{Co} C_c + N_{pro} C_p, \tag{6.3}$$

where sub-indices indicate normalisation. N_{Co} corresponds to the number of constant-size (a given byte amount) communications between agents in the system. N_{pro} corresponds to the number of times a given computational resource measure (i.e. one flop) has been used.

As done in eq. (4.13), it is important to normalise the cost to the number of assets present in an experiment, N, and the total time of the simulation T:

$$K = \frac{1}{TN}\left(N_C + \alpha N_P + N_{Co} C_c + N_{pro} C_p\right). \tag{6.4}$$

6.3 NetLogo implementation

In this study, the architectures described in Chapter 3 are implemented using the Multi-Agent simulation software NetLogo. To do so, each of the agents composing the architectures has to be programmed, together with the routines computed by their sub-components. The pseudocode of the agents used in this implementation is shown in Appendix B.1.

With regards to these routines, prognostics is performed using python's least squares fit instead of MATLAB's `lsqnonlin`, as this showed to be computationally more efficient. Computational speed is important in this experiment as much larger fleets of assets are simulated compared to the implementation in Chapter 4. To ensure scalability, the data used by each agent is limited to the 400 most recent Health Indicator data points.

For the case of the Hierarchical, Heterarchical and Centralised architectures, clustering is performed by the Social Platform using the `scikit-learn` library k-means clustering algorithm. For the case of the Distributed Architecture, a distributed clustering algorithm (written by Maharshi Dadha) is used. It is briefly described here for completeness.

6.3.1 Distributed clustering algorithm

The distributed multi-agent architecture is an architecture characterised by being formed by a single type of agent: Digital Twins (see Section 3.2.2 for a detailed description). As such, clustering has to be implemented in a distributed way, taking into account the fact that each Digital Twin has only partial information about the rest of the fleet.

This clustering algorithm has the goal of forming k clusters of assets, where k is the number of different types of assets in the fleet. The general flow of the algorithm follows: first, a random asset in the fleet is chosen as a centroid. The agent assigned to this asset then calculates the distances of the other agents to itself using the available information that it has regarding their Health Indicator values. It then assigns the farthest agent to the identity of the second centroid. This is repeated iteratively. Once all initial centroids are assigned. Distances to the centroids are re-calculated from each agent and clusters are assigned. The pseudocode for the algorithm follows:

Algorithm 2 The distributed k-means clustering algorithm implemented in this chapter.

```
Select one random agent from the fleet;
while number of centroids <k do
    for the agent selected above do
        Calculate the distances between the agents and the centroids;
        Record the distances of the agents from their closest centroid;
        Append the farthest agent to the list of centroids;
    end for
end while
These centroids represent the clusters;
for every other agent in the system do
    Calculate the distances from each centroid;
    Assign self to the cluster represented by the closest centroid;
end for
```

6.4 Results

The raw results of the experiments presented here are shown in Tables A.1 and A.2 in the Appendix A. These results are shown as quadruplets of (N_{P},N_{C},N_{Co},N_{pro}). The two first components show the number of times that a preventive maintenance, and a corrective maintenance action have been taken. The last two components show the number of communications of a fixed length, and the number of times that a fixed processing resource has been consumed. These tables can be converted to specific costs when weighted by the quadruplet of cost weights $(\alpha, 1, C_p, C_c)$ (note that the unitary component is there to ensure normalisation with the corrective maintenance cost).

To reduce the variable space of the solution, the plots shown here incorporate the same relation between the corrective maintenance cost and the preventive maintenance cost as in Chapter 4: $\alpha = \frac{1}{100}$. This means that preventive maintenance is assumed to be one hundred times cheaper than the corrective maintenance cost. Additional to this, C_{p} is chosen to be 20 times the processing cost per flop C_{c}, to ensure a significant contribution of the processing cost in the results (we will see in the results that $N_{\mathrm{pro}} \ll N_{\mathrm{Co}}$). Tables A.1 and A.2 can be used to replicate any other parametric configuration.

The results of the experiment, shown in Fig. 6.1, feature the following phenomena:

1. **For the case of agent failures, Distributed and Heterarchical architectures minimise costs for high-value assets:** this also applies to the case of low communication costs. In Fig. 6.1, compare the green and yellow dashed lines with the blue and red dashed lines for $C_{\mathrm{c}} \leq 10^{-4}$. This difference occurs because in the case of Distributed and Heterarchical architectures, the failure of a single higher-level agent such as the Social Platform or a Mediator Agent results in the halt of maintenance operations for several agents in the fleet, increasing maintenance costs. This can be explicitly seen in the number of corrective maintenance actions appearing in Tables A.1 and A.2 in the Appendix. For larger communication costs, or low-value assets, this increase in corrective maintenance costs in not enough to overcome the much higher communication costs of the Distributed and Heterarchical architectures.

2. **Low-value assets have a larger normalised cost per asset**: this means that collaborative prognostics is more cost-efficient (relative to the individual asset cost), the more expensive the assets in the fleet are.

3. **If agents can't fail, Centralised and Hierarchical architectures are cost-competitive**: this is an expected result because the characteristics of these two architectures imply less communications and processing costs than their more distributed counterparts.

There are some conditions, however, where this effect is not observed: large values of noise in the system, very low communication costs, or very high asset values.

4. **If agents are not assumed to fail, increased asset value means less difference between architectures**: this effect can be clearly observed in Fig. 6.1 from the convergence of solid lines as the asset value increases (left of the figure). This is due to the prominence of maintenance costs, that in the case of no agent failures are similar for all the architectures.

5. **The high-communication/low asset value cost limit**: for very high communication costs or low asset values, the presence of agent failures actually lowers the total cost. This has a simple explanation: despite the increase on corrective maintenance actions caused by agent failure, there will always be a point in which communication costs are high enough so that the money saved by halting communications is higher than the money spent by more corrective maintenance.

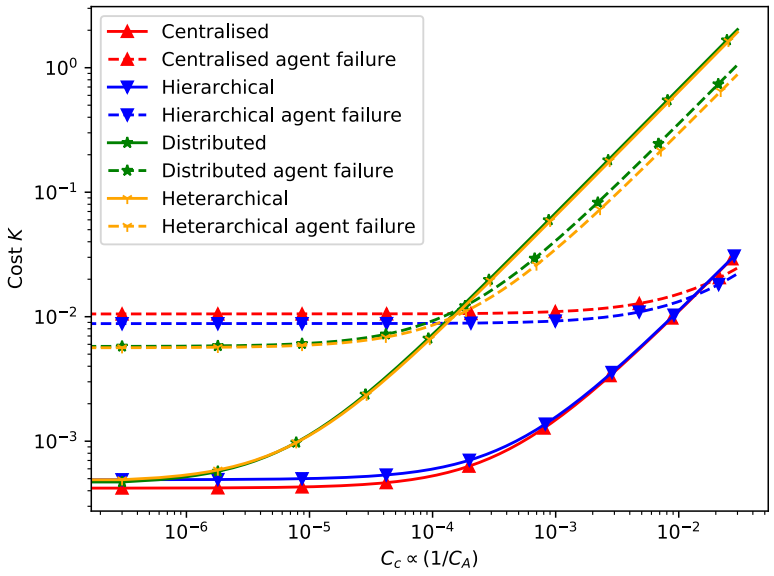


Fig. 6.1 Normalised cost K for each of the architectures studied in this thesis with respect to the normalised communication cost C_c or the inverse asset value $1/C_A$ (for a constant C_c). Dashed lines indicate the case of agent failures, and solid lines the case of no agent failures. Data shown for $\sigma = 0.1$, $\alpha = \frac{1}{100}$, and $C_p = 20C_c$.

Hitherto, the dependence with different cost parameters and architectures has been studied. Chapter 4, showed that the dependency with the noise present in the Health Indicator was also important. Intuitively, one would expect that the difference between architectures would decrease the more noisy the system becomes due to the dominance of corrective maintenance

costs caused by unpredicted failures. To check whether this is true, it is useful to calculate the normalised index of dispersion, D_σ^{norm}, across different noise levels:

$$D_\sigma^{\text{norm}} = \frac{1}{\max_\sigma (D_\sigma)} \frac{\text{Var}\left(K_{\text{cent}}^\sigma, K_{\text{hier}}^\sigma, K_{\text{dist}}^\sigma, K_{\text{hete}}^\sigma\right)}{\text{Mean}\left(K_{\text{cent}}^\sigma, K_{\text{hier}}^\sigma, K_{\text{dist}}^\sigma, K_{\text{hete}}^\sigma\right)}. \tag{6.5}$$

Where $\text{Var}\left(K_{\text{cent}}^\sigma, K_{\text{hier}}^\sigma, K_{\text{dist}}^\sigma, K_{\text{hete}}^\sigma\right)$ is the variance of the cost per time and asset across all different architectures at a given value of σ. Similarly, $\text{Mean}\left(K_{\text{cent}}^\sigma, K_{\text{hier}}^\sigma, K_{\text{dist}}^\sigma, K_{\text{hete}}^\sigma\right)$ is the mean of the cost per time across all the architectures. $\max_\sigma (D_\sigma)$ is the maximum index of dispersion measured across all values of σ.

The results obtained from calculating this index are shown in Fig. 6.2. This figure shows that the cost variation between architectures decreases as noise increases when communication costs are low enough, or asset value is high enough. It must be mentioned that a communication cost of 0.2 means that per each communication that the agent performs, it will incur on an expense equivalent to 20 % of the cost of the asset. This is hardly realistic, so for most applications a system with higher noise will mean less differences between architectures. The reversion at high communication costs is given by the difference in clustering results between the architectures, that increases with σ (see tables A.3 and A.4 in the Appendix), and by the vanishing importance of corrective maintenance costs. The practical takeaway from this result is that the more accurate a prognostics algorithm gets, the more economical gain there is from appropriately choosing its multi-agent architecture.

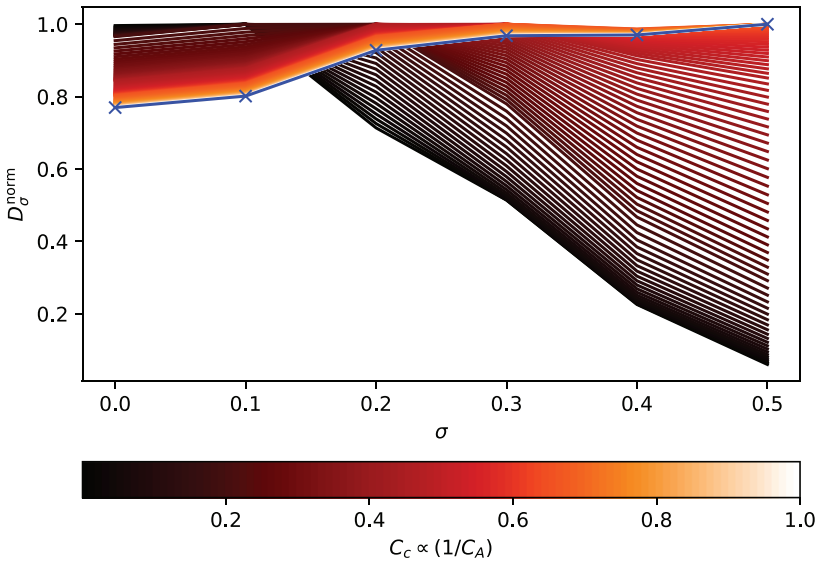


Fig. 6.2 Normalised index of dispersion D_σ^{norm} with respect to the noise σ. The color bar indicates dependency with normalised communication cost C_{c} or the inverse asset value $1/C_{\text{A}}$. The blue line with markers represents the low asset value / high C_{c} limit.

6.5 Architecture selection

The experimental procedure presented earlier in this chapter can be used to select the best multi-agent architecture for a collaborative prognostics (or other kinds of distributed prognostics) application. An asset manager faced with the question of choosing among different architectures should adhere to the following steps:

1. Estimate the corrective and predictive maintenance costs of the fleet's assets: (Γ, γ).
2. Estimate the cost of communicating and processing a standardised unit of data (C_C, C_P).
3. Determine the accuracy of real-time prognostics. This accuracy can be encoded in eq. (4.1) through the noise term. If hard to estimate, assume a low-variance noise term (as this prepares the system for the case of maximum difference between architectures).
4. Determine the fleet properties, for example the number of assets N.
5. Select a maintenance policy, and code it in the simulation software.
6. Estimate the probability of agent failure and the maximum time of agent downtime, encode it in the simulation as described in the preceding paragraphs.
7. Test the different architectures available to the asset manager as proposed in this chapter, and compare the cost incurred by them.
8. Choose the most suitable architecture from the simulation results.

6.6 Conclusion

This chapter studies the cost implications of using different multi-agent architectures in Distributed Collaborative Prognostics. The main conclusion that can be drawn from its results is that the individual value of the assets in the fleet is a key element to evaluate whether a distributed approach is economically viable.

The results of this chapter quantify a known qualitative result: only when communication and processing costs are low enough with respect to the cost of the asset, decentralised approaches make sense. This result must be put in context: with the continued decrease of the cost of IoT technologies more and more types of assets are expected to fall into this category. This means that Distributed Collaborative Prognostics can be expected to become increasingly relevant in the future.

Another key result of this study is the importance of agent failures in the cost of the system: if agents fail, more centralised architectures are costlier because a large number of crucial tasks rely on a minority of agents.

Although this study comes closer to a realistic cost analysis than the study presented in Chapter 4, it still has two important drawbacks. The first one is the absence of real-time optimisation of the predictive maintenance policy (given by a fixed value of η). In Distributed Collaborative Prognostics, this policy would be optimised in real-time. The second one is the simplistic prognostics model with strong prior assumptions (a simple parametric non-linear fit instead of a neural network).

This chapter addresses the first and last of this thesis' research questions. The first question, regarding the use of Multi-Agent Systems in Distributed Collaborative Prognostics is addressed by providing an estimation of the cost consequences of agent failures. On top of this, this chapter extends the experiments from Chapter 4 to realistically large fleets of assets. The last research question, referring to the specific circumstances in which Distributed Collaborative Prognostics overcomes a traditional approach, is addressed by comparing a Centralised architecture with decentralised architectures from a cost perspective.

In the next chapter, the Distributed Collaborative Prognostics implementation presented in Chapter 5 is used in an industrial case study. This implementation uses Recurrent Neural Networks for real-time prognostics, thus dropping any assumption on the nature of the deterioration process.

Chapter 8

Conclusion and future work

This chapter presents the conclusions to this thesis. The chapter is composed of six sections. The first section is dedicated to the general conclusions, in which the research questions of this thesis are revisited and a summary of the results of this thesis is presented. Following this, the second section explores the caveats of the presented tool, including its large operational cost and its tendency to over-fitting. The third section provides an overview of this thesis' contributions to the academic knowledge, including a brief description of each published journal paper. The fourth section outlines the technologies used in this thesis, and discusses possible alternatives. The fifth section discusses this thesis' contribution to industrial practice. This chapter ends with a section dedicated to discuss the future work.

8.1 General conclusions

This thesis presents a tool, Distributed Collaborative Prognostics, designed to operate in the conditions of industrial fleets of assets: non-ergodicity and dynamism. It does so by profiting from the properties of Multi-Agent Systems, especially designed to handle such conditions. The design and development of this tool has drawn on three different fields: reliability, machine learning, and asset management. Distributed Collaborative Prognostics uses machine learning algorithms to provide prognostics and maintenance recommendations in real time. These recommendations are tailored to each specific asset by means of an agent that is assigned to it. Agents are programmed so that they communicate with each other in order to improve their prognostics and learn from the experiences of their respective assets.

This thesis is, therefore, not so much an incremental addition to a given field of research, nor a description of new experimental findings. Rather, it helps broadening an existing field (asset management) by connecting it to other research fields. In doing so, it presents the first comprehensive attempt to design a tool that uses machine learning and Advanced

Multi-Agent Systems to solve the prognostics problem in real industrial scenarios. The key addition introduced by Distributed Collaborative Prognostics is collaboration: for the first time, the agents used in a prognostics framework based on Multi-Agent Systems are able to horizontally communicate with each other in order to improve their predictions. This is shown to be vital for the prognostics solution to operate in realistic industrial scenarios.

In the process of designing this tool, this thesis answers three research questions:

1. *Theoretical*: How can Multi-Agent Systems be used for prognostics in asset fleets? Although Multi-Agent Systems is a well-known paradigm, its utilisation in prognostics has been so far limited. This thesis answers this research question in three steps. First, Chapter 3 shows that the properties required for Distributed Collaborative Prognostics overlay with the properties of Advanced Multi-Agent Systems. These include their real-time nature, scalability, resilience, etc. Second, Chapters 4 and 6 show how Multi-Agent Systems can be used to calculate the effect of Distributed Collaborative Prognostics in theoretical scenarios. Third, Chapters 5 and 7 present and experimentally test a fully operational implementation of Distributed Collaborative Prognostics using an Advanced Multi-Agent System programmed in python.

2. *Technical*: How can Distributed Collaborative Prognostics be used in conjunction with predictive maintenance? This research question is addressed in Chapters 4 and 6, in which NeLogo is used to compute the effect that a predictive maintenance policy has in the cost of the system. It is found that a time-based replacement policy approximates well the optimal maintenance time for predictions updated in real time. Apart of this, agent failure is found to have important effects on maintenance cost, as it results in the halt of preventive maintenance actions.

3. *Practical*: Under which circumstances Distributed Collaborative Prognostics outperforms traditional approaches? This thesis compares Distributed Collaborative Prognostics with traditional approaches from a cost and managerial perspective (in Chapters 4 and 6), from an accuracy perspective (Chapters 4, 5 and 7), and from a practical or industrial perspective (Chapter 7). All findings point towards one general conclusion: Distributed Collaborative Prognostics is ideal for large fleets of expensive assets that operate in dynamic environments.

The agents in the Multi-Agent Systems developed in this thesis perform prognostics using a combination of traditional reliability techniques (survival analysis), and modern machine learning techniques (LSTM Recurrent Neural Networks). These techniques are used to incorporate information from the asset's sensors in a regression model capable to perform accurate predictions in real time.

From the experiments performed in this thesis, comprehending both simulated and real data, it can be concluded that Distributed Collaborative Prognostics outperforms competing approaches in the following cases: (1) For the case of low sensor noise and a small fleet of assets with a previously-known deterministic deterioration process and a real-time predictive maintenance policy as in Chapter 4. (2) In a large fleet of assets, featuring a substantial probability of agent failures, with the rest of conditions conforming to case (1) as in Chapter 6. (3) In a small fleet of synthetic industrial assets without previous knowledge on their deterioration processes, high sensor noise, dynamism, non-ergodicity, and sensor and agent failures as in Chapter 5. (4) In a medium-size fleet of real industrial assets (industrial gas turbines) with all the conditions of case (3) as shown in the case study of this thesis, presented in Chapter 7.

As part of this, it has been shown that Distributed Collaborative Prognostics can handle situations typical of real-life industrial scenarios such as agent failures, sensor failures, etc. This is in large part achieved thanks to unsupervised learning methods such as clustering algorithms that automatically group machines according to their operational status. These conditions represent the dynamic and heterogeneous properties of real industrial systems that make of prognostics a difficult task.

This work finds experimental support in an unusually large data set of industrial data. The lack of appropriate run-to-failure data sets is one of the principal problems in the field of prognostics. This drives a large portion of research studies to rely on simulated data such as for example the C-MAPSS data set [169]. Thanks to the collaboration with Siemens Turbomachinery, this thesis is able to overcome this problem and demonstrate the validity of its findings also within a realistic industrial data set.

8.2 Caveats

Distributed Collaborative Prognostics as presented in this thesis is not a blanket solution to every prognostics scenario. Its most important weakness is at the same time its biggest strength: its complexity. Compared to traditional approaches (including a centralised approach), Distributed Collaborative Prognostics employs many more agents, and trains multiple prognostics models in real time, consuming much more computational resources (see Chapter 6). This increase in complexity has two negative consequences: (1) a higher operational cost, and (2) a tendency to produce over-fitting.

(1) Has been discussed at length in Chapter 6 and in [149]. The cost benefits of Distributed Collaborative Prognostics compensate the increase in operational cost if the value of the assets is significantly larger than their processing and communication costs.

(2) Refers to the tendency for a model to learn patterns from a portion of the data set that do not generalise to the complete data set. Distributed Collaborative Learning divides an asset fleet in smaller clusters of assets that then share data with each other. This split decreases the size of the training data set for the prognostics algorithms run in the agents compared to a centralised scenario, and conflicts with the fact that some optimisation frameworks such as Neural Networks benefit from large data sets [192]. In theory, a centralised approach with a single Recurrent Neural Network could be expressive enough to adapt to all the different types of machines in the fleet whilst transferring information from one cluster to the other without negatively affecting performance. In practice, however, it has been shown that this is not always the case (see the results in Chapter 7). Intuitively, the better defined the cluster of assets are, the smaller this problem is. This has been shown to be the case for simple theoretical scenarios (see Chapter 4).

8.3 Contribution to the academic knowledge

Academically, the work presented in this thesis has contributed to the field of reliability in three ways: (1) by proposing the concept of distributed collaborative learning, as published in [38, 49], (2) by studying the architectures best suited for distributed collaborative learning, and the economical implications of coupling them to a maintenance policy (as shown in [149, 193]), and (3) by proving the applicability of Distributed Collaborative Prognostics for real industrial scenarios (as shown in [23] and the last chapter of this thesis).

The following paragraphs detail the academic contributions of the journal papers published during the elaboration of this thesis, ordered according to antiquity. Conference papers are not discussed, as they include findings later published in the papers that follow (see [82, 193]), or explore problems tangential to the main topic of this thesis such as dimensionality reduction (see [85]).

- [38] Adrià Salvador Palau, Z. Liang, D. Lütgehetmann and Ajith Parlikad. *Collaborative prognostics in social asset networks. Future Generation Computer Systems (Accepted in 2018, published in 2019)*: This paper is the first piece of work to present the concept of real-time collaborative learning as a solution for prognostics without the need for extensive historical data. Distributed Collaborative Prognostics is shown to have a faster convergence and lower cost than self-learning and fleet-wide approaches, and the effect of using a traditional maintenance policy to provide prognostics in real time is studied.

- [49] Hao Li, Ajith Parlikad, and Adrià Salvador Palau. *A Social Network of Collaborating Industrial Assets. Proceedings of the Institution of Mechanical Engineers (2018)*: this paper embeds Distributed Collaborative Prognostics within the context of Social Asset Networks developed by Li and Parlikad. This paper introduces the concept of a Social Platform, which then will become one of the crucial agents in the multi-agent architectures used to deploy Distributed Collaborative Prognostics.

- [149] Adrià Salvador Palau, Maharshi Dhada, and Ajith Parlikad. *Multi-Agent System architectures for collaborative prognostics. Journal of Intelligent Manufacturing, 2019.* This paper presents an analysis of the implications of implementing different Multi-Agent System architectures for collaborative prognostics and real-time maintenance planning. Agent failures and a large fleet of assets are considered. This paper also provides a method to design cost-effective Multi-Agent Systems for predictive maintenance.

- [23] Adrià Salvador Palau, Maharshi Dhada, Kshitij Bakliwal, and Ajith Parlikad. *An Industrial Multi Agent System for real-time distributed collaborative prognostics. Engineering Applications of Artificial Intelligence, 2019.* This paper presents the first deployable implementation of Distributed Collaborative Prognostics. Its main research contribution is that it compares its properties with those of Advanced Multi-Agent Systems. This means that this paper presents one of the first successful implementations of Advanced Multi-Agent Systems in real industrial scenarios. Additionally, this paper shows a real-time prognostics algorithm able to handle sensor faults and agent failure.

8.4 Technologies

This thesis features a multidisciplinary piece of work that includes several technologies. This section summarises the *practical* knowledge that has been generated by developing Distributed Collaborative Prognostics, and the most important technological choices that have been made to make it an operational tool.

- *On Multi-Agent Systems*: when it comes to the development of a Multi-Agent System, there are two different phases that require different tools: the phase of theoretical design and simulation, and the phase of industrial development, deployment and validation. For the first phase, NetLogo has been found to be the best option, as it is light-weight, easy to program, and can be integrated with python, MATLAB, and even with Java code snippets if high-speed agent processing is required. For the second phase, the best option has been found to be python. This decision has been made because python

balances access to state of the art machine learning libraries with agent communication through the internet. A Multi-Agent System-specific framework is not used because, at the time of this thesis, none was found that 1) could be deployed industrially, 2) could compile the necessary python libraries.

- *On the physical implementation of the system*: Distributed Collaborative Prognostics requires running multiple agents in real time. Aggregated, this corresponds to a large amount of computational resources. For a physical implementation, this adds a considerable cost component to the system. In this thesis, two approaches have been considered: either running the Multi-Agent System in a high performance computer controlled by the asset owner, or running it through service providers such as Amazon Web Services or others. Due to most service providers charging per hour, an in-house computer has been found to be the cheapest option. This contrasts with other machine learning tasks in which the training and prediction of the algorithms is done separately, and for which services such as AWS can be cost effective. Running the agents directly in the assets (as in edge computing) has also been considered and programmed into Distributed Collaborative Prognostics. However, for security and intellectual property reasons most industrial partners prefer avoiding it.

- *On prognostics*: regression frameworks based on survival analysis are the natural choice for time to failure prediction. However, they require large amounts of data to appropriately fit distributions such as the Weibull distribution using maximum likelihood estimation. From a practical perspective, if the number of trajectories to failure is limited, a Gaussian-based loss function is found to be more numerically stable. Python is used to deploy real-time training of Recurrent Neural Networks. Python is found to be computationally efficient for prognostics, as failure events occur rarely enough to leave sufficient time to the training of the Neural Network. In real fleets of machines, an additional problem consists of handling the large amount of uninformative data. This thesis proposes a classification approach with two classes: within the prediction time window, and outside the prediction window. For this approach, off-the-shelf classification algorithms provide good accuracies (see Chapter 7).

- *On clustering*: appropriately choosing the groups of collaborating assets is an important element of Distributed Collaborative Prognostics. From the conversations maintained with several industrial contacts, it seems clear that semantic information about the properties of the assets is often the most important element to assess their difference. For example, the model of a car and the factory where the car was produced contain

more information about its difference with another car than real-time sensor readings. From a practical perspective, this is incorporated into the clustering algorithm of choice by adding semantic-based features to each of the clustering individuals. With regards to clustering algorithms, this thesis found DBSCAN to be the algorithm providing the best results, as it was capable of adapting to different industrial scenarios and feature dimensions.

8.5 Contribution to industrial practice

This thesis outlines a deployable solution for Distributed Collaborative Prognostics. This solution can be used to guide industries in the development of their own prognostics tool, or can be directly adapted to provide prognostics in existing asset fleets. Additionally, this thesis can be used to inform industrial practice on the cost consequences of deploying distributed prognostics. The results of Chapters 6 and 7 can be directly used by asset managers in order to take specific managerial decisions on their multi-agent architecture and IoT strategy.

This thesis main managerial contribution is to aid industry in its conversion to servitisation. Distributed Collaborative Prognostics is designed to be transferable across different industrial scenarios, and can be adapted to perform other asset management tasks such as maintenance planning and condition monitoring. Apart of a deployable engineering solution, this thesis provides an example (in its case study) of how it can be implemented in practice for alert prediction. This case study was performed in conjunction with Siemens' Digital Transformation team, and has been showcased to the Siemens leadership as an example of the potential cost savings attainable through a careful use of modern prognostics approaches.

During private conversations held with different industrial partners, many mentioned that they had embedded their assets with sensors and connection to the internet without having a good estimation of the cost implications that this action would entail. Despite this, they often coincided on their interest for a shift towards servitisation, and their hope that condition monitoring and machine learning would benefit their bottom-line in the mid-term. This thesis shines light on this hope by providing guidelines on the suitability of condition monitoring and real-time distributed prognostics depending on the value of the assets and communication and processing costs. The steps proposed in Section 6.5 can be followed to get an estimate of the different cost components of implementing a solution of the likes of Distributed Collaborative Prognostics.

8.6 Future work

Future work should concentrate on three fronts: Theoretical, Experimental and Developmental.

1. *Theoretical:* the clustering algorithm should be improved in order to include a metric that does not only weight features and asset make, but also information theory criteria about the amount and quality of data in each cluster. This, in theory, could be used to bypass the problem of overfitting mentioned before in this chapter, as agents would cooperate with more similar groups of assets only if the quality of their data justified to do so. Additionally, the clustering algorithm should be specifically designed for the case of physical assets, and adapted to the number of clustered individuals present in realistic industrial scenarios. Clustering algorithms like HDBSCAN (a hierarchical clustering adaptation to DBSCAN [167]), and other hierarchical clustering implementations should be explored to see if they can be used to reveal asset similarities.

2. *Experimental:* experiments with a large fleet of real industrial assets are necessary. Even if the data set used in this thesis ranks amongst the best available for machine prognostics in terms of size and variability, the number of assets sharing a common fault is relatively small for the application of collaborative learning (on the order of 20 assets). Clustering has not been designed as an approach for a small number of objects. Unfortunately, such extensive data sets are available just to a few companies that might be reluctant to disclose their findings publicly. However, the positive experience from this thesis gives hope to practitioners that given the appropriate intellectual property assurances, prognostics with large amounts of industrial data will become increasingly common.

3. *Developmental:* all the programming done in this thesis has been approached from a perspective of scientific utility, and focused on showing the basic tenets of Distributed Collaborative Prognostics. While writing this thesis, it was found that there was a lack of non-proprietary tools to help with the development of efficient Multi-Agent Systems with deep learning capabilities. Further research should focus on publishing a distributed collaborative framework code in open source.

References

[1] International Organization for Standardization. ISO 55000:2014 - Asset management – Overview, principles and terminology - ISO, 2014.

[2] Max Cary and H. H. Scullard. *History of Rome: Down to the Age of Constantine*. 1975.

[3] Cornelis van Tilburg. *Traffic and congestion in the Roman Empire*. 2006.

[4] R. J. Forbes. *Studies in ancient technology. 2*. Studies in Ancient Technology. E.J. Brill, 1993.

[5] Gerald S. Cole. The changing relationships between original equipment manufacturers and their suppliers. *International Journal of Technology Management*, 3(3):299–324, 1988.

[6] Andy Neely. Exploring the financial consequences of the servitization of manufacturing. *Operations management research*, 1(2):103–118, 2008.

[7] A. K. S. Jardine, D. Lin, and D. Banjevic. A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, 20:1483–1510, 2006.

[8] A. K. S. Jardine. *Maintenance, Replacement and Reliability*, volume 1542. 2013.

[9] ReliaSoft Corporation. *Life Data Analysis Reference Book*. 2019.

[10] Alexandre Muller, Adolfo Crespo Marquez, and Benoît Iung. On the concept of e-maintenance: Review and current research. *Reliability Engineering & System Safety*, 93(8):1165–1187, 2008.

[11] Ryan Walker, Sureshkumar Perinpanayagam, and Ian K. Jennions. Rotordynamics Faults: Recent Advances in Diagnosis and Prognosis. *International Journal of Rotating Machinery*, 2013:12, 2013.

[12] O. E. Dragomir, R. Gouriveau, F. Dragomir, E. Minca, and N. Zerhouni. Review of prognostic problem in condition-based maintenance. In *2009 European Control Conference (ECC)*, pages 1587–1592, 2009.

[13] R. K. Pathria and Paul D. Beale. *Statistical Mechanics*, volume 37. 3rd edition, 2011.

[14] C. Cempel, H. G. Natke, and M. Tabaszewski. A passive diagnostic experiment with ergodic properties. *Mechanical Systems and Signal Processing*, 11(1):107–117, 1997.

[15] A. Ghenaiet, S. C. Tan, and R. L. Elder. Prediction of an axial turbomachine performance degradation due to sand ingestion. *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy*, 219(4):273–287, 2005.

[16] M. Tullmin and P. R. Roberge. Atmospheric corrosion. *Uhlig's Corrosion Handbook*, page 305, 2000.

[17] Dragan Djurdjanovic, Jay Lee, and Jun Ni. Watchdog agent - An infotronics-based prognostics approach for product performance degradation assessment and prediction. *Advanced Engineering Informatics*, 17(3-4):109–125, 2003.

[18] Fred M. Discenzo, Francisco P. Maturana, Raymond J. Staron, Kenwood H. Hall, Pavel Tichý, Petr Šlechta, Jan Bezdicek, and Vladimír Marík. *Dynamic Reconfiguration of Complex Systems to Avoid Failure*. Springer, 2010.

[19] Andrea Omicini. Introduction to Distributed Systems. *Materiale didattico Università di Bologna*, 1:5–7, 2009.

[20] Maarten van Steen and Andrew S. Tanenbaum. A brief introduction to distributed systems. *Computing*, 98(10):967–1009, 2016.

[21] Ernest C. Puig, Kareem S. Aggour, Jie Lu, Eric Pool, and Mina Botros. Enterprise Historian for Efficient Storage and Analysis of Industrial Big Data. In *General Electric*, 2013.

[22] Paulo Leitão and Stamatis Karnouskos. *Industrial Agents Emerging Applications of Software Agents in Industry*. Elsevier, 2015.

[23] Adrià Salvador Palau, Maharshi Dhada, Kshitij Bakliwal, and Ajith Parlikad. An Industrial Multi Agent System for real-time distributed collaborative prognostics. *Engineering Applications of Artificial Intelligence*, 85:590–606, 2019.

[24] Hyacinth S. Nwana. Software agents: an overview. *The Knowledge Engineering Review*, 11(3):205–244, 1996.

[25] Ming Tan. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

[26] Liviu Panait and Sean Luke. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[27] Paul Guyer and Rolf-Peter Horstmann. Idealism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 201 edition, 2018.

[28] Alexander Miller. Realism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 201 edition, 2016.

[29] Catherine Legg and Christopher Hookway. Pragmatism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 201 edition, 2019.

[30] Peter Markie. Rationalism vs. Empiricism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2017 edition, 2017.

[31] Thomas Dabney, Luis Hernandez, Philip A. Scandura, and Roger Vodicka. Enterprise health management framework - A holistic approach for technology planning, R&D collaboration and transition. *2008 International Conference on Prognostics and Health Management, PHM 2008*, 2008.

[32] N. Papakostas, P. Papachatzakis, V. Xanthakis, D. Mourtzis, and G. Chryssolouris. An approach to operational aircraft maintenance planning. *Decision Support Systems*, 48(4):604–612, 2010.

[33] Andrew K. S. Jardine. Optimizing condition based maintenance decisions. *2002 Proceedings Anual reliability and Maintanability Symposium*, pages 90–97, 2002.

[34] Benjamin S. Blanchard, Dinesh Verma, and Elmer L. Peterson. *Maintainability : a key to effective serviceability and maintenance management*. Wiley, 1995.

[35] J. Z. Sikorska, M. Hodkiewicz, and L. Ma. Prognostic modelling options for remaining useful life estimation by industry. *Mechanical Systems and Signal Processing*, 25(5):1803–1836, 2011.

[36] Toshio Nakagawa. Periodic and sequential preventive maintenance policies. *Journal of Applied Probability*, 23(2):536–542, 1986.

[37] Viliam Makis and Andrew K. S. Jardine. Optimal Replacement In The Proportional Hazards Model. *INFOR: Information Systems and Operational Research*, 30(1):172–183, 1992.

[38] Adrià Salvador Palau, Zhenglin Liang, Daniel Lütgehetmann, and Ajith Kumar Parlikad. Collaborative prognostics in social asset networks. *Future Generation Computer Systems*, 92:987–995, 2019.

[39] Gilbert Haddad, Peter A. Sandborn, and Michael G. Pecht. An options approach for decision support of systems with prognostic capabilities. *IEEE Transactions on reliability*, 61(4):872–883, 2012.

[40] Tim Pearce, Mohamed Zaki, Alexandra Brintrup, and Andy Neely. High-Quality Prediction Intervals for Deep Learning: A Distribution-Free, Ensembled Approach. *arXiv:1802.07167 [stat.ML]*, 2018.

[41] A. K. Jain, M. Dhada, M. P. Hernandez, M. Herrera, and A. K. Parlikad. Influence of the Imperfect Prognostics on Maintenance Decision. *International Conference on Precision, Meso, Micro and Nano Engineering*, Accepted, 2019.

[42] D. Dyer and R. M. Stewart. Detection of Rolling Element Bearing Damage by Statistical Vibration Analysis. *Journal of Mechanical Design*, 100(2):229–235, 1978.

[43] C. Berenguer, C. Chu, and A. Grall. Inspection and maintenance planning : an application of semi-Markov decision processes. *Journal of Intelligent Manufacturing*, 8(5):467–476, 1997.

[44] Deepam Goyal and B. S. Pabla. Condition based maintenance of machine tools-A review. *CIRP Journal of Manufacturing Science and Technology*, 10:24–35, 2015.

[45] INFSO. Internet of Things in 2020: Roadmap for the Future. *European Comission: Information Society and Media*, 4, 2008.

[46] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[47] Duncan McFarlane. Industrial Internet of Things: Applying IoT in the Industrial Context. EPSRC, 2018.

[48] Alasdair Gilchrist. *Industry 4.0: the industrial internet of things*. Apress, 2016.

[49] Hao Li, Ajith Parlikad, and Adrià Salvador Palau. A Social Network of Collaborating Industrial Assets. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 232(4):389–400, 2018.

[50] Ababacar Ndiaye, Cheikh M.F. Kébé, Pape A. Ndiaye, Abdérafi Charki, Abdessamad Kobi, and Vincent Sambou. A Novel Method for Investigating Photovoltaic Module Degradation. *Energy Procedia*, 36:1222–1231, 2013.

[51] ISO. ISO 17359:2011 - Condition monitoring and diagnostics of machines — General guidelines. Technical report, 2011.

[52] International Organization for Standardization. ISO 13372:2012 - Condition monitoring and diagnostics of machines - Vocabulary. 2012.

[53] A. J. Guillén, A. Crespo, J. F. Gómez, and M. D. Sanz. A framework for effective management of condition based maintenance programs in the context of industrial development of E-Maintenance strategies. *Computers in Industry*, 82, 2016.

[54] Linguo Gong and Kwei Tang. Monitoring machine operations using on-line sensors. *European Journal of Operational Research*, 96(3):479–492, 1997.

[55] D. Taraza, N. Henein, and W. Bryzik. The Frequency Analysis of the Crankshaft's Speed Variation: A Reliable Tool for Diesel Engine Diagnosis. *Journal of Engineering for Gas Turbines and Power*, 123(2):428, 2001.

[56] C Hatch. Improved Wind Turbine Condition Monitoring Using Acceleration Enveloping. *Orbit*, Q2:58–61, 2004.

[57] James Hare, Xiaofang Shi, Shalabh Gupta, and Ali Bazzi. Fault diagnostics in smart micro-grids: A survey. *Renewable and Sustainable Energy Reviews*, 60:1114–1124, 2016.

[58] Ye Zhao, Ling Yang, Brad Lehman, Jean-Francois de Palma, Jerry Mosesian, and Robert Lyons. Decision tree-based fault detection and classification in solar photovoltaic arrays. *2012 Twenty-Seventh Annual IEEE Applied Power Electronics Conference and Exposition (APEC)*, (December):93–99, 2012.

[59] Qian Huang, Dongxiang Jiang, Liangyou Hong, and Yongshan Ding. Application of wavelet neural networks on vibration fault diagnosis for wind turbine gearbox. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5264 LNCS, pages 313–320, 2008.

[60] Jihong Yan and Jay Lee. Degradation Assessment and Fault Modes Classification Using Logistic Regression. *Journal of Manufacturing Science and Engineering*, 127(4):912–914, 2004.

[61] Peter Caselitz and Jochen Giebhardt. Rotor Condition Monitoring for Improved Operational Safety of Offshore Wind Energy Converters. *Journal of Solar Energy Engineering*, 127(2):253, 2005.

[62] Frank Kimmich, Anselm Schwarte, and Rolf Isermann. Fault detection for modern Diesel engines using signal- and process model-based methods. *Control Engineering Practice*, 13(2):189–203, 2005.

[63] Mari Cruz Garcia, Miguel A. Sanz-Bobi, and Javier del Pico. SIMAP: Intelligent System for Predictive Maintenance. Application to the health condition monitoring of a windturbine gearbox. *Computers in Industry*, 57(6):552–568, 2006.

[64] Fang Qian and Gang Niu. Remaining useful life prediction using ranking mutual information based monotonic health indicator. *2015 Prognostics and System Health Management Conference (PHM)*, pages 1–5, 2015.

[65] Steven M. Wood and Douglas L. Goodman. Prognostics in High Reliability Telecom Applications. pages 1–3, 2006.

[66] Hatem M. Elattar, Hamdy K. Elminir, and A. M. Riad. Prognostics: a literature review. *Complex & Intelligent Systems*, 2(2):125–154, 2016.

[67] Man Shan Kan, Andy C. C. Tan, and Joseph Mathew. A review on prognostic techniques for non-stationary and non-linear rotating systems. *Mechanical Systems and Signal Processing*, 62-63:1–20, 2015.

[68] Tangbin Xia, Yifan Dong, Lei Xiao, Shichang Du, Ershun Pan, and Lifeng Xi. Recent advances in prognostics and health management for advanced manufacturing paradigms. *Reliability Engineering & System Safety*, 178:255–268, 2018.

[69] A. Muszynska. Stability of whirl and whip in rotor/bearing systems. *Journal of Sound and Vibration*, 127(1):49–64, 1988.

[70] Tc. Gupta, K. Gupta, and Dk. Sehgal. Instability and chaos of a flexible rotor ball bearing system: an investigation on the influence of rotating imbalance and bearing clearance. *Journal of Engineering for Gas Turbines and Power*, 133(8):082501, 2011.

[71] Karthik Kappaganthu and C. Nataraj. Nonlinear modeling and analysis of a rolling element bearing with a clearance. *Communications in Nonlinear Science and Numerical Simulation*, 16(10):4134–4145, 2011.

[72] Chao Liu and Dongxiang Jiang. Crack modeling of rotating blades with cracked hexahedral finite element method. *Mechanical Systems and Signal Processing*, 46(2):406–423, 2014.

[73] Amit Mathur. Data mining of aviation data for advancing health management. In *Component and Systems Diagnostics, Prognostics, and Health Management II*, volume 4733, pages 61–72. International Society for Optics and Photonics, 2002.

[74] Kai Goebel and Neil Eklund. Prognostic Fusion for Uncertainty Reduction. In *AIAA Infotech@ Aerospace 2007 Conference and Exhibit*, number May, page 2843, 2007.

[75] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, New York, NY, USA, 2012.

[76] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning. *Elements*, 1:337–387, 2009.

[77] S. Hong, Z. Zhou, C. Lu, B. Wang, and T. Zhao. Bearing remaining life prediction using gaussian process regression with composite kernel functions. *Journal of Vibroengineering*, 17(2), 2015.

[78] Gishan Don Ranasinghe and Ajith Kumar Parlikad. A review of machine learning-based prognostics solutions and their applications. *Under Review*, 2019.

[79] Harris Drucker·, Chris J. C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapoik. Support Vector Regression Machines. Technical report.

[80] Abhinav Saxena, José Celaya, Bhaskar Saha, Sankalita Saha, and Kai Goebel. Evaluating algorithm performance metrics tailored for prognostics. In *IEEE Aerospace Conference Proceedings*, pages 1–13, 2009.

[81] R. C M Yam, P. W. Tse, L. Li, and P. Tu. Intelligent predictive decision support system for condition-based maintenance. *International Journal of Advanced Manufacturing Technology*, 17(5):383–391, 2001.

[82] Adrià Salvador Palau, Kshitij Bakliwal, Maharshi Harshadbhai Dhada, Tim Pearce, and Ajith Kumar Parlikad. Recurrent Neural Networks for real-time distributed collaborative prognostics. In *2018 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 1–8. IEEE, 2018.

[83] Egil Martinsson. *WTTE-RNN : Weibull Time To Event Recurrent Neural Network*. PhD thesis, Chalmers University Of Technology, 2016.

[84] V. Sugumaran, V. Muralidharan, and K.I. Ramachandran. Feature selection using Decision Tree and classification through Proximal Support Vector Machine for fault diagnostics of roller bearing. *Mechanical Systems and Signal Processing*, 21(2):930–942, 2007.

[85] Oluseun Omotola Aremu, Adrià Salvador Palau, Ajith Kumar Parlikad, David Hyland-Wood, and Peter Ross McAree. Structuring Data for Intelligent Predictive Maintenance in Asset Management. *IFAC-PapersOnLine*, 51(11):514–519, 2018.

[86] J. D. Kalbfleisch and R. L. Prentice. *The statistical analysis of failure time data*, volume 5. 1980.

[87] Per Kragh Andersen and R. D. Gill. Cox's regression model for counting processes: a large sample study. *The Annals of Statistics*, 10(4):1100–1120, 1982.

[88] Felix O. Heimes. Recurrent neural networks for remaining useful life estimation. In *2008 International Conference on Prognostics and Health Management*, pages 1–6. IEEE, 2008.

[89] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer System Science*, 50:132–150, 1995.

[90] Yingya Zhou, Moncef Chioua, and Weidou Ni. Data-driven multi-unit monitoring scheme with hierarchical fault detection and diagnosis. In *2016 24th Mediterranean Conference on Control and Automation (MED)*, pages 13–18, 2016.

[91] Tianyi Wang, Jianbo Yu, David Siegel, and Jay Lee. A similarity-based prognostics approach for remaining useful life estimation of engineered systems. In *2008 International Conference on Prognostics and Health Management, PHM 2008*, pages 1–6. IEEE, 2008.

[92] E. E. McArthur, S. D. J. Mangina, and J. R. McDonald. COMMAS (COndition Monitoring Multi-Agent System). *Autonomous Agents and Multi-Agent Systems*, 4:279–282, 2001.

[93] Jaouher Ben Ali, Brigitte Chebel-Morello, Lotfi Saidi, Simon Malinowski, and Farhat Fnaiech. Accurate bearing remaining useful life prediction based on Weibull distribution and artificial neural network. *Mechanical Systems and Signal Processing*, 56-57:150–172, 2015.

[94] X. Li, B. S. Lim, J. H. Zhou, S. Huang, S. J. Phua, K. C. Shaw, and M. J. Er. Fuzzy Neural Network Modelling for Tool Wear Estimation in Dry Milling Operation. Technical report, 2009.

[95] Jacques Ferber and Gerhard Weiss. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999.

[96] Michael Wooldridge and Nicholas R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[97] Yves Sallez, Thierry Berger, Silviu Raileanu, Sondes Chaabane, and Damien Trentesaux. Semi-heterarchical control of FMS: From theory to application. *Engineering Applications of Artificial Intelligence*, 23(8):1314–1326, 2010.

[98] Paulo Leitão. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 22(7):979–991, 2009.

[99] Damien Trentesaux. Distributed control of production systems. *Engineering Applications of Artificial Intelligence*, 22(7):971–978, 2009.

[100] Paulo Leitão and Francisco Restivo. ADACOR: A holonic architecture for agile and adaptive manufacturing control. *Computers in Industry*, 57(2):121–130, 2006.

[101] José Barbosa, Paulo Leitão, Emmanuel Adam, and Damien Trentesaux. Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution. *Computers in Industry*, 66:99–111, 2015.

[102] Stefan Poslad. Specifying Protocols for Multi-agent Systems Interaction. *ACM Transactions on Autonomous and Adaptive Systems*, 2(4), 2007.

[103] Stefan Poslad, Patricia Charlton, and Monique Calisti. Specifying Standard Security Mechanisms in Multi-agent Systems. In *Proceedings of the 2002 International Conference on Trust, Reputation, and Security: Theories and Practice*, AAMAS'02, pages 163–176, Berlin, Heidelberg, 2003. Springer-Verlag.

[104] Benjamin M. Gyori. PyKQML: an implementation of KQML messaging in Python, 2018.

[105] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an agent communication language. In *Proceedings CIKM '94*, pages 456–463, 1994.

[106] Li Liu, Kevin P. Logan, David A. Cartes, and Sanjeev K. Srivastava. Fault detection, diagnostics, and prognostics: software agent solutions. *IEEE Transactions on Vehicular Technology*, 56(4):1613–1622, 2007.

[107] Gustavo Gonzalez, Cecilio Angulo, and Cristobal Raya. A Multi-Agent-Based Management Approach for Self-Health Awareness in Autonomous Systems. In *Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe'07)*, pages 79–88. IEEE, 2007.

[108] M. Lebold, K. Reichard, and D. Boylan. Utilizing dcom in an open system architecture framework for machinery monitoring and diagnostics. In *2003 IEEE Aerospace Conference Proceedings (Cat. No.03TH8652)*, volume 3, pages 1227–1236. IEEE.

[109] Michael Thurston and Mitchell Lebold. Standards Developments for Condition-Based Maintenance Systems. *Defense Technical Information Center Compilation Part Notice - ADP013508*, 2000.

[110] Ivan S. Cole, Penny A. Corrigan, Wayne Ganther, Tim Ho, Chris J. Lewis, Tim H. Muster, David Paterson, Don C. Price, D. A. Scott, David Followell, Steve Galea, and Bruce Hinton. Development of a sensor-based learning approach to prognostics in intelligent vehicle health monitoring. In *2008 International Conference on Prognostics and Health Management, PHM 2008*, pages 1–7. IEEE, 2008.

[111] X. Desforges, M. Diévart, P. Charbonnaud, and B. Archimède. A distributed Architecture to implement a Prognostic Function for Complex Systems. *Proceedings of the Annual Conference of the Prognostics and Health Management Society*, pages 2–9, 2012.

[112] Wei Wu, Min Liu, Qing Liu, and Weiming Shen. A quantum multi-agent based neural network model for failure prediction. *Journal of Systems Science and Systems Engineering*, 25(2):210–228, 2016.

[113] Amy J. C. Trappey, Charles V. Trappey, and Wei-Chun Ni. A multi-agent collaborative maintenance platform applying game theory negotiation strategies. *Journal of Intelligent Manufacturing*, 24(3):613–623, 2013.

[114] L. Fasanotti. A distributed intelligent maintenance system based on artificial immune approach and multi-agent systems. *Industrial Informatics (INDIN)*, pages 783–786, 2014.

[115] Luca Fasanotti, Sergio Cavalieri, Emanuele Dovere, Paolo Gaiardelli, and Carlos E. Pereira. An artificial immune intelligent maintenance system for distributed industrial environments. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 232(4):401–414, 2018.

[116] Dejun Ning, Junli Huang, Jian Shen, and Dongjie Di. A cloud based framework of prognostics and health management for manufacturing industry. In *2016 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 1–5. IEEE, 2016.

[117] Bencheikh Ghita, Letouzey Agnes, and Desforges Xavier. Scheduling of production and maintenance activities using multi-agent systems. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 508–515. IEEE, 2018.

[118] Jie Liu and Enrico Zio. Prognostics of a multistack PEMFC system with multiagent modeling. *Energy Science & Engineering*, 7(1):76–87, 2019.

[119] Chance Elliott, Vipin Vijayakumar, Wesley Zink, and Richard Hansen. National instruments LabVIEW: a programming environment for laboratory automation and measurement. *JALA: Journal of the Association for Laboratory Automation*, 12(1):17–24, 2007.

[120] Jihong Yan, Jay Lee, and Yi-Cheng Pan. Introduction of watchdog prognostics agent and its application to elevator hoistway performance assessment. *Journal of the Chinese Institute of Industrial Engineers*, 22(1):56–63, 2005.

[121] Edzel R. Lapira, Amit Deshpande, Jay Lee, and John Snyder. Smart Machine Health and Maintenance: Tool Assembly Prognostics. In *ASME 2008 International Manufacturing Science and Engineering Conference, Volume 2*, pages 75–83. ASME, 2008.

[122] Zhe Shi, Jay Lee, and Peng Cui. Prognostics and health management solution development in LabVIEW: Watchdog agent® toolkit and case study. In *2016 Prognostics and System Health Management Conference (PHM-Chengdu)*, pages 1–6. IEEE, 2016.

[123] G. D. Hadden, P. Bergstrom, B. H. Bennett, G. Vachtsevanos, and J. Van Dyke. Distributed multi-algorithm diagnostics and prognostics for US Navy ships. *In Proceedings 2002 AAAI Spring Symposium*, 2002.

[124] J. Wang, L. Zhang, L. Duan, and R. X. Gao. A new paradigm of cloud-based predictive maintenance for intelligent manufacturing. *Journal of Intelligent Manufacturing*, 25(5):1125–1137, 2017.

[125] Junhong Zhou, Xiang Li, Anton J. R. Andernroomer, Hao Zeng, Kiah Mok Goh, Yoke San Wong, and Geok Soon Hong. Intelligent prediction monitoring system for predictive maintenance in manufacturing. In *IECON Proceedings (Industrial Electronics Conference)*, pages 2314–2319, 2005.

[126] Chaochao Chen, Douglas Brown, Chris Sconyers, Bin Zhang, George Vachtsevanos, and Marcos E. Orchard. An integrated architecture for fault diagnosis and failure prognosis of complex engineering systems. *Expert Systems with Applications*, 39(10):9031–9040, 2012.

[127] Michael J. Roemer, Carl S. Byington, Gregory J. Kacprzynski, and George Vachtsevanos. An overview of selected prognostic technologies with application to engine health management. In *ASME turbo expo 2006: power for land, sea, and air*, pages 707–715. American Society of Mechanical Engineers Digital Collection, 2006.

[128] S. Saha, B. Saha, and K. Goebel. Distributed prognostics using wireless embedded devices. In *PHM 2008. International Conference on Prognostics and Health Management*, pages 1–7, 2008.

[129] Yi Lu Murphey, Jacob A. Crossman, ZhiHang Chen, and John Cardillo. Automotive fault diagnosis-part II: a distributed agent diagnostic system. *IEEE transactions on vehicular technology*, 52(4):1076–1098, 2003.

[130] Tadeusz J. Sobczyk, Tomasz Wegiel, Maciej Sulowicz, Adam Warzecha, and Konrad Weinreb. A distributed system for diagnostics of induction motors. In *2005 5th IEEE International Symposium on Diagnostics for Electric Machines, Power Electronics and Drives*, pages 1–5. IEEE, 2005.

[131] Jan Neuzil, Ondrej Kreibich, and Radislav Smid. A distributed fault detection system based on IWSN for machine condition monitoring. *IEEE Transactions on Industrial Informatics*, 10(2):1118–1123, 2013.

[132] Matthew Daigle, Anibal Bregon, and Indranil Roychoudhury. A distributed approach to system-level prognostics. Technical report, 2012.

[133] Tim Pearce, Mohamed Zaki, Alexandra Brintrup, Nicolas Anastassacos, and Andy Neely. Uncertainty in Neural Networks: Bayesian Ensembling. *arXiv preprint arXiv:1810.05546*, 2018.

[134] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *Advances in Neural Information Processing Systems*, pages 6402–6413, dec 2016.

[135] D. A. Nix and A. S. Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, pages 55–60 vol.1. IEEE, 1994.

[136] Richard J. Rossi. *Mathematical Statistics: An Introduction to Likelihood Based Inference*. John Wiley & Sons, 2018.

[137] R. C. Geary. The Distribution of "Student's" Ratio for Non-Normal Samples. *Supplement to the Journal of the Royal Statistical Society*, 3(2):178, 1936.

[138] Jürgen Schmidhuber. Deep Learning in Neural Networks: An overview. *Neural Networks*, 61:85–117, 2014.

[139] Xin Yan and Xiaogang Su. *Linear regression analysis: theory and computing*. World Scientific, 2009.

[140] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[141] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[142] J. G. Makin. Backpropagation. *Cornell Computer Science Courses*, 2006.

[143] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[144] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A Search Space Odyssey. 2015.

[145] Justin Simon Bayer. *Learning Sequence Representations*. PhD thesis, Technische Universität München, 2015.

[146] Jeff Mell, Jonathan Millar, and Adam Kelleher. Increased corporate concentration and the influence of market power. Technical report, Barclay Impact Series, 2019.

[147] Yu Liu, Yang Yang, Xiaopeng Lv, and Lifeng Wang. A Self-Learning Sensor Fault Detection Framework for Industry Monitoring IoT. *Mathematical Problems in Engineering*, 2013:1–8, 2013.

[148] Nicole El Zoghby, Valeria Loscri, and Enrico Natalizio. *Chapter 8: Robot Cooperation and Swarm Intelligence*. 2014.

[149] Adrià Salvador Palau, Maharshi Dhada, and Ajith Parlikad. Multi-Agent System architectures for collaborative prognostics. *Journal of Intelligent Manufacturing*, 2019.

[150] U. Wilensky. NetLogo. http://ccl.northwestern.edu/netlogo/. *Center for Connected Learning and Computer Based Modeling Northwestern University*, 2009, 1999.

[151] Jihong Yan, Muammer Koç, and Jay Lee. A prognostic algorithm for machine performance assessment and its application. *Production Planning & Control*, 15(8):796–801, 2004.

[152] Bach Phi Duong, Sheraz Ali Khan, Dongkoo Shon, Kichang Im, Jeongho Park, Dong-Sun Lim, Byungtae Jang, and Jong-Myon Kim. A Reliable Health Indicator for Fault Prognosis of Bearings. *Sensors (Basel, Switzerland)*, 18(11):3740, 2018.

[153] A. Thieullen, M. Ouladsine, and J. Pinaton. A Survey of Health Indicators and Data-Driven Prognosis in Semiconductor Manufacturing Process. *IFAC Proceedings Volumes*, 45(20):19–24, 2012.

[154] K. Medjaher, N. Zerhouni, and J. Baklouti. Data-driven prognostics based on health indicator construction: Application to PRONOSTIA's data. In *2013 European Control Conference (ECC)*, pages 1451–1456, 2013.

[155] D. Randall Wilson and Tony R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[156] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[157] Andrew K. S. Jardine and Albert H. C. Tsang. *Maintenance, replacement, and reliability: theory and applications*. CRC press, 2005.

[158] Matthew B. Biggs and Jason A. Papin. Novel Multiscale Modeling Tool Applied to Pseudomonas aeruginosa Biofilm Formation. *PLoS ONE*, 8(10), 2013.

[159] Martín Abadi, Google Brain, and Et al. TensorFlow: A System for Large-Scale Machine Learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, 2016.

[160] François Chollet. Keras: Deep Learning library for Theano and TensorFlow. *GitHub Repository*, pages 1–21, 2015.

[161] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide*. 1998.

[162] I. Fette and A. Melnikov. RFC 6455 - The WebSocket Protocol. *Internet Engineering Task Force*, 2011.

[163] Several Contributors. asyncio: https://docs.python.org/3/library/asyncio.html. 2012.

[164] Several Contributors. Threading: https://docs.python.org/3/library/threading.html.

[165] Y. Labrou and T. Finin. Agent communication languages: the current landscape. *IEEE Intelligent Systems*, 14(2):45–52, 1999.

[166] Michael Steinbach, Levent Ertöz, and Vipin Kumar. The Challenges of Clustering High Dimensional Data. In *New Directions in Statistical Physics*, pages 273–309. 2004.

[167] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017.

[168] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In Jan den Bussche and Victor Vianu, editors, *Database Theory ICDT 2001*, pages 420–434, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[169] A. Saxena and K. Goebel. Turbofan engine degradation simulation data set. *NASA Ames Prognostics Data Repository*, 2008.

[170] Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 International Conference on Prognostics and Health Management, PHM*, 2008.

[171] Jonathan DeCastro, Jonathan Litt, and Dean Frederick. A modular aero-propulsion system simulation of a large commercial aircraft engine. In *44th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, page 4579, 2008.

[172] Kevin Ni, Mani Srivastava, Nithya Ramanathan, Mohamed Nabil Hajj Chehade, Laura Balzano, Sheela Nair, Sadaf Zahedi, Eddie Kohler, Greg Pottie, and Mark Hansen. Sensor network data fault types. *ACM Transactions on Sensor Networks*, 5(3):1–29, 2009.

[173] Edward Balaban, Abhinav Saxena, Prasun Bansal, Kai F. Goebel, and Simon Curran. Modeling, detection, and disambiguation of sensor faults for aerospace applications. *IEEE Sensors Journal*, 9(12):1907–1917, 2009.

[174] Abishek B. Sharma, Leana Golubchik, and Ramesh Govindan. Sensor Faults : Detection Methods and Prevalence in Real-World Datasets. *ACM Transactions on Sensor Networks*, 6(3):23, 2010.

[175] Jastian Ganaputra and Bens Pardamean. Asynchronous publish/subscribe architecture over WebSocket for building real-time web applications. *Internetworking Indonesia*, 7(2):15–19, 2015.

[176] Gábor Imre and Gergely Mezei. Introduction to a WebSocket benchmarking infrastructure. In *2016 Zooming innovation in consumer electronics international conference (ZINC)*, pages 84–87. IEEE, 2016.

[177] Siemens. Annual Report, 2018.

[178] Siemens AG. We power the world with innovative gas turbines, Siemens Gas Turbines Brochure. *siemens.com/gasturbines*, 2016.

[179] Y. Sanjay, Onkar Singh, and B. N. Prasad. Energy and exergy analysis of steam cooled reheat gas–steam combined cycle. *Applied Thermal Engineering*, 27(17-18):2779–2790, 2007.

[180] Siemens. cRSP IT security concept. *siemens.com/bt/services*, 2018.

[181] James Porter. Implementing a Predictive Maintenance System in Siemens Remote Diagnostics. *Long Project, IFM Manufacturing Engineering Tripos*.

[182] Jeremy Bulow. An Economic Theory of Planned Obsolescence. *The Quarterly Journal of Economics*, 101(4):729–750, 1986.

[183] Jianhua Lin. Divergence Measures Based on the Shannon Entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[184] Lei Chen. Curse of Dimensionality. In *Encyclopedia of Database Systems*. 2014.

[185] Mark Schmidt, Glenn Fung, and Romer Rosales. Optimization methods for l1-regularization. *University of British Columbia, Technical Report TR-2009*, 19, 2009.

[186] Jonathon Shlens. A Tutorial on Principal Component Analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[187] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[188] Emmanuel Ramasso and Abhinav Saxena. Performance Benchmarking and Analysis of Prognostic Methods for CMAPSS Datasets. *International Journal of Prognostics and Health Management*, (ISSN2153-2648):1–15, 2014.

[189] He Li, Kaoru Ota, and Mianxiong Dong. Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing. *IEEE Network*, 32(1):96–101, 2018.

[190] Anand Avati, Tony Duan, Sharon Zhou, Kenneth Jung, Nigam H. Shah, and Andrew Ng. Countdown Regression: Sharp and Calibrated Survival Predictions. *arXiv preprint arXiv:1806.08324*, 2018.

[191] Denis Cousineau. Fitting the three-parameter weibull distribution: review and evaluation of existing and new methods. *IEEE Transactions on Dielectrics and Electrical Insulation*, 16(1):281–288, 2009.

[192] G. M. Foody, M. B. McCulloch, and W. B. Yates. The effect of training set size and composition on artificial neural network classification. *International Journal of Remote Sensing*, 16(9):1707–1723, 1995.

[193] Kshitij Bakliwal, Adrià Salvador Palau, and M. H. Dhada. A Multi Agent System architecture to implement Collaborative Learning for social industrial assets. *IFAC-PapersOnLine*, 51(11):1237–1242, 2018.

[194] Eric W. Weisstein. Least Squares Fitting-Exponential. *From MathWorld–A Wolfram Web Resource*, 1999.

[195] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. 2008.

Appendix A

Numerical and analytical results

This chapter presents the numerical and analytical results referenced along this thesis. If any results are shown without uncertainty it is because the variance of their components is significantly smaller than the parameters in question.

A.1 Introductory example formula

In the introduction section, it is claimed that the uncertainty associated to the point at which a linear least squares fit crosses the x-axis decreases in first order as $1/n$ where n are the number of points in the fit. For the example of interest, ($x = (\Delta t, 2\Delta t, ...n\Delta t)$, $y > 0$) this is relatively easy to see. Assuming Gaussian errors in the y axis, given $\bar{T}_{self} = -a/b$ where $y = a + bx$ is the fitted linear equation:

$$\sigma^2_{\bar{T}_{self}} = \left(\frac{1}{b}\right)^2 \sigma_a^2 + \left(\frac{a}{b^2}\right)^2 \sigma_b^2. \tag{A.1}$$

We can obtain the standard errors σ_a and σ_b from [194]:

$$\sigma_a^2 = \sigma^2 \left(\frac{1}{n} + \frac{\bar{x}^2}{ss_{xx}}\right), \qquad \sigma_b^2 = \frac{\sigma^2}{ss_{xx}}. \tag{A.2}$$

Where σ^2 is an estimator of the variance in the y points of our linear fit. Here $ss_{xx} = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2$. $\bar{x}$ is the mean of the x points. Assume n is large enough so that $\bar{x} \approx \frac{n\Delta t}{2}$. Then:

$$ss_{xx} \approx \sum_{i=1}^{n} x_i^2 - \frac{n^2\Delta t^2}{4} = \Delta t^2 \sum_{i=1}^{n} i^2 - \frac{n^2\Delta t^2}{4} = \Delta t^2 \left(\frac{n^3}{3} + \frac{n^2}{4} + \frac{n}{6}\right). \tag{A.3}$$

Combining this into eq. (A.1) gives:

$$\sigma^2_{\bar{T}_{self}} = \left(\frac{\sigma}{b}\right)^2 \left\{\frac{1}{n} + \frac{1}{ss_{xx}}\left(\bar{x} + \left(\frac{a}{b}\right)^2\right)\right\} = \left(\frac{\sigma}{b}\right)^2 \left\{\frac{1}{n} + \frac{1}{\Delta t^2\left(\frac{n^3}{3} + \frac{n^2}{4} + \frac{n}{6}\right)}\left(\bar{x} + \bar{T}^2_{self}\right)\right\}. \tag{A.4}$$

Only points before failure are considered for the fit. Thus, for large enough n, $n\,\bar{T}^2_{self} \to n^2\Delta t^2$, $\bar{x}^2 + \bar{T}^2_{self} \propto n^2\Delta t^2$. Similarly, $\frac{n^3}{3} + \frac{n^2}{4} + \frac{n}{6} \approx \frac{n^3}{3}$. Thus:

$$\sigma^2_{\bar{T}_{self}} \approx \left(\frac{\sigma}{b}\right)^2 \left\{\frac{1}{n} + \frac{1}{\Delta t^2 \frac{n^3}{3}} n^2\Delta t^2\right\} \propto \left(\frac{\sigma}{b}\right)^2 \frac{1}{n}. \tag{A.5}$$

A.2 Multi-Agent System simulations: results

This section shows the results for the Multi-Agent simulations presented in Chapter 6. This results have been submitted as part of a paper to the “Journal of Intelligent Manufacturing" [149].

The following tables feature quadruplets $(N_{\mathrm{P}}, N_{\mathrm{C}}, N_{\mathrm{Co}}, N_{\mathrm{pro}})$ used to obtain the results presented in the section “Results and Discussion". A table showing the purity of the clustering algorithms for each architecture and standard deviation is also included (see [195] for a description of purity).

	$\sigma = 0.0$	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$
Cent.	[6045, 0, 200000, 598]	[6266, 21, 200000, 626]	[5969, 1056, 200000, 632]	[4423, 4028, 199996, 638]	[2672, 10201, 199986, 653]	[1792, 16321, 199970, 401]
Hier.	[6030, 0, 201099, 651]	[6505, 32, 201099, 660]	[5853, 971, 201099, 704]	[4628, 3698, 201099, 696]	[2923, 9144, 201099, 715]	[1714, 16477, 201099, 483]
Dist.	[6000, 0, 12618174, 349]	[6249, 27, 13498609, 360]	[5363, 1305, 16589558, 343]	[3497, 4988, 17319831, 335]	[2233, 10595, 17421179, 335]	[1379, 16462, 17959496, 317]
Hete.	[5962, 0, 12687189, 4111]	[6232, 31, 12754968, 5110]	[5428, 1320, 12703401, 5980]	[4041, 4215, 13107021, 6471]	[2358, 9746, 12952249, 6812]	[1414, 15560, 13260666, 5414]

Table A.1 Results for no agent failure, featuring quadruplets [N_{P},N_{C},N_{Co},N_{pro}]. These values have been rounded to the closest integer from the average of eight experiments.

	$\sigma = 0.0$	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$
Cent.	[3399, 1657, 92718, 424]	[3143, 2070, 86102, 400]	[2585, 3013, 88106, 408]	[1750, 5730, 86550, 421]	[1167, 10172, 89391, 432]	[769, 16204, 84349, 350]
Hier.	[3486, 1674, 77877, 330]	[3625, 1721, 81468, 371]	[2175, 2861, 60803, 304]	[1619, 5045, 68676, 321]	[1053, 11082, 74663, 379]	[711, 16886, 74001, 276]
Dist.	[4537, 918, 7366053, 228]	[4484, 1108, 6989799, 223]	[3526, 2405, 6646838, 220]	[2342, 5282, 7419548, 215]	[1488, 10189, 8322924, 215]	[919, 17021, 8293278, 167]
Hete.	[4504, 924, 5988868, 2587]	[4482, 1080, 5786474, 3028]	[3738, 2179, 5918000, 3501]	[2334, 5431, 5797278, 4099]	[1452, 10220, 5851001, 4300]	[885, 16856, 5984762, 3553]

Table A.2 Results for agent failure, featuring quadruplets [N_{P},N_{C},N_{Co},N_{pro}]. These values have been rounded to the closest integer from the average of eight experiments.

	$\sigma = 0.0$	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$
Centralised	1.0	0.996	0.972	0.954	0.961	0.922
Hierarchical	1.0	0.995	0.982	0.946	0.909	0.946
Distributed	1.0	0.949	0.842	0.749	0.719	0.734
Heterarchical	1.0	1.000	0.982	0.917	0.948	0.928

Table A.3 Table showing clustering purity results at t=400 for the case of no agent failure. These values have been averaged over eight experiments.

	$\sigma = 0.0$	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$
Centralised	0.995	0.998	0.999	0.994	0.974	0.934
Hierarchical	0.784	0.801	0.689	0.666	0.748	0.782
Distributed	0.660	0.679	0.655	0.679	0.615	0.608
Heterarchical	0.763	0.769	0.775	0.734	0.740	0.758

Table A.4 Table showing clustering purity results at t=400 for the case of agent failure. These values have been averaged over eight experiments.

A.3 Weibull analysis

This section presents the Weibull analysis performed to the events of interest in the industrial dataset employed in Chapter 7. The fit was performed by means of the `stats.exponweib.fit` function with initial parameters $\alpha = 0$ and $\beta = 1$. Events occurring within half an hour from each other were deleted because they were assumed to correspond to the same physical event.

A.4 Jensen-Shannnon divergence results

This section features the complete results of the Jensen-Shannon divergence calculations described in Chapter 7. The results are here shown in one table in descending order of values (recall that higher values correspond to frequency distributions that are very different close to events than far away from events, thus indicating informative sensors).

	event	scale	shape	number
0	INTERDUCT THERMOCOUPLE FAULT	18.21	0.44	44
1	INTERDUCT THERMOCOUPLE FAULT - DEVIATION LOW	15.23	0.44	51
2	INTERDUCT TEMPERATURE DEVIATION	22.64	0.48	52
3	DC LUB OIL PUMP CONTACT FAILED TO CLOSE- DCC5	28.82	0.50	118
4	GAS FUEL COMPRESSOR FAULT	22.67	0.57	10
5	WASTE HEAT BOILER COMMON FAULT ...	7.79	0.58	56
6	LIQUID FUEL SYSTEM FAULT - CHANGEOVER INHIBITED	8.36	0.59	160
7	COMPRESSOR EXIT THERMOCOUPLE DEVIATION	15.34	0.59	105
8	TURBINE SHAFT VIBRATION HIGH	17.11	0.60	176
9	TURBINE SHAFT VIBRATION HIGH -UD10X or UD11X	16.91	0.61	184
10	GAS FUEL COMPRESSOR FAULT ...	24.01	0.61	16
11	PT EXIT TEMPERATURE DEVIATION	62.20	0.62	26
12	ELECTRONIC CONTROL UNIT DATALINK FAULT	7.95	0.62	393
13	GAS GENERATOR SHAFT VIBRATION HIGH	14.05	0.64	184
14	STARTER MOTOR THERMISTOR FAULT - TD6	25.26	0.65	41
15	WASTE HEAT BOILER COMMON FAULT ...	5.94	0.67	339
16	Running Trip (External Cause)	33.74	0.67	712
17	Running Trip	36.63	0.68	368
18	GAS FUEL COMPRESSOR FAULT SHUTDOWN ...	17.43	0.68	75
19	FAULT ON FIRE AND GAS SYSTEM - YM3	9.07	0.72	657
20	WASTE HEAT BOILER COMMON FAULT ...	8.89	0.72	163
21	GENERATOR COOLING FAULT - GCP131	6.28	0.73	277
22	WASTE HEAT BOILER COMMON FAULT ...	5.71	0.74	336
23	STARTER MOTOR THERMISTOR FAULT - TD5	23.97	0.74	26
24	Gas Fuel Compressor Fault	12.59	0.75	178
25	Turbine At Maximum Limiting Temperature	2.97	1.04	96

Table A.5 Table showing the values of α and β for the Weibull fits in each event in the dataset together with the number of times that each event has been recorded the scale parameter is in days.

	Elec. Cont. Uni. Data. Fault	Fault Fir. & Gas Syst	Gas Gen. Shaft Vib. High	Gen. Cool. Fault	L. Fuel Sy. Fault - Cng. Inh.	Start. Mot. Therm. Fault	T. At Max. Lim. Temp.	W. Heat Boil. Comm. Fault	Running Trip
0	PT2: 0.41	TC7: 0.61	T.Op: 0.86	TC10: 0.3	BND3: 0.69	BND1: 0.47	BND2: 0.95	T.Op: 0.73	T.Op: 0.74
1	TC20: 0.4	TC8: 0.6	BND1: 0.56	PT6: 0.3	BND2: 0.69	TC21: 0.43	BND1: 0.95	TC4: 0.48	TC255: 0.67
2	BND1: 0.39	TC23: 0.55	PT2: 0.51	BND3: 0.29	BND1: 0.68	TC2: 0.41	BND3: 0.95	TC1: 0.48	TC256: 0.67
3	BND3: 0.39	TC22: 0.55	PT6: 0.49	T.Op: 0.29	PT6: 0.67	TC4: 0.41	TC12: 0.92	TC2: 0.47	TC260: 0.62
4	PT220: 0.33	TC6: 0.53	PT220: 0.47	PT2: 0.28	TC10: 0.65	TC23: 0.41	TC11: 0.91	TC12: 0.46	TC6: 0.61
5	PT6: 0.32	TC61: 0.53	PT7: 0.43	TC1: 0.27	PT2: 0.59	TC22: 0.41	TC1: 0.83	TC5: 0.46	TC259: 0.59
6	BND2: 0.31	TC21: 0.53	FFDEM: 0.38	PT8: 0.27	PT220: 0.49	FFDEM: 0.41	TC2: 0.82	TC11: 0.46	TC3: 0.57
7	TC12: 0.3	TC24: 0.52	FFDEMGAS: 0.37	TC12: 0.26	TC11: 0.48	TC5: 0.4	TC23: 0.74	TC3: 0.46	PT7: 0.51
8	TC19: 0.3	TC4: 0.5	TC1: 0.37	TC2: 0.26	FFDEMGAS: 0.48	T.Op: 0.4	GCP25: 0.59	TC22: 0.43	PT6: 0.5
9	TC10: 0.3	TC20: 0.49	PT3: 0.36	PT3: 0.26	TC19: 0.46	BND3: 0.4	PT6: 0.59	TC24: 0.43	PT2: 0.5
10	TC11: 0.28	TC5: 0.47	PT8: 0.35	TC11: 0.26	TC12: 0.45	TC24: 0.39	TC21: 0.53	PT220: 0.42	TC12: 0.49
11	PT182: 0.27	TC3: 0.43	BND3: 0.35	PT220: 0.26	PT182: 0.43	TC1: 0.38	TC20: 0.5	TC23: 0.41	TC5: 0.49
12	TC2: 0.25	BND1: 0.4	PDT1: 0.35	BND2: 0.25	TC1: 0.43	TC3: 0.37	TC10: 0.49	TC21: 0.41	TC258: 0.49
13	PT181: 0.24	TC2: 0.4	PT181: 0.34	BND1: 0.23	TC2: 0.37	TC12: 0.36	PT8: 0.48	PT2: 0.41	TC257: 0.48
14	TC1: 0.24	TC1: 0.4	PT182: 0.34	GCP25: 0.2	PT181: 0.37	BND2: 0.36	TC19: 0.46	TC10: 0.41	PT220: 0.47
15	PT8: 0.24	TC11: 0.38	TC10: 0.32	PT182: 0.2	GCP25: 0.24	TC11: 0.36	TC22: 0.33	PDT1: 0.39	TC11: 0.47
16	FFDEMGAS: 0.23	TC12: 0.38	GCP25: 0.3	PT181: 0.2	PT8: 0.19	TC10: 0.35	FFDEMGAS: 0.24	GCP25: 0.33	TC21: 0.47
17	GCP25: 0.22	BND3: 0.37	BND2: 0.26	TC19: 0.19	FFDEM: 0.15	GCP25: 0.34	GGSPD: 0.23	TC19: 0.33	TC2: 0.46
18	PT3: 0.21	TC10: 0.37	GGSPD: 0.15	FFDEMGAS: 0.18	PTSPD: 0.1	PT3: 0.33	T-Fire: 0.16	BND1: 0.32	TC1: 0.45
19	T.Op: 0.13	FFDEM: 0.34	PTSPD: 0.13	GGSPD: 0.12	GGSPD: 0.08	PT6: 0.33	FFDEM: 0.15	TC20: 0.32	TC20: 0.45
20	GGSPD: 0.11	BND2: 0.33	NaN	NaN	NaN	TC20: 0.32	NaN	PT182: 0.31	PT8: 0.44
21	NaN	PT3: 0.33	NaN	NaN	NaN	PT2: 0.32	NaN	PT181: 0.31	TC19: 0.43
22	NaN	PT220: 0.32	NaN	NaN	NaN	FFDEMGAS: 0.3	NaN	BND2: 0.29	TC10: 0.43
23	NaN	PT8: 0.32	NaN	NaN	NaN	PT220: 0.28	NaN	PT8: 0.28	BND3: 0.4
24	NaN	PT6: 0.31	NaN	NaN	NaN	PT8: 0.27	NaN	PT3: 0.28	TC4: 0.4
25	NaN	PT2: 0.3	NaN	NaN	NaN	TC19: 0.26	NaN	PT6: 0.28	PT3: 0.39
26	NaN	TC19: 0.3	NaN	NaN	NaN	PT181: 0.25	NaN	FFDEM: 0.22	FFDEMGAS: 0.38
27	NaN	PT181: 0.3	NaN	NaN	NaN	PT182: 0.25	NaN	FFDEMGAS: 0.22	PT181: 0.37
28	NaN	PT182: 0.28	NaN	NaN	NaN	T-Fire: 0.13	NaN	PT7: 0.21	PDT1: 0.37
29	NaN	T.Op: 0.28	NaN	NaN	NaN	GGSPD: 0.12	NaN	BND3: 0.21	PT182: 0.37
30	NaN	GCP25: 0.27	NaN	NaN	NaN	NaN	NaN	GGSPD: 0.16	FFDEM: 0.36
31	NaN	FFDEMGAS: 0.26	NaN	NaN	NaN	NaN	NaN	PTSPD: 0.11	TC22: 0.35
32	NaN	T-Fire: 0.13	NaN	NaN	NaN	NaN	NaN	NaN	BND2: 0.35
33	NaN	GGSPD: 0.12	NaN	NaN	NaN	NaN	NaN	NaN	BND1: 0.34
34	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	GCP25: 0.32
35	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	BND4: 0.27
36	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	GGSPD: 0.2
37	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	PTSPD: 0.15

Table A.6 Table showing the complete values of the Jensen-Shannon divergence for different events and different sensors in SGT-100 gas turbines.

	Comp. Ex. Therm. Dev.	Fault Fir. & Gas Syst	Inter. Temp. Dev.	Inter. Thermocoup. Faul.	Start. Mot. Therm. Fault	T. Shaft Vib. High	W. Heat Boil. Comm. Fault	Running Trip
0	GCP25: 0.74	PDT8: 0.56	PDT8: 0.68	TC2: 0.64	PT12: 0.66	PDT8: 0.68	TC14: 0.43	PT12: 0.64
1	TC24: 0.56	GCP25: 0.51	PT181: 0.62	TC1: 0.62	PT3: 0.63	GCP25: 0.63	TC23: 0.39	PT16: 0.6
2	TC14: 0.55	PT181: 0.46	PT12: 0.59	TC12: 0.6	PT2: 0.63	PT181: 0.57	TC10: 0.38	TC255: 0.57
3	PT12: 0.55	PT2: 0.45	TC2: 0.59	TC15: 0.59	TC13: 0.61	TC2: 0.56	GCP25: 0.37	TC2: 0.55
4	TC10: 0.53	PT16: 0.42	TC1: 0.54	TC23: 0.59	PDT8: 0.6	TC21: 0.56	TC17: 0.36	TC25: 0.53
5	TC16: 0.53	PT12: 0.41	TC23: 0.54	TC22: 0.57	TC15: 0.55	PT2: 0.55	PT2: 0.35	TC11: 0.52
6	TC2: 0.52	TC2: 0.39	TC15: 0.53	PT12: 0.57	TC12: 0.55	PT12: 0.54	TC18: 0.35	TC14: 0.52
7	TC15: 0.49	TC10: 0.39	TC12: 0.52	TC11: 0.56	PT181: 0.54	TC15: 0.53	TC12: 0.33	TC12: 0.51
8	TC11: 0.48	TC14: 0.39	TC22: 0.52	TC10: 0.55	TC10: 0.54	TC14: 0.51	TC19: 0.3	TC1: 0.51
9	TC23: 0.48	TC17: 0.38	PT3: 0.51	TC21: 0.55	TC14: 0.54	TC13: 0.5	TC20: 0.3	TC10: 0.5
10	TC12: 0.47	TC1: 0.38	TC21: 0.51	TC14: 0.55	TC1: 0.54	TC10: 0.5	TC16: 0.3	TC21: 0.49
11	PT2: 0.47	TC11: 0.37	TC10: 0.5	GCP25: 0.53	TC17: 0.54	TC11: 0.49	TC21: 0.28	GCP25: 0.49
12	PT16: 0.47	TC18: 0.37	TC11: 0.49	TC13: 0.52	TC18: 0.53	TC12: 0.48	TC2: 0.27	TC17: 0.49
13	TC1: 0.47	TC12: 0.37	TC13: 0.48	TC16: 0.5	TC2: 0.53	PT3: 0.48	PT3: 0.25	TC24: 0.49
14	PT3: 0.46	PT3: 0.36	TC14: 0.47	PT3: 0.5	PT16: 0.53	TC16: 0.48	TC11: 0.24	TC13: 0.48
15	TC18: 0.46	TC21: 0.36	TC16: 0.46	PT16: 0.46	TC11: 0.53	TC22: 0.47	TC15: 0.23	TC18: 0.46
16	TC17: 0.44	TC16: 0.36	PT16: 0.46	PT2: 0.45	TC16: 0.51	TC1: 0.47	TC1: 0.21	TC16: 0.45
17	TC21: 0.44	TC15: 0.35	GCP25: 0.4	TC19: 0.44	TC20: 0.37	TC26: 0.47	TC13: 0.21	TC15: 0.45
18	TC13: 0.41	TC13: 0.35	PT2: 0.38	TC20: 0.44	TC19: 0.32	TC25: 0.47	TC22: 0.19	FFDEM: 0.45
19	TC26: 0.39	TC20: 0.31	TC17: 0.34	TC18: 0.4	PTSPD: 0.22	TC18: 0.47	PTSPD: 0.07	PT181: 0.45
20	TC27: 0.38	TC19: 0.28	TC20: 0.33	TC17: 0.4	NaN	TC24: 0.45	NaN	BND3: 0.44
21	TC25: 0.3	TC23: 0.27	TC18: 0.33	PTSPD: 0.06	NaN	TC20: 0.44	NaN	TC23: 0.44
22	TC20: 0.29	TC22: 0.25	TC19: 0.32	NaN	NaN	TC17: 0.43	NaN	PT2: 0.41
23	TC19: 0.28	PTSPD: 0.13	PTSPD: 0.1	NaN	NaN	PT16: 0.43	NaN	PT3: 0.38
24	TC22: 0.27	NaN	NaN	NaN	NaN	TC19: 0.41	NaN	TC22: 0.38
25	PTSPD: 0.15	NaN	NaN	NaN	NaN	TC23: 0.36	NaN	TC20: 0.37
26	NaN	NaN	NaN	NaN	NaN	PTSPD: 0.11	NaN	BND4: 0.36
27	NaN	NaN	NaN	NaN	NaN	NaN	NaN	PT220: 0.35
28	NaN	NaN	NaN	NaN	NaN	NaN	NaN	TC19: 0.35
29	NaN	NaN	NaN	NaN	NaN	NaN	NaN	PT182: 0.32
30	NaN	NaN	NaN	NaN	NaN	NaN	NaN	BND1: 0.28
31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	BND2: 0.21
32	NaN	NaN	NaN	NaN	NaN	NaN	NaN	PTSPD: 0.06

Table A.7 Table showing the complete values of the Jensen-Shannon divergence for different events and different sensors in SGT-200 gas turbines.

	Fault Fir. & Gas Syst	Gas Gen. Shaft Vib. High	Start. Mot. Therm. Fault	T. At Max. Lim. Temp.	T. Shaft Vib. High	W. Heat Boil. Comm. Fault	Running Trip
0	BND4: 0.43	PT2: 0.28	T.Op: 0.86	BND4: 0.82	BND4: 0.7	BND4: 0.55	BND4: 0.59
1	PT6: 0.34	PT220: 0.26	PT182: 0.65	BND3: 0.79	BND2: 0.54	BND2: 0.41	BND2: 0.46
2	PT220: 0.34	PDT8: 0.26	PT3: 0.63	PT6: 0.78	PT2: 0.47	PT220: 0.39	PT6: 0.38
3	BND2: 0.31	PDT1: 0.25	TC10: 0.62	BND1: 0.76	PT220: 0.47	PT2: 0.33	PT220: 0.36
4	BND3: 0.28	PT3: 0.24	TC12: 0.6	TC11: 0.63	PT6: 0.46	PT6: 0.31	PDT1: 0.34
5	FFDEMGAS: 0.21	PT8: 0.22	GCP25: 0.58	TC1: 0.6	BND3: 0.42	TC12: 0.28	BND1: 0.34
6	TC11: 0.2	PT181: 0.2	TC20: 0.57	TC12: 0.55	GCP25: 0.34	BND3: 0.26	BND3: 0.33
7	GCP25: 0.2	RTD48: 0.2	TC2: 0.55	PT181: 0.51	PT182: 0.34	BND1: 0.26	PT2: 0.3
8	PT8: 0.2	PT182: 0.2	TC11: 0.54	PT182: 0.49	PT181: 0.33	PT8: 0.26	T-Fire: 0.29
9	BND1: 0.2	BND3: 0.19	TC19: 0.53	TC10: 0.47	PT8: 0.33	PT3: 0.25	PT8: 0.28
10	PT181: 0.19	BND1: 0.19	PT6: 0.53	PT220: 0.43	TC11: 0.32	TC2: 0.25	PT182: 0.27
11	TC19: 0.19	PT6: 0.18	TC1: 0.5	TC19: 0.41	FFDEMGAS: 0.32	TC11: 0.24	GCP25: 0.27
12	PT182: 0.19	RTD1: 0.18	BND3: 0.45	PT3: 0.39	BND1: 0.32	GCP25: 0.24	PT181: 0.27
13	TC1: 0.19	BND2: 0.18	BND4: 0.45	TC2: 0.36	TC1: 0.3	TC1: 0.24	PT3: 0.26
14	TC10: 0.18	FFDEMGAS: 0.18	BND1: 0.45	FFDEM: 0.3	TC10: 0.28	PT181: 0.24	FFDEMGAS: 0.26
15	TC2: 0.17	FFDEM: 0.18	PT220: 0.36	GGSPD: 0.3	TC12: 0.27	PT182: 0.24	TC1: 0.25
16	TC12: 0.16	T-Fire: 0.17	BND2: 0.33	FFDEMGAS: 0.3	TC2: 0.26	PDT1: 0.23	TC2: 0.25
17	PT3: 0.13	GCP25: 0.16	TC21: 0.31	PT2: 0.3	PT3: 0.24	FFDEMGAS: 0.23	TC19: 0.25
18	GGSPD: 0.13	PTSPD: 0.04	TC22: 0.27	GCP25: 0.26	TC19: 0.24	TC19: 0.23	FFDEM: 0.25
19	PT2: 0.12	GGSPD: 0.04	GGSPD: 0.25	PTSPD: 0.2	GGSPD: 0.11	TC10: 0.22	TC11: 0.25
20	NaN	NaN	NaN	NaN	NaN	FFDEM: 0.21	RTD48: 0.24
21	NaN	NaN	NaN	NaN	NaN	T-Fire: 0.21	PDT8: 0.24
22	NaN	NaN	NaN	NaN	NaN	RTD48: 0.18	TC12: 0.24
23	NaN	NaN	NaN	NaN	NaN	PDT8: 0.18	TC10: 0.21
24	NaN	NaN	NaN	NaN	NaN	RTD1: 0.16	RTD1: 0.21
25	NaN	NaN	NaN	NaN	NaN	GGSPD: 0.1	GGSPD: 0.13
26	NaN	NaN	NaN	NaN	NaN	PTSPD: 0.07	PTSPD: 0.09

Table A.8 Table showing the complete values of the Jensen-Shannon divergence for different events and different sensors in SGT-300 gas turbines.

	Gas Fuel Comp. Fault	Gas Gen. Shaft Vib. High	L. Fuel Sy. Fault - Cng. Inh.	W. Heat Boil. Comm. Fault	Running Trip
0	PDT8: 0.38	PDT8: 0.61	BND3: 0.5	BND2: 0.36	MCC4: 0.73
1	PT258: 0.35	BND3: 0.5	BND2: 0.48	BOV2 FBACK: 0.34	PT6B: 0.55
2	BND3: 0.34	PDT1: 0.47	PDT8: 0.45	PT220: 0.34	PDT1: 0.51
3	PT181: 0.33	PDT27: 0.47	FFDEMGAS: 0.35	BND3: 0.24	PDT8: 0.5
4	GCP25: 0.32	PT258: 0.47	FFDEM: 0.35	PDT1: 0.22	PT6A: 0.5
5	PT6A: 0.32	BND1: 0.47	GCP25: 0.31	BND1: 0.2	BOV2 FBACK: 0.49
6	FFDEM: 0.32	PT181: 0.46	PT296: 0.28	FFDEMGAS: 0.18	PT258: 0.46
7	PT296: 0.31	PT2: 0.45	PT258: 0.27	FFDEM: 0.18	BOV1 FBACK: 0.44
8	PT182: 0.31	GGSPD: 0.44	PT181: 0.27	PDT8: 0.18	PDT27: 0.43
9	FFDEMGAS: 0.31	GCP25: 0.44	PT3: 0.26	PDT27: 0.15	PT7: 0.43
10	GGSPD: 0.31	BOV2 FBACK: 0.44	BND1: 0.25	PT181: 0.15	LT1: 0.42
11	BND1: 0.31	PT296: 0.44	GGSPD: 0.25	GCP25: 0.15	RTD1: 0.42
12	PT295: 0.3	PT182: 0.44	PT182: 0.24	PT258: 0.15	PT296: 0.42
13	PDT1: 0.3	PT220: 0.43	PT295: 0.22	BOV1 FBACK: 0.14	PT181: 0.41
14	LT1: 0.29	BND2: 0.43	BND4: 0.22	BND4: 0.13	RTD48: 0.41
15	PT3: 0.27	FFDEMGAS: 0.42	PDT27: 0.22	PT182: 0.13	GCP25: 0.41
16	PT220: 0.27	FFDEM: 0.41	PT2: 0.18	GGSPD: 0.12	PT220: 0.41
17	PT2: 0.27	PT295: 0.37	PT220: 0.17	PT3: 0.11	TC10: 0.4
18	PDT27: 0.27	PT3: 0.37	PDT1: 0.17	PT2: 0.09	PT182: 0.4
19	BOV2 FBACK: 0.25	BND4: 0.36	PTSPD: 0.11	PTSPD: 0.08	TC1: 0.39
20	BND2: 0.25	BOV1 FBACK: 0.33	NaN	NaN	FFDEM: 0.37
21	BOV1 FBACK: 0.24	PTSPD: 0.13	NaN	NaN	FFDEMGAS: 0.37
22	PTSPD: 0.18	NaN	NaN	NaN	BND1: 0.36
23	BND4: 0.17	NaN	NaN	NaN	GGSPD: 0.34
24	NaN	NaN	NaN	NaN	PT8: 0.34
25	NaN	NaN	NaN	NaN	BND4: 0.33
26	NaN	NaN	NaN	NaN	BND3: 0.32
27	NaN	NaN	NaN	NaN	PT2: 0.32
28	NaN	NaN	NaN	NaN	BND2: 0.3
29	NaN	NaN	NaN	NaN	PT3: 0.27
30	NaN	NaN	NaN	NaN	PTSPD: 0.19
31	NaN	NaN	NaN	NaN	PT6: 0.16

Table A.9 Table showing the complete values of the Jensen-Shannon divergence for different events and different sensors in SGT-400 gas turbines.

Appendix B

Code

B.1 Pseudocode for multi-agent simulations

In this experiment each of the agents described in Chapter 3 is implemented in Netlogo, the pseudocode for these agents follows here (extracted from the author's paper in the Journal of Intelligent Manufacturing [149]). Note how in this simulation, Virtual Assets are not really necessary as the data is entirely synthetic. Thus, in practice, the Virtual Assets and the Digital Twins are merged into a single agent.

Algorithm 3 Virtual Asset

if $\mathrm{HI_i} \geq 0$ **then**
    Set $\mathrm{HI_i} = \mathrm{HI_i}(t_{li})$;
    Set $t_{li} = t_{li} + 1$;
    if agent-fault **is** False **then**
        Update agent connections;
        Send HI_i to a higher-level agent;
    end if
end if
if $\mathrm{HI_i} < 0$ **then**
    Set fault True;
    Set $\mathrm{HI_i} = 0$;
    Set $t_{li} = 0$;
end if

Algorithm 4 Digital Twin

Receive HI_i from the Virtual Asset;
Receive data from other Digital Twins;
Fit data using python's least_squares algorithm;
if fault **is** False **then**
    Set t_{fi}^{e} from fit;
    if $t_{li} > \eta t_{fi}^{e}$ **then**
        Preventively maintain;
    end if
end if
if fault **is** True **then**
    Correctively maintain;
end if
if distributed **is** True **then**
    execute distributed clustering algorithm;
end if
Send data to other Digital Twins;
Calculate computation time;

Algorithm 5 Mediator Agent

Receive HI_i from the Virtual Assets;
Fit data using python's least_squares algorithm;
for Assets connected to the agent **do**
    if fault **is** False **then**
        Set t_{fi}^{e} from fit;
        if $t_{li} > \eta t_{fi}^{e}$ **then**
            Preventively maintain;
        end if
    end if
    if fault **is** True **then**
        Correctively maintain;
    end if
end for
Calculate computation time;

Algorithm 6 Social Platform

```
Receive data from the Digital Twins or Mediator Agents (depending on architecture);
if centralised is False then
    compute k-means clustering;
    send data to the pertinent clusters;
end if
if centralised is True then
    for Assets assigned to each cluster do
        Fit data using python's least_squares algorithm;
        if fault is False then
            Set t^e_fi from fit;
            if t_li > η t^e_fi then
                Preventively maintain;
            end if
        end if
        if fault is True then
            Correctively maintain;
        end if
    end for
end if
Compute purity and cost metrics;
Calculate computation time;
```

B.2 Implementation code

The full code corresponding to the experiments presented in Chapters 5 and 7 is not included in this document because of proprietary reasons. However, extracts of it are described here in pseudo-code.

B.2.1 Bespoke bash script

A bespoke bash script was used to run several python processes in parallel. Here, it is described in pseudo-code. Note how in this script the number of turbines is given manually, but in reality any turbine can join (or leave) the fleet of operative gas turbines by connecting to the Social Platform.

Algorithm 7 Bash script for fleet initialisation

```
timestep=T
N_assets=N
epochs_rnn=E
prediction_window=W
Calculate number of websocket ports needed (2·N+2)
idle_traj=I                    ▷ number of trajectories needed to perform prognostics
run getport procedure that provides (2·N+2) unused ports
export all variables
procedure FUNCTION_FOR_ITERATIONS
    local run=$1
    export run
    python3 -c run Virtual Asset(exports) and Digital Twin(exports) &
    '#fg'
end procedure
python3 -c run Social Platform(exports)
echo ${port[@]}
for run in (eval echo "{1..N_assets}") do;
    do FUNCTION_FOR_ITERATIONS "$run" & done &&
    '#fg'
end for
```

B.2.2 Leanness experiment

The aim of this experiment presented in Sec. 5.5.4 is to demonstrate that the proposed framework can adapt to a different industrial scenario with minimal modification to the agents code. In this case, the industrial scenario corresponded to data coming from Siemens Turbomachinery. In order to adapt the system to this data, the following lines of code were changed:

Algorithm 8 Modifications in the VM

```
(... rest of the agent code ...)
procedure LOAD_VA_DATA(data_source)        ▷ Most modifications done here
    Load data in one batch → Parallel data loading / sampling    ▷ Memory constraints
    C-MAPSS segmentation → Event-based segmentation              ▷ Recurrent events
end procedure
```

Note how in a real industrial scenario, the data is loaded into the Virtual Asset in batches due to memory constraints. This modification can also be used for the case of the C-MAPSS data-set, and thus is shared across all industrial scenarios.

B.2.3 Data preparation for Principal Component Analysis

One of the steps of the dimensionality reduction framework presented in this thesis relies on Principal Component Analysis (PCA). PCA decomposition is a method in which a set of linearly correlated features is transformed into a set of linearly uncorrelated variables. This new set of features is defined so that its components are ordered in decreasing value of their captured variance.

In this thesis, PCA is only applied to features in the data-set that are strongly correlated (a Pearson coefficient of 0.8 is chosen as an arbitrary boundary). Only the first two principal components of every group of seven or less strongly correlated sensors are kept for further prognostics.

Algorithm 9 PCA dimensionality reduction

```
1:  procedure PCA_ANALYSIS(df_sensors,min_correlation=0.8)
2:      Delete all sensors with more than 90% missing values
3:      Calculate the Pearson correlation coefficient of all sensor pairs
4:      sensors_unused=set(all_sensors)
5:      while len(sensors_unused)>3 do
6:          S= sensor in sensors_unused with more highly-correlated sensors
7:          sensors_for_pca=higly_corr_sensors(S)[:7]
8:          use sklearn's RobustScaler to scale sensors_for_pca
9:          pca.fit(sensors_for_pca)
10:         pca.transform(sensors_for_pca)
11:         save [scaler, pca_components, pca_decomposition]
12:         sensors_unused=set(sensors_unused)-sensors_for_pca-S
13:     end while
14:     return all triplets of [scaler, pca_components, pca_decomposition]
15: end procedure
```

B.2.4 Class balance in the classification algorithm

For the classification algorithm used in the case study included in this thesis, it was crucial to ensure that trajectories were split as a whole in the training and test data sets. Here included

is a pseudo-code description of the algorithm used to perform the split. The pseudo code also includes the target definition for the classification algorithm.

Algorithm 10 Train/test split for classification

```
1:  procedure PREP_TRAIN_CLASS(df_sensors,tte,days,balanced)
2:      re-normalise trajectory identifiers
3:      targets=tte                                  ▷ tte is the time to event
4:      for k in number_trajectories do;
5:          targets[k][targets[k]<=days]=0
6:          targets[k][targets[k]>days]=1
7:      end for
8:      split=3/4
9:      while condition==False do
10:         for j in number_trajectories do;
11:             randomly assign trajectory j to test (p=1/4) or train (p=3/4)
12:             if balanced then
13:                 check how many targets of each type have been insofar assigned
14:                 append a subset of the trajectory so the targets are balanced
15:             end if
16:         end for
17:         if train/test split ≈ split then
18:             condition=True
19:         end if
20:     end while
21:     return indexes belonging to train and test, targets
22: end procedure
```