2019-12-02

Bayesian Filtering Methods For Dynamic System Monitoring and Control

Erotokritos Skordilis
University of Miami, sge12@miami.edu

UNIVERSITY OF MIAMI

BAYESIAN FILTERING METHODS FOR DYNAMIC SYSTEM MONITORING AND CONTROL

By

Erotokritos Skordilis

A DISSERTATION

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Coral Gables, Florida

December 2019

UNIVERSITY OF MIAMI

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

BAYESIAN FILTERING METHODS FOR DYNAMIC SYSTEM MONITORING AND CONTROL

Erotokritos Skordilis

Approved:

Ramin Moghaddas, Ph.D. Assistant Professor of Industrial Engineering	Murat Erkoc, Ph.D. Associate Professor of Industrial Engineering
Shihab S. Asfour, Ph.D. Professor of Industrial Engineering	Vincent Omachonu, Ph.D. Professor and Chair, Department of Industrial Engineering
Mei-Ling Shyu, Ph.D. Professor of Electrical & Computer Engineering	Guillermo Prado, Ph.D. Dean of the Graduate School

SKORDILIS, EROTOKRITOS (Ph.D., Industrial Engineering)

(December 2019)

Bayesian Filtering Methods for Dynamic System Monitoring and Control

Abstract of a dissertation at the University of Miami.

Dissertation supervised by Professor Ramin Moghaddas.
No. of pages in text. (175)

Real-time system monitoring and control represent two of the most important issues that characterize modern industries in critical areas of civilian and military interest, including power grid, energy, healthcare, aerospace, and infrastructure. During the past decade, there has been a rapid development of robust dynamic system monitoring and control methods for fault diagnosis and failure prognosis. Among various monitoring and control policies, condition-based maintenance (CBM) has been studied by many researchers due to its ability to enable a large amount of monitoring data for real-time diagnostics and prognostics. A considerable amount of literature has been published on the subject, providing a large volume of dynamic system control methods. Previously published studies are limited by assumptions that can generally be distinguished into three main categories: i) predefined system failure thresholds, ii) simplified latent dynamics, and iii) unrealistic parametric forms that describe the evolution of system dynamics through time. This thesis provides an array of solution approaches that overcomes the aforementioned assumptions in a smart and effective way by introducing novel quantitative frameworks for real-time monitoring, control, and decision-making for dynamic systems. The proposed frameworks are categorized into two main phases of a comprehensive framework.

The first phase contains two original Bayesian filtering methods for condition monitoring and control of systems with either linear or non-linear degradation dynamics.

The former is designed only for systems with linear latent and observable dynamics and utilizes Kalman filtering for state-parameter inference. It considers a failure process that is purely stochastic and is based on logistic regression. This process is directly affected by the latent system dynamics, therefore avoiding the need for a priori failure thresholds. The latter takes into consideration multiple levels of system dynamics that evolve either linearly or non-linearly. A hybrid particle filter is developed for state-parameter inference, while an Extreme Learning Machine artificial neural network is utilized to relate sensor observations to latent system dynamics. Both frameworks are tested and validated on synthetic and real-world time-series datasets.

The second phase of this thesis introduces an original method for optimal control and decision-making that employs Bayesian filtering-based deep reinforcement learning with fully stochastic environments. Sets of deep reinforcement learning agents were trained to develop control policies. Bayesian filtering methods from the first phase were utilized to provide environment states that use the estimates from latent system dynamics. This method is used in two different applications for maintenance cost minimization and estimating remaining useful life of a system under condition monitoring. Results obtained from applying the framework on simulated and real-world time-series data suggest that the proposed Bayesian filtering-based deep reinforcement learning algorithm can be trained even with limited data, which can be useful for real-time control and decision making for many dynamic systems.

Dedicated to my fantastic parents, Gordios and Vasiliki, for providing me with every opportunity and encouragement in life

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor, Professor Ramin Moghaddas, for all his help in carrying out the work presented in this thesis. Ever since we started working together four years ago, he has been an incredible source of knowledge, advice, and encouragement and has provided me with countless opportunities. Beyond that, he has also been a great mentor and I will miss our weekly meetings discussing all sorts of interesting topics and developments. I would also like to thank my committee members, Professor Murat Erkoc, Professor Shihab S. Asfour, Professor Vincent Omachonu, and Professor Mei-Ling Shyu, whom I have had the privilege of working with and for the invaluable inputs they provided during writing this thesis. I am also grateful to all the staff in the Department of Industrial Engineering at University of Miami, for providing such a great environment to work in. Moreover, I would like to give my thanks to all members in our Data Analytics Lab: Abdullah Anad Alghuried, Feiran Xu, and Francisco Manuel Romero Veitia for their help along the road. Of course, I must thank those with whom I shared an office space with for many years; Mehrad Bastani, Xu Dong, Aristotelis-Manos Thanos, and Mona Issabahksh. I will always remember my experience cooperating with them and forever cherish our friendship. Furthermore, I would like to express my gratitude to my friend Emily Parks for her diligent proofreading of this thesis and her invaluable assistance in improving the figure quality.

A very special credit goes out to my uncle, Professor Michael Scordilis, and my cousin Leonidas, for their constant support all these years that made my entire Ph.D student experience more constructive. Our weekly discussions over dinner were always entertaining.

I would also like to give special acknowledgements to my friend and mentor, assistant professor G.K.D. Saharidis of the Department of Mechanical Engineering, University of Thessaly. His help and guidance opened to me the academic road that led me here today.

Finally, I would like to thank my parents, Gordios and Vasiliki, who have supported me at every stage of my college studies over the last 15 years. I would not be here if not for them.

Erotokritos Skordilis

University of Miami

December 2019

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background and Motivation

The development of reliable real-time control and decision-making tools for dynamic industrial systems that operate under uncertainty has proven to be one of the most challenging problems in many critical industries. The main reasons behind that are the increasing complexity of industrial equipment and the escalating demand for cost-effective practices to keep it operational. As a result, domain researchers have proposed many control frameworks for determining suitable actions to maximize the effective lifetimes and minimize the maintenance costs of these systems. In general, three main maintenance procedures have been widely investigated in the literature: corrective, preventive, and predictive. Corrective maintenance (also referred to as *run-to-failure* or reactive maintenance) involves operating equipment until failure. Preventive, or *periodic* maintenance, requires regularly scheduled maintenance actions to keep equipment in satisfying health status, therefore avoiding failures during system operation. Both of these approaches have drawbacks. While corrective maintenance allows systems to maximize their lifetimes, it involves high secondary costs, including the cost of maintenance due to cascade failures, increased system downtime

leading to revenue losses, and high costs for labor and spare parts. Preventive maintenance, on the other hand, yields unnecessary costs since predefined maintenance actions can lead to replacing healthy equipment, while simultaneously preventing systems from operating for the full length of their lifetimes.

In order to surpass these disadvantages, predictive or *condition-based* maintenance (CBM) involves real-time monitoring of degrading equipment to infer its health condition, which is considered hidden and not directly observable. Unlike corrective and preventive maintenance approaches, the main objective of CBM is the minimization of unexpected equipment failures. Thus, unnecessary maintenance actions can be avoided, and the useful life of the system is maximized. Figure 1 shows the cost benefit of condition-based maintenance over preventive and corrective maintenance options. A well-designed CBM strategy enables real-time diagnostics as well as

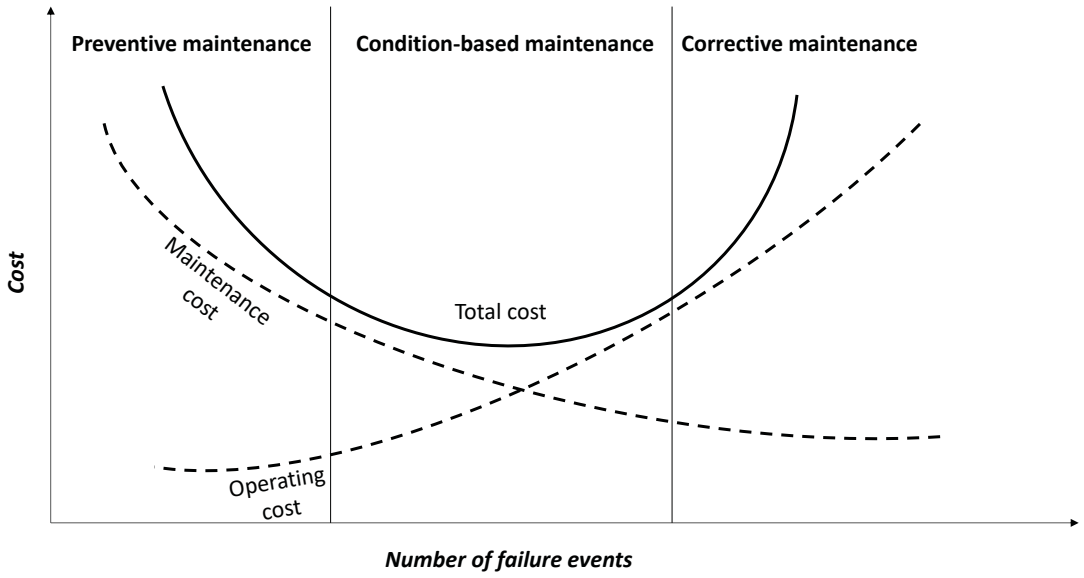


Figure 1: Cost-failure comparison for preventive, corrective, and condition-based maintenance options.

produces accurate predictions for an array of important prognostic measures, (e.g., remaining useful life (RUL)). For a CBM strategy to be successfully implemented, the development of proper stochastic models representing the evolution of the degradation process, as well as its relationship with the observation process (e.g., sensor measurements), are required. A plethora of mathematical models and software packages have been developed and numerous case studies have been reported pertaining to the successful application of CBM policies. Among these, state-space models (SSMs) have been recognized as a powerful and flexible class of time-series models for analyzing dynamic systems, and they have been widely applied for degradation and health monitoring in various application domains. The majority of SSMs used for system health monitoring follow a generic dynamic structure that involves two main processes: an autoregressive process that describes the evolution of the hidden/latent degradation state, and a process that shows the mapping of the degradation states into observable outputs collected from sensors. Since degradation is hidden, the only information we have for such dynamic systems comes from the sensor observations that are assumed to be noisy representations of the latent system dynamics. Such systems are known as *partially observable dynamic systems*. Among available statistical methods for analyzing SSMs, Bayesian filtering techniques have received significant attention due to their mathematical convenience and strong structural properties. In addition to SSMs and Bayesian filtering, many condition-based maintenance decision policies have been recently studied and developed using dynamic programming and Markov decision processes (MDPs). MDPs are characterized by actions and rewards that define the state of the system at any given time.

Despite their significant contribution to condition monitoring approaches, applications utilizing SSMs and MDPs in the current literature consider limiting assumptions that, while simplifying the problem, fail to capture the true complexity of the system dynamics. These assumptions are summarized into four main categories:

1. *User-defined failure threshold.* System failure is approximated in an entirely deterministic fashion, with a predefined failure threshold based on unreliable historical data. In these cases, a system fails when the hidden process exceeds the predefined threshold value (see Figure 2). This approach may work for when the hidden process expresses a certain identity (e.g., crack growth), but it is not realistic for abstract processes, such as degradation.

2. *Simplified latent dynamics.* The degradation process evolves independently of other latent processes, such as the system operating condition, at any given time (see Figure 3). Because of this, these approaches ignore operational complexities of complex systems where it is highly possible for the degradation process evolution to be directly affected by multiple levels of latent dynamics. The following state-space model is an example of the aforementioned simplified system dynamics with a single latent process (e.g., degradation) and a single sensor observation process that ignores any operational complexities:

$$\boldsymbol{x}_t = f_x(\boldsymbol{x}_{t-1}, \boldsymbol{\theta}_x) + \boldsymbol{\epsilon},$$

$$\boldsymbol{y}_t = f_y(\boldsymbol{x}_t, \boldsymbol{\theta}_y) + \boldsymbol{\delta}.$$

   A detailed description of state-space models is given in Chapter 2.

3. *Parametric form of the observation process.* The process stochastically relates hidden states to sensor observations by taking a parametric form that is defined

empirically, depending on the system under observation. This approach is unrealistic, since it is virtually impossible to define a formula that can accurately capture the mapping between hidden and observable states.

4. *Fully observable systems.* Previous studies in control and maintenance decision-making applications have assumed the availability of fully observable systems and/or full knowledge of the state transition and risk/reward models, both of which are unrealistic in complex systems with dynamic structures and various sources of uncertainty.

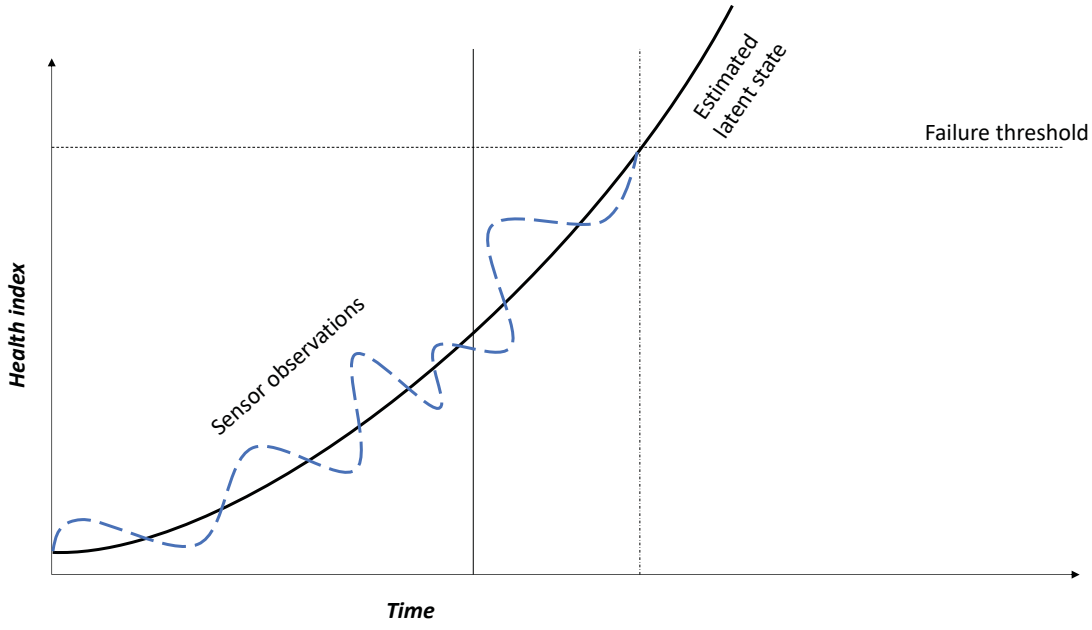


Figure 2: User-defined failure threshold. The dashed line represents sensor observations, while the continuous line denotes the actual latent state.

This thesis provides two data-driven hierarchical SSM structures that are able to describe the probabilistic dependences between latent states and sensor observations for both linear and non-linear system degradation dynamics. Both structures manage to

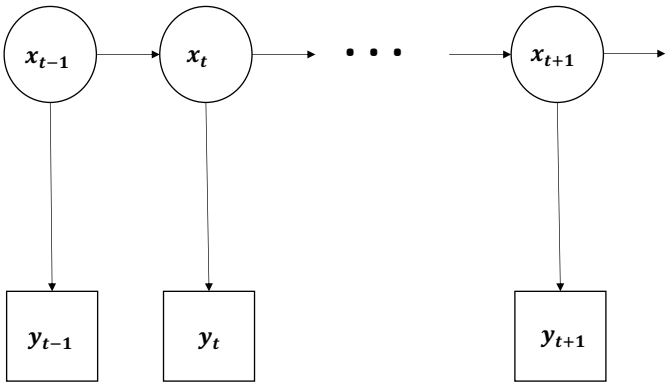


Figure 3: Simplified system dynamics.

address the aforementioned limitations, leading to more dynamic real-time monitoring and control procedures. In addition to these structures, this thesis demonstrates an original mathematical framework for dynamic control and decision-making that is based on deep reinforcement learning (DRL). This framework is built upon latent state estimates obtained from Bayesian filtering. It offers a monitoring, control, and decision-making approach that combines Bayesian statistics and artificial intelligence in an innovative manner.

1.1.1 Monitoring and Control of Systems with Linear and Gaussian Dynamics

This framework provides a new stochastic approach for real-time prognostics and health management of condition-monitored degrading systems. It is designed to describe purely linear system dynamics, both latent and observable. As previously explained, the degradation process is assumed to be partially observable over time, and the only available information is obtained through sensor observations and the working condition of the system at any given time. Aimed at addressing impor-

tant challenges arising from employing SSMs and classification methods in reliability analysis, this framework involves three levels of stochastic processes: a degradation process, an observation process, and a hazard process. The framework relates a) the Kalman filter, a Bayesian filtering method that infers the continuous degradation process by utilizing sensor observations, and b) the logistic regression, a binary classification method employed to connect the degradation process to the stochastic process associated with the system failure. Logistic regression has been independently used for fault detection and reliability analysis in previous works, mainly to relate the probability of failure to a set of covariates including, but not limited to, degradation-related features [1, 2, 3, 4]. The proposed model is able to separately infer the states of both degradation and hazard processes using the latest set of sensor observations. It also exploits the positive benefits of Kalman filter and logistic regression, such as mathematical simplicity, fast computation time, convexity, and parameter interpretability. Algorithms with such a cascade combination involving Kalman filter have also been utilized in other applications. Interested readers may refer to [5], which combines extended Kalman filter with the Gaussian Process regression for respiratory motion prediction and [6], which combines Kalman filter and RBF (Radial Basis Function) neural network for stellar spectral recognition.

1.1.2 Monitoring and Control of Hybrid Non-Linear Systems

The framework described in this Section is a generalized version of the framework presented in Section 1.1.1. It analyzes a fully hierarchical SSM for systems with many levels of latent and observable dynamics and highlights the dependencies between different system processes. The framework was designed to focus on sys-

tems with a non-linear degradation trend and can be easily applied to any shape of system dynamics. This method addresses the need for quantitative frameworks that can efficiently utilize multivariate time-series data and generate real-time insights for systems with dynamic operating conditions without relying on many distributional assumptions. To address system complexities, the method introduces a new hybrid state-space generative framework with multiple stochastic layers that include: i) a latent degradation state, ii) a latent operating condition state representing the operating mode of the system (e.g., faulty or normal), iii) a latent hazard process representing the systems probability of failure within the next observation interval, iv) an observation process representing the evolution of the multi-dimensional condition monitoring sensor measurements, and v) a binary failure process representing the overall working status of the system (i.e., working v.s. failure). For inferring multi-layered latent states, an original generalized hybrid particle filter was developed to accommodate all stochastic layers. This comprehensive framework relaxes many assumptions that exist in previous research by considering both system operation and failure as entirely stochastic processes, similar to the degradation process. It therefore illustrates the first part of the main model described in this thesis. A similar hybrid state-space structure for condition monitoring and RUL prediction with a flexible multiple-phase approach modeling the condition monitoring signal under multiple health conditions with different characteristics was previously proposed in [7, 8]. The thesis proposed methods consider latent degradation signals, while in [7, 8], the degradation signal is a one-dimensional observable signal following a parametric form. Furthermore, in [7, 8] there is no multidimensional observation process related to the degradation process. Finally, predefined failure thresholds were used in [7, 8], and not in this thesis.

1.1.3 Bayesian Filtering-based Dynamic Decision Making

The frameworks in Sections 1.1.1 and 1.1.2 were designed primarily to infer latent system states and, using those inferences, to determine important prognostic measures, such as the remaining useful life (RUL) estimation for degrading systems. However, a framework able to suggest the optimal time to perform maintenance based on cost minimization is also necessary. This thesis provides an original method for dynamic decision-making that exploits the advantages of deep reinforcement learning (DRL). This framework introduces a Bayesian filtering-based DRL in an intelligent and coordinated manner with two possible applications. First, a decision making approach that uses sensor data to determine when to perform maintenance prior to a system failure, based on the relative relationship between the costs of replacement and failure, was developed. Second, a prognostics method for remaining useful life estimation was developed to generates warnings based on the relative relationship between early warning and late warning signals. The original inputs are multidimensional sensor data that are stochastically related to the systems latent degradation state. None of the approaches depend on a system dynamics structure or prior distributional assumptions, making them highly generic and more applicable to a wide range of degrading systems.

1.2 Objectives and Contributions

The main objective of this thesis is to present a fully automated real-time control and decision-making structure that brings together Bayesian filtering and advanced machine learning methods. The only inputs needed are multidimensional sensor data

and the systems working condition at any given time. An array of outputs, including hidden states, RUL estimates, and optimal maintenance costs, can be obtained from every step of the process, depending on the users needs. The three frameworks described in the thesis can be divided into two main parts as:

- **Phase I:** Bayesian filtering frameworks (Sections 1.1.1-1.1.2).

- **Phase II:** Dynamic decision-making (Section 1.1.3).

The developed solutions have demonstrated capabilities and applicability to both simulated and real-world applications. This thesis provides a set of contributions that address the existing limitations in the current literature and were highlighted in Section 1.1. They are categorized based on each part of the structure:

- **Phase I:**

    1. *Condition Monitoring on Systems with Linear & Gaussian Dynamics:* The proposed framework does not take into consideration predefined, deterministic failure thresholds; rather, it considers system failure as a stochastic process that treats degradation process as an input to define the failure process. Other significant contributions include closed-form solutions for the marginal likelihood function to increase model training efficiency in the presence of large-scale data, as well as the following prognostic measures: i) mathematical formulas for RUL and reliability leading to faster CBM models capable of handling large-scale data, and ii) a dynamic, cost-effective replacement model for determining the optimal time of replacement using the most updated history of the monitoring data. This framework is not designed to outperform, in terms of accuracy, other methods currently

in the literature. Its purpose is to suggest a new method of handling partially observed degradation systems in a mathematically convenient manner. Nevertheless, it may perform poorly in cases where the linearity assumptions of the Kalman filter are not satisfied. Previously published works that involve Kalman filtering in similar cascade combinations have also been introduced for other application domains. Ref. [5] combined an extended Kalman filter with the Gaussian Process regression for respiratory motion prediction. Additionally, the study in [6] combined the Kalman filter and the RBF (Radial Basis Function) neural network for stellar spectral recognition. For all intents and purposes, the proposed framework is the first in the literature to combine Bayesian filtering with a classical binary classification technique for sensor-driven condition-based maintenance.

2. *Condition Monitoring for Hybrid Systems:* The main contributions of this framework, compared to current models for degradation monitoring and RUL prediction regarding systems with non-linear and/or non-Gaussian dynamics, can be summarized in the following ways: i) Similar to the framework in Section 1.1.1, this framework does not require a predefined failure threshold to relate the degradation state with the actual failure process. Instead, the failure process is defined as a two-state stochastic process that depends on a stochastic hazard process and other possible covariates. ii) The stochastic relationship between multidimensional sensor measurements and latent dynamics is defined by a non-parametric framework based on a neural network that can accommodate a variety of non-homogeneous

CM sensor measurements. The stochastic relationships defined for other variables in the developed hybrid SSM can be defined by any arbitrary distribution; that is, the framework has no predefined parametric and distributional assumptions. iii) The proposed framework develops an iterative unsupervised learning approach based on the Expectation-Maximization (EM) algorithm. The Markov Chain Monte Carlo (MCMC) Metropolis-Hastings algorithm and hybrid particle filtering were employed to train the model using only past observable sensor measurements. An iterative approach for RUL estimation is developed.

- **Phase II:**

1. *Dynamic Decision-Making:* This framework claims the following contributions: i) a real-time control and decision making method for system maintenance that combines the capabilities of Bayesian filtering and DRL; ii) an original Bayesian filtering-based prognostics framework that employs DRL for RUL estimation using various types of warnings depending on the users preferences; iii) proving that DRL can infer system control policies using a stochastic representation of the systems latent states over time without the need to develop full sweeps of the state space or exhaustive back-ups. The developed DRL frameworks generalize historical sensor observations with new and previously unseen states and system conditions to make real-time actions. Additionally, the frameworks can be used for large problems with large state spaces, including continuous space systems. Finally, they are model-free, and no explicit distribution for system dynamics, state transition, sensor data, or risk/reward functions are required.

1.3 Examples of Real-world Applications

Bayesian filters and reinforcement learning have been used in many studies with important applications in degradation monitoring, smart grid control, and healthcare. A list of notable examples is given below:

- *Degradation monitoring:* The Kalman filter has been used for prognostics and RUL estimation on many systems including automatic transmission clutches in automobiles [9], aircraft fuel systems and helicopter gearboxes [10, 11], and solenoid valves [12].

  Applications of particle filters in prognostics and RUL have been reported for steam generator tubes, which are essential for the smooth operation of nuclear power plants [13], lithium batteries [14], high-speed shaft bearings of wind turbines [15], and auxiliary power units [16].

  Interesting reinforcement learning applications for dynamic decision-making have also been recently reported, such as traffic proactive congestion control [17]. Ref. [18] demonstrated the application of DRL for industrial process controller maintenance, with applications in paper-making machines, high purity distilliation columns, and HVAC systems. Finally, a reinforcement learning application for anomaly detection can be found in ref. [19], where it was used to improve the accuracy of predictive analytics models for financial market returns.

- *Smart grid control:* Bayesian filters have also found numerous applications in the area of smart grid control. Ref. [20] introduced and compared an adaptive Kalman Filter with inflatable noise variances against a variety of classic Kalman filters for improving the quality of monitoring and controllability in smart grids.

The study in [21] described a Kalman filter-based optimal feedback control method for the microgrid state estimation and stabilization.

Regarding particle filter applications in smart grid control, ref. [22] proposed a particle filter-based grid synchronization scheme for distributed generators in order to estimate the phase angle of grid voltage signal in a smart grid. In a different approach, the study in [23] targets at reducing the communication bandwidth overload and achieving the real-time state estimation of a smart grid network using an event-triggered particle filter.

The complexity of smart grid systems is the most important factor in developing reinforcement learning applications for monitoring and control. Ref. [24] surveys both past and recent reinforcement learning considerations to solve power system control and decision problems. Study [25] proposed autonomous broker agents for retail electricity trading, able to operate in a wide range of smart electricity markets. These agents are capable of deriving long-term, profit-maximizing policies and use reinforcement learning to accommodate arbitrary economic signals from their environments.

- *Healthcare:* Yet another area that Bayesian filters and artificial intelligence find important applications is in healthcare. The study in [26] described the application of Kalman filter for extracting heart rate variability from photoplethysmogram (PPG) signals, which can then be used for diagnosing important health indices like heart disease. Kalman filter was also used in [27] for medical monitoring, particularly for detecting kidney transplant rejection.

  Particle filters have also been used in healthcare studies. Ref. [28] demonstrates the application of particle filters in the context of personalized medicine, where

the model-based assessment of drug effectiveness plays an important role. Inference on these systems based on data gained from clinical studies with several patient groups becomes a major challenge, and particle filters are a promising approach for tackling these difficulties. A technical report presented in [29] surveyed the robustness of different particle filter applications regarding the localization and delineation of anatomical structures in medical images.

Reinforcement learning has also found broad applications in healthcare. A detailed survey of these studies can be found in [30]. Furthermore, ref. [31] provides some important guidelines for applying reinforcement learning for decisions regarding patient treatment.

1.4 List of Publications

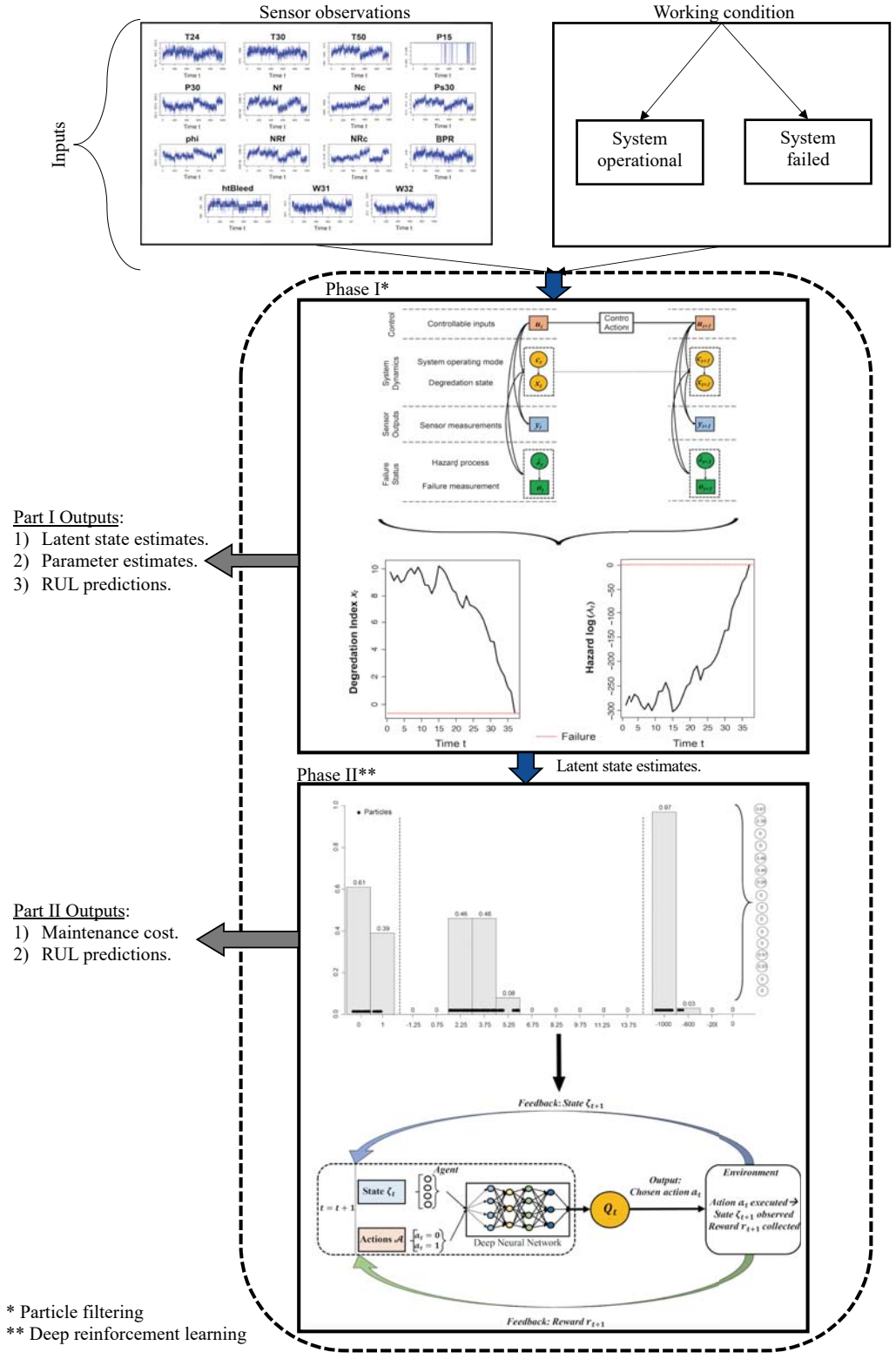
1. Skordilis, E. and Moghaddass, R. A Deep Reinforcement Learning Approach for Real-time Sensor-Driven Decision Making and Predictive Analytics. *Computers & Industrial Engineering*, submitted for publication, Sep 2019.

2. Skordilis, E. and Moghaddass, R., 2019. A Double Hybrid State-Space Model for Real-Time Sensor-Driven Monitoring of Deteriorating Systems. *IEEE Transactions on Automation Science and Engineering, DOI: 10.1109/TASE.2019.2921285.*

3. Skordilis, Erotokritos, and Ramin Moghaddass. "A Hybrid State Particle Filter for Failure Prognosis in Deteriorating Systems." *In 2018 Annual Reliability and Maintainability Symposium (RAMS)*, pp. 1-6. IEEE, 2018.

4. Skordilis, E. and Moghaddass, R., 2017. A Condition Monitoring approach for real-time monitoring of degrading systems using Kalman filter and logistic regression. *International Journal of Production Research, 55(19)*, pp.5579-5596.

5. Moghaddass, R., Mohammed, O.A., Skordilis, E. and Asfour, S., 2019. Smart Control of Fleets of Electric Vehicles in Smart and Connected Communities. *IEEE Transactions on Smart Grid*, 10 (6), pp. 6883-6897.



Structure overview: **Phase I:** SSM and hidden state inferrence; **Phase II:** Dynamic decision-making.

CHAPTER 2

Related Work

Timely scheduling of maintenance activities for critical equipment is one of the most important real-time control decisions that many industries must take in order to avoid high costs of failures and resulting downtimes. Over the years, a large and growing body of literature has investigated numerous real-time control and decision-making approaches for industrial systems and equipment due to the steady demand for increased reliability. This work has presented numerous algorithms and solution approaches for decision-making applications, including system maintenance [32]. Both machine health prognosis, or the ability to assess current health status of a deteriorating machine, and the forecasting of future degradation trends for maintenance scheduling [33] have also received a considerable amount of attention in the last decade. Despite this attention, the utilization of real-time data for diagnosis, prognosis, and maintenance decision-making regarding systems with a partially observable degradation process is a long-standing problem in the management of mechanical asset maintenance under condition monitoring. These systems are also known as systems with incomplete state information, imperfect information, or partially observed degrading systems [34]. Advanced sensor technologies and condition monitoring tools have been extensively developed during the past decade, producing large-scale system

condition datasets and rapidly disrupting the field of maintenance decision-making for partially observable degrading systems. The contemporary industrial field has become extremely competitive, increasing the importance of maintenance scheduling in fulfilling the unique requirements of modern production systems [35]. However, until recently, such data could not be effectively analyzed due to a lack of efficient analysis methods [36].

2.1 Condition-Based Maintenance

Condition-based maintenance (CBM), aided by condition monitoring devices and sensors, provides one of the most efficient real-time maintenance strategies for degrading systems. CBM involves the real-time analysis of data collected through sensors and performance of maintenance activities, which are determined based on the condition of the system. In contrast to corrective and preventive maintenance approaches, CBM demonstrates an innovative maintenance approach that aims to minimize the instances of equipment failures that lead to maintenance activities and costs; at the same time, it aims to maximize the useful life of the system - where maintenance actions are performed only when a failure is imminent - in order to ensure safety, reliability, and a reduction of the total life costs of the system. A plethora of mathematical models and software packages capable of performing CBM actions have been developed; successful applications have been reported to multiple case studies, as can be seen in [37, 38]. Ref. [39] presented the first review and analysis for the various methods dealing with prognostic-based decision support for CBM within complex manufacturing environments. These methods have generally diversified characteristics, and their implementation is independently determined by each CBM application.

Generally, four main technical processes that comprise the course of machinery prognostics have been identified: data acquisition, health indicator construction, health stage division, and RUL prediction. A comprehensive review on these processes can be found in [40].

2.2 State-Space Modeling

Successful implementation of CBM requires an accurate representation of the systems dynamics. This can be achieved using mathematical structures representing the latent degradation process and its stochastic connection to condition monitoring data. The most popular and widely used mathematical approach for modeling the degradation of partially observable mechanical systems is the use of state-space models (SSM) of time-series [41]. SSMs are highly popular because of their mathematical convenience and reasonable way of modeling and analyzing dynamic systems with multiple inputs and outputs. They have been widely applied for degradation and health monitoring in various application domains. SSMs can be categorized as physics-based, data-driven, or a hybrid combination of the two [42]. Physics-based SSMs are used when explicit knowledge of the physical properties that describe the degradation behavior is available. These, combined with measured data, are able to infer future system dynamics. In the absence of such models, data-driven SSMs are able to utilize historical data for identifying system dynamics and predicting the future behavior. Hybrid models combine both approaches; research on these is relatively recent. Most SSMs follow a stochastic-dynamic structure that involves two types of variables: the latent degradation state and the observable system outputs. In these models, observations are assumed to be noisy representations of the latent

degradation process [43, 44, 45, 46]. The most important feature of latent states is that they follow the *Markov property* - when the state of the system at time t depends only on the state's value at time $t-1$. Conversely, observations between consecutive time intervals are conditionally independent. Assuming that $\boldsymbol{x}_t$ denotes an n-dimensional continuous-valued latent degradation process and $\boldsymbol{y}_t$ denotes an m-dimensional observation process, an SSM can be presented as follows:

$$\boldsymbol{x}_t = f_x(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t, \boldsymbol{\theta}_x, \boldsymbol{\epsilon}),$$

$$\boldsymbol{y}_t = f_y(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\theta}_y, \boldsymbol{\delta}),$$

Stochastic functions $f_{\cdot}$ control the evolution of model variables, while $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_x, \boldsymbol{\theta}_y\}$ are model parameters that characterize the patterns of these functions. Variable $\boldsymbol{u}_t$ represents a d-dimensional vector of controllable and uncontrollable operational inputs that are problem-specific. Both processes are perturbed by statistical noise $\boldsymbol{\epsilon}$ and $\boldsymbol{\delta}$ for the system and observation noises. These noise processes are considered time-invariant, and are usually assumed to follow normal distributions with zero means and covariance matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, as:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_t),$$

$$\boldsymbol{\delta} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_t).$$

Despite the fact that most of the SSMs in the literature used for health monitoring are focused on system state tracking, diagnostics, and estimation (as can be seen in [47]), SSMs have also found application in prognostics of degrading systems, particularly RUL estimation. Ref. [48] shows an SSM that models a latent state process for degradation prognostics with a RUL estimation of a gas turbine. An SSM was presented in [49] for lithium-ion batteries, where the observation model was used to approximate

the discharging profile at each time step. The model was later used for RUL prediction. A common case of SSM is the Hidden Markov Model (HMM), where variable dependencies are represented by a Bayesian network [50]. In HMMs, the degradation process follows a predefined latent discrete or continuous process, whereas an observation process stochastically relates latent degradation states to sensor observations over time. Recent examples of HMMs for degradation diagnostics and prognostics can be found in [2, 51]. It can be safely assumed that SSMs are invaluable in describing latent and observed system dynamics. However, classic SSMs sometimes fail to adequately describe these dynamics due to modern industrial systems complexities, which are expressed by multiple latent processes, such as system operating condition. These processes affect the degradation process significantly and need to be taken into consideration. Systems with such multilevel dynamics are referred to as hybrid systems (not to be confused with the hybrid models described earlier, which denote a cross-section between physics-based and data-driven approaches). Operating conditions, expressed as discrete states, are usually represented by modes [52]. In order to properly describe such systems, hybrid state-space models (HSSMs) are required. HSSMs, as the name suggests, combine a latent discrete mode operation process, a latent continuous degradation process, and a sensor observation process. In HSSMs, the evolution of both the degradation and observation processes is directly affected by changes in the operation process, leading to fully hierarchical modeling. HSSMs have found substantial applications in state estimation and tracking (e.g., [53]).

However, few works have utilized hybrid state space systems for prognostics and RUL prediction. Ref. [54] developed an HSSM for sensor fault detection and identification, where the dynamics of the system consisted of a discrete process modeling

sensor states and a continuous process for tracking changes of the system parameters. The set of possible sensor states included normal, faulty, and partially faulty states. A general hybrid state dynamic model was used in [55] to represent system behavior under normal and faulty operating conditions; later, it was used for RUL prediction regarding a UH-60 planetary carrier plate. A similar approach appeared in [56], where a HSSM was utilized for determining the state and the RUL of a rotor bar in an induction motor. Discrete and continuous variables represented the state of the bar and an indicator value of a defect, respectively.

2.3 Bayesian Filters

SSMs are employed with the purpose of inferring latent states using sensor observations. This procedure is known as state inference, and it allows system dynamics to be modeled as random variables represented by unknown probability density functions. These random variables can be estimated through recursive Bayesian filtering over time using incoming observations and the defined SSM [57]. Bayesian filtering computes the posterior probability density function (PDF) $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$, which occurs by recursively by applying two steps: a *prediction* and an *update*. During the prediction step, the prior PDF $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$ is computed using the filtering distribution $p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1})$ and the properties of Chapman-Kolmogorov equation as:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1})d\boldsymbol{x}_{t-1}.$$

The update step incorporates a new measurement vector $\{\boldsymbol{y}_t\}$, the prior PDF $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$, the likelihood function $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$, and the Bayes rule to estimate the posterior PDF

over $\boldsymbol{x}_t$ as:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) = \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{p(\boldsymbol{y}_{1:t}|\boldsymbol{y}_{1:t-1})}$$
$$\propto p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}).$$

For brevity, notation regarding operating inputs $\boldsymbol{u}$, noise $\boldsymbol{\epsilon}, \boldsymbol{\delta}$, and parameter vector $\boldsymbol{\Theta}$ was omitted. This process demonstrates how the optimal Bayesian solution is obtained. However, recursive computation of PDF usually cannot be executed analytically, except in special cases where SSMs have linear and Gaussian forms. In these cases, the well-known Kalman filter is proved to be the optimal estimator. However, when non-linear and/or non-Gaussian dynamics are present, particle filtering is then considered the optimal option. Both of these methods have been extensively used for state-parameter estimation in CBM applications (see examples [58, 59, 60, 61]).

2.3.1 Applications of Kalman Filter

The Kalman filter has been employed in multiple occasions for fault detection [62] or for modeling the dynamic behavior of degrading systems [63]. The majority of Kalman filtering-based cases for reliability analysis consider single-unit deteriorating systems. Ref. [64] used the Kalman filter for preventive maintenance of a DC motor. The angle displacement of the motor was considered the hidden state and its derivative, the rotating speed, was considered the observed output. The parameters of the proposed SSM were known *a priori*, and no estimation procedure was conducted. In [63], a condition-based failure prediction and processing scheme for preventive maintenance of a thermal power-plant was proposed. A failure Petri Net was established to describe relations between hidden states such as fuel flow and temperature and

noisy observations obtained by their respective sensors. Kalman filtering was used for state estimation based on these observations. The study in [65] examined the structural damage of grid array interconnections during vibration using a Kalman filtering approach. The interconnected resistances were modeled as hidden states, and the measurement noises were considered observations. The Kalman filter was utilized for future state estimation and RUL prediction based on these noises. In [66], a method for predicting the state of damage on a steel band undergoing vibration for prognostic maintenance was presented. The Kalman filter was used for state estimation and a predefined frequency value was considered as a failure threshold. Ref. [67] presented a process for estimating the RUL of a single-stage gearbox connected to a DC motor was presented. The Expectation-Maximization algorithm was used to estimate the parameters of the underlying state-space model, while the Kalman filter was used to model the distribution of the hidden system states with respect to the current parameter estimates and the system outputs. A case study of condition monitoring for an induction furnace was presented in [68], where the authors employed Kalman filtering for state estimation. A decision-making policy for inductor replacement was also introduced.

As previously mentioned, the Kalman filter is the best linear estimator, in that it requires system dynamics to follow Gaussian distribution. Therefore, its application is limited due to the ubiquitous non-linearity that governs most real world system dynamics. Various extensions have consequentially been introduced, which have additionally been used in reliability analysis:

- **Extended Kalman filter (EKF):** The Taylor series is used to linearize the non-linear system dynamics before proceeding to state-parameter inference (e.g., [43, 69, 44]).

- **Unscented Kalman filter (UKF):** A similar approach to EKF, but in this case, system dynamics are linearized using samples drawn from suggested prior distributions of the random variables and linear regression (e.g., [70, 71]).

- **Switching Kalman filter (SKF):** A piece-wise version of the Kalman filter, where a different set of linear functions is used at every time step following changes on the pattern followed by the system dynamics (e.g., [45]).

2.3.2 Applications of Particle Filter

Particle filters present interesting properties that overcome some of the fundamental drawbacks found in Kalman filters, since they provide a consistent theoretical framework for handling non-linear and/or non-Gaussian systems [55]. Thorough reviews on particle filters and their applications can be found in [72, 73, 74]. Particle filtering is frequently used for RUL predictions because of its reasonable and theoretically accepted manner for characterizing future uncertainty in real time [40]. The majority of particle filtering-based works for condition monitoring focus on generic SSMs with a single latent process. However, hybrid particle filters (HPF) that can perform state-parameter inference in HSSMs have also appeared in various application domains, such as robot navigation [75, 76, 77, 78, 79, 80] and target tracking [81, 82, 83]. Other applications of HPFs can be found in sensor fault detection and identification, e.g., the study in [54], where the process of online detection and validation of fault status was investigated using a model containing both discrete and

continuous states. An application of HPF for state tracking and fault detection was also presented in [84]. Ref. [53] utilized a fuzzy-based HPF for developing an innovative state estimation approach. Applications of HPF outside fault detection include, but are not limited to, complex-equation solution [85], jump Markov systems [52, 86], and temporal intensity modeling in medical science [87].

Unlike fault diagnostics and state estimation, a limited number of examples using particle filtering for prognostics exist. According to [88], no more than 50 published works have used particle filtering for prognostics and RUL estimation. Some of these applications can be found in domains such as drilling processes [89], heating, ventilation, air-conditioning systems [90], and batteries [91]. Refs. [55], [92], and [93] described a framework of two autonomous modules - a helicopter planetary gearbox and a bearing - for fault detection and isolation (FDI), and failure prognostics. These studies proposed an HPF with discrete operating conditions represented by Boolean states and a continuous latent degradation evolution process. In another example, the study in [94] demonstrated an HPF for gas turbine engine condition monitoring by considering a continuous state process for degradation and a discrete state process for system health. Ref. [95] developed a combined model-based/data-driven approach for prognostics in an industrial Proton Exchange Membrane fuel cell. The study in [96] developed a prognostic framework for state of health (SOH) and RUL prediction of energy storage devices using particle filtering. Two hidden state processes were considered for the battery SOH and additional available SOH due to the regeneration phenomena. Other similar prognostics approaches based on HPF can be found in [97], and [98].

2.3.3 Other Approaches

Apart from classic Bayesian filtering approaches, recent studies have proposed advanced methods of state-parameter estimation that combine the abilities of Sequential Monte Carlo and Markov Chain Monte Carlo (MCMC) algorithms into frameworks that are extremely capable of sequential Bayesian inference in SSMs. The study in [99] presented a new state-parameter inference method called *particle marginal Metropolis-Hastings* algorithm, where particle filtering is used for proposal distribution for the Metropolis-Hastings MCMC algorithm. A natural extension of this method was thoroughly described in [100]. Both methods have shown great potential in the field of state-parameter estimation, and future research is worthwhile.

2.4 Remaining Useful Life Estimation

Remaining useful life (RUL) estimation represents one of the most important prognostic measures and has gathered significant attention in dynamic system monitoring and control methods. Recent RUL estimation examples can be found in [101], where the problem of fault prognosis on an industrial gas turbine under transient conditions was investigated. The method was then validated on RUL estimation of a degrading gas turbine system. Ref. [102] focused on the development of a multi-level condition monitoring policy for a system with multiple units that can experience both soft and hard failure modes while considering economic dependencies between them. In [103], a fault prognostic and RUL prediction method for a wind turbine gearbox that used adaptive neuro-fuzzy inference and particle filtering was presented. Ref. [104] addressed the issue of inaccurate RUL predictions for early failed systems by propos-

ing a signal-based RUL prediction method focused on imbalanced historical data. A novel approach that combined different categories of observations by inducing a proper health index was also proposed in [105]. Health indices are generally used for fault diagnostics and prognostics by measuring equipment degradation, and they constitute convenient tools for RUL estimation. Table 1 provides recent works for a number of RUL estimation methods. Other data-driven methods for RUL estimation

Table 1: Samples of recently published RUL estimation methods.

CBM metric	Type	Reference
RUL estimation	Auto-Regressive models	[106, 107, 108]
	Random coefficient models	[98, 109]
	Wiener process models	[110, 111]
	Gamma & Inverse Gamma process models	[112, 113]
	Proportional Hazards models	[114, 115, 34]

in the literature include artificial neural networks [116, 109], support vector machines [117, 118], and, more recently, deep learning [119, 120].

2.5 Markov Decision Processes for Real-Time Monitoring and Control

Dynamic condition-based maintenance decision policies have also been developed using Markov decision processes (MDPs). MDPs are discrete time stochastic control processes and are characterized by actions and rewards that define the state of the system at any given time. Previous works on dynamic system monitoring and control were focused on specific mathematical optimization procedures that are used for studying MDPs, with dynamic programming (DP) being the most prominent among them. DP is a very convenient decision-making approach where decisions are reached

by defining a sequence of value functions that are solved separately. DP was used in [121] to develop an optimal CBM policy for multi-unit systems, whereas the study in [122] presented a dynamic optimal policy based on backward DP for choosing cost-effective maintenance actions for wind turbine gearbox operations. Ref. [123] presented a DP-based opportunistic preventive maintenance scheduling method for multi-unit series systems. The study in [124] described a robust DP model for deriving optimal control policies of degrading systems in case the proposed degradation model return inaccurate posterior distributions.

2.5.1 Model-Free Control Approaches

A major drawback of DP is the necessary requirement of full access to the entire state transition and reward model in order to solve the Hamilton-Jacobi-Bellman equation that forms the basis of DP. This model is mostly unknown in modern systems. Therefore, model-free methods capable of obtaining optimal control policies, such as reinforcement learning (RL) and, more specifically, $\mathcal{Q}$-*learning*, are necessary. $\mathcal{Q}$-learning solves the Hamilton-Jacobi-Bellman equation without relying on any parametric model form, but instead uses either $\mathcal{Q}$-tables or approximation functions, such as artificial neural networks (ANNs). The process learns to make proper decisions by observing its own behavior and the responses returned from the system. Based on these, RL uses built-in mechanisms that can improve the control policy through a reinforcement course of action [125]. RL has recently found applications in real-time control methods. The study in [126] utilized RL for obtaining optimal maintenance and control policies for deteriorating stochastic production/inventory systems. A different approach can be found in [127], which presented a semi-MDP

model for describing the degradation of a flow line system and applied RL to obtain a control-limit maintenance policy for each machine in the flow line. Ref. [128] presented a framework for resilient design and the operation of cyber-physical systems via self-organization and control reconfiguration strategies using RL with the purpose of increasing the lifetime of these systems. Similarly, [129] used RL to derive an optimal dynamic software rejuvenation policy by maximizing steady-state system availability using a semi-Markov decision process. These studies demonstrated the potential of using RL in real-time control and decision policies. However, they assumed small state-action spaces that do not require heavy computations and can be solved using static methods ($\mathcal{Q}$-tables) to store value function solutions. While this approach is simple and resilient, it cannot be used when the state space and the number of possible actions are very large, as is the case for modern industrial systems. In these cases, function approximators such as ANNs are more appropriate to use.

ANNs are generally structured as three-layered networks that include an input layer, a hidden layer, and an output layer. Advancements in computer science and hardware have eventually allowed the development of multi-layer neural network architectures with two or more hidden layers, leading to *deep neural networks (DNNs)* [130, 131]. The term *deep* implies multiple hidden layers that are able to approximate functions of increasing complexity. Specialized designs that allow for recurrence and convolution are among the most powerful DNN structures and have applications in both classification and regression. A thorough description of ANNs/DNNs, utilized for the models presented in this thesis, is given in subsequent Chapters. The advancements in DNNs led to the rising of *deep reinforcement learning* (DRL), with significant breakthroughs that include the deep $\mathcal{Q}$-network [132], and AlphaGo [133].

DRL developed control policies for previously intractable problems, managing to surpass human experts in areas such as Go. Reviews on the various applications of DRL can be found in [134, 135]. DRL will play a major role in the development of general artificial intelligence. Both deep learning and reinforcement learning were included in the MIT Technology Review list of the ten breakthrough technologies for the years 2013 and 2017 ([136, 137]). However, the mechanisms that underpin potential benefits of applying DRL in condition monitoring and control are not thoroughly explored.

2.5.2 Limitations in the Current Literature

Previous studies have introduced a variety of CBM approaches that have expanded the understanding of this area. However, they are all bounded by the assumptions that were described in the previous Chapter and are briefly demonstrated below:

1. *State-space models:* Despite the ability of typical SSMs to reasonably represent the dynamics of a CM degradation process, they are unable to accurately determine the systems failure status without defining a relationship between the degradation process and the failure process. Many models assume deterministic approaches, such as a predefined (known) threshold for degradation state, to determine the occurrence of the failure. Such a fixed and known threshold often does not exist for real systems and, even if it exists, it cannot accommodate uncertainty.

2. *Sensor observation processes:* Most available SSMs have assumed a fixed parametric relationship between thedegradation state and the set of CM signals (observation process). Such a parametric relationship (which is often made for mathematical convenience) is not only difficult to analytically prove but is also

hard to validate empirically, particularly in high-dimensional settings with many types of sensor outputs. In addition to the main challenges described above, there is no approach available to deal with the joint estimation of states and parameters for an HSSM with multiple layers that does not consider parametric or distributional assumptions between latent states and the multi- dimensional observation states.

3. *Kalman filter:* Despite its ability to represent the dynamics of a condition-monitored degradation process, the Kalman filter cannot directly provide information on the failure status of the system unless another process is employed to relate the degradation process to the failure process. The majority of research reported in the literature has simply used a deterministic approach, where a predefined failure threshold, based on the state or the output process, is defined to determine the occurrence of failure (e.g., [133], [46], [11], [64], [63]). Among these research studies, very few have discussed how this threshold can be determined or estimated in practice.

4. *Logistic regression:* This has been used independently for fault detection and reliability analysis in previous years, with the primary aim of relating the probability of failure to a set of covariates that include, but are not limited to, degradation-related features (e.g., [1], [2], [3], [4]). In these works, the inputs of the logistic regression, as well as the measures used to represent the degradation process, are assumed to be fully observable over time.

Finally, there is great potential in the recent advancements of artificial intelligence that has not fully been realized in the domain of sensor-driven prognostics and system

maintenance decision-making. Previous studies in control and maintenance decision-making applications have assumed the availability of fully observable systems and/or full knowledge of the state transition and risk/reward models. Both of these are unrealistic in complex systems with dynamic structures and various sources of uncertainty. For this reason, it is necessary to develop real-time control approaches that take into consideration the uncertainty of system dynamics and are tractable without making too many distributional and parametric assumptions. Furthermore, new methods are needed to find optimal maintenance times and to predict the remaining useful life of systems; these should depend mainly on sensor observations and be able to be sufficiently trained with past data.

2.6 Thesis Contributions

This thesis manages to address the limitations explained in Section 2.5.2, by introducing a condition-monitoring, control, and decision-making structure that combines Bayesian filtering algorithms and recent artificial intelligence approaches. The first phase of the proposed structure presents two condition monitoring and control frameworks based on Bayesian filtering; they manage to address these shortcomings and provide new insights in the area of sensor-driven prognostics. Specifically, these two frameworks introduce SSM structures with additional latent dynamic processes that: i) remove the necessity for establishing predefined failure thresholds, and ii) avoid distributional assumptions for describing the probabilistic dependencies between system dynamics levels. The second phase of the proposed structure provides a dynamic decision-making framework that highlights the great potential of recent advancements in artificial intelligence which have not yet been fully realized in the domain of dy-

namic system maintenance and prognostics. This framework introduces new methods for developing real-time control and decision-making policies using i) Bayesian filtering for making inferences regarding the latent state of the system, and ii) DRL in which agent actions are chosen based on a stochastic environment with states inferred by Bayesian filtering. The second phase also provides a first-of-its-kind sensor-driven RL-based decision-making approach, where the environment that the agent observes is entirely stochastic and inferred using Bayesian filtering and sensor observations..

CHAPTER 3

Preliminaries

An overview of the theoretical foundation of the various mathematical methods employed is provided here. Sections 3.1 and 3.2 provide a review of Kalman filter and logistic regression methods. Section 3.3 introduces the generic form of a particle filter with a single latent process and a single observation process. The hybrid particle filter is then demonstrated in Chapter 5. Sections 3.4, 3.6, and 3.7 describe three state-parameter inference algorithms that were applied at the proposed frameworks. Section 3.8 provides a detailed description of a set of neural network architectures that were employed as function approximators in Chapters 5 and 6. More specifically, deep/recurrent neural networks are presented in Sections 3.8.2-3.8.3, and the Extreme Learning Machine is described in Section 3.8.4. Finally, a description of reinforcement and deep reinforcement learning methods is presented in Section 3.9.

3.1 Kalman Filter

Kalman filtering represents a set of mathematical equations that implement a predictor-corrector estimator and is considered optimal (e.g., minimization of error covariance) when both latent and observed processes are linear and normally distributed. In this sense, Kalman filter is the best *linear* estimator for sequential state-

parameter estimation. The Kalman filter implements the prediction and update steps given in Section 2.3 to approximate the posterior distribution $p(\boldsymbol{x}_{1:t}|\boldsymbol{y}_{1:t})$. The equations in the prediction step obtain prior estimates for degradation and observation state values, whereas the equations in the update step incorporate new observation measurements that provide an improved estimate of the posterior distribution [138]. Prediction and update steps are executed recursively each time a new measurement is acquired, and are described below:

$$\text{Prediction Step}: \begin{cases} \hat{\boldsymbol{x}}_{t|t-1} = \boldsymbol{F}_t \hat{\boldsymbol{x}}_{t-1|t-1}, \\ \boldsymbol{P}_{t|t-1} = \boldsymbol{F}_t \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t^\top + \boldsymbol{Q}. \end{cases} \tag{3.1}$$

$$\text{Update Step}: \begin{cases} \tilde{\boldsymbol{z}}_t = \boldsymbol{y}_t - \boldsymbol{H}_t \hat{\boldsymbol{x}}_{t|t-1}, \\ \boldsymbol{S}_t = \boldsymbol{H}_t \boldsymbol{P}_{t|t-1} \boldsymbol{H}_t^\top + \boldsymbol{R}, \\ \boldsymbol{K}_t = \boldsymbol{P}_{t|t-1} \boldsymbol{H}_t^\top \boldsymbol{S}_t^{-1}, \\ \hat{\boldsymbol{x}}_{t|t} = \hat{\boldsymbol{x}}_{t|t-1} - \boldsymbol{K}_t \tilde{\boldsymbol{z}}_t, \\ \boldsymbol{P}_{t|t} = (\mathbb{I} - \boldsymbol{K}_t \boldsymbol{H}_t) \boldsymbol{P}_{t|t-1}, \end{cases} \tag{3.2}$$

where $\hat{\boldsymbol{x}}_{t|t-1}$ and $\hat{\boldsymbol{x}}_{t|t}$ represent the prior and posterior degradation state estimates, respectively. Matrices $\boldsymbol{P}_{t|t-1}$ and $\boldsymbol{P}_{t|t}$ represent the prior and posterior covariance estimates, respectively. Finally, $\boldsymbol{y}_t$ is the sensor observation, $\tilde{\boldsymbol{z}}_t$ is the measurement residual, $\boldsymbol{S}_t$ is the residual covariance, $\boldsymbol{K}_t$ is the optimal Kalman gain at time t, and $\mathbb{I}$ is the identity matrix. The status of the system can then be monitored in real-time using $\hat{\boldsymbol{x}}_{t|t}$ and $\boldsymbol{P}_{t|t}$.

3.2 Logistic Regression

Logistic regression is a binary classification method that can be used to formulate the failure probability of a system using a binary assumption for the system's

operating condition. That is, the system can either be at the working state (class 0), or the failure state (class 1). Therefore, a hazard function λ_t can be defined as the conditional probability of failure between time $t-1$ and t, given the system's survival up to time $t-1$. Function λ_t normally depends on a set of covariates, the most important of which is the true degradation state ($\boldsymbol{x}_t$). Other covariates may include the age of the system (t), control inputs ($\boldsymbol{u}_t$), or more latent states, covariates and constants. Logistic regression is able to define the probability of the system being in each class at time t, given that it has survived up to time $t-1$ using the following equation:

$$\lambda_t = \frac{1}{1+\exp\big(-(\boldsymbol{\alpha}\boldsymbol{x}_t + \boldsymbol{\beta}\cdot\mathcal{C} + \beta_0)\big)} \equiv \sigma((\boldsymbol{\alpha}\boldsymbol{x}_k + \boldsymbol{\beta}\cdot\mathcal{C} + \beta_0)),$$

where $\sigma(.)$ is the *logistic sigmoid* function, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the regression coefficients associated with degradation state and the rest of covariates, respectively, and β_0 is the intercept. Set $\mathcal{C}$ denotes all covariates except the degradation state.

3.3 Particle Filter

Particle filtering (PF) is a Sequential Monte Carlo algorithm, and uses a set of particles to approximate the posterior probability density function $p(\boldsymbol{x}_1|\boldsymbol{y}_{1:t})$. The advantage of PF over Kalman filter is that the former can be used regardless of the shape of system dynamics. Particle filter approximates $p(\boldsymbol{x}_{1:t}|\boldsymbol{y}_{1:t})$, with a weighted set S of N_s samples, or particles

$$S = \{\boldsymbol{x}^i_{1:t}, w^i_t\}^{N_s}_{i=1}.$$

Particles $\boldsymbol{x}^i_{1:t}$ represent latent state values and are associated with weights w^i_t that denote approximations to the relative particle posterior probabilities. To remove

biases between particles, weights need to be normalized:

$$\sum_{i=1}^{N_s} w_t^i = 1.$$

The posterior PDF is then estimated as:

$$\hat{p}(\boldsymbol{x}_{1:t}|\boldsymbol{y}_{1:t}) \approx \sum_{i=1}^{N_s} w_t^i \delta_{\boldsymbol{x}_{1:t}}(\boldsymbol{x}_{1:t}^i),$$

where $\delta_{\cdot}$ denotes the delta Dirac function centered at particles $\boldsymbol{x}_{1:t}^i$. Theoretically, as $N_s \to \infty$, then the filter approximates the actual posterior PDF [72].

3.3.1 Sequential Importance Sampling/Resampling

A popular method for calculating particle weights is the *sequential importance sampling/resampling (SIS/R)*. Since sampling particles $\boldsymbol{x}_{1:t}^i$ from the true PDF $p(\boldsymbol{x}_{1:t}|\boldsymbol{y}_{1:t})$ is difficult (the true PDF is unknown), SIS/R utilizes a proposal distribution, or *importance density*, $q(\boldsymbol{x}_{1:t}|\boldsymbol{y}_{1:t})$ that has a known form and, therefore, is easier to sample from. The normalized particle weights represent the discrepancy between the true and the proposal distributions and can be recursively calculated as:

$$\begin{aligned} w_t^i &\propto \frac{p(\boldsymbol{x}_{1:t}|\boldsymbol{y}_{1:t})}{q(\boldsymbol{x}_{1:t}|\boldsymbol{y}_{1:t})} \\ &\propto \frac{p(\boldsymbol{x}_{1:t-1}|\boldsymbol{y}_{1:t-1})p(\boldsymbol{y}_t|\boldsymbol{x}_t^i)p(\boldsymbol{x}_t^i|\boldsymbol{x}_{t-1}^i)}{q(\boldsymbol{x}_{1:t-1}|\boldsymbol{y}_{1:t-1})q(\boldsymbol{x}_t^i|\boldsymbol{x}_{1:t-1}, \boldsymbol{y}_{1:t-1})} \\ &= w_{t-1}^i \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t^i)p(\boldsymbol{x}_t^i|\boldsymbol{x}_{t-1}^i)}{q(\boldsymbol{x}_t|\boldsymbol{x}_{1:t-1}, \boldsymbol{y}_{1:t-1})}. \end{aligned}$$

The particle weights are calculated by setting the importance density to the prior PDF as $q(\boldsymbol{x}_t|\boldsymbol{x}_{1:t-1}, \boldsymbol{y}_{1:t-1}) = p(\boldsymbol{x}_t^i|\boldsymbol{x}_{t-1}^i)$. After this substitution, the weight update is given by

$$w_t^i = w_{t-1}^i p(\boldsymbol{y}_t|\boldsymbol{x}_t^i).$$

Hence, the particle weights are recursively updated based on the likelihood of new observations.

3.4 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) is a class of algorithms that approximate posterior PDFs of random variables $\mathcal{X}$ through random sampling in large dimensional spaces. The most important characteristic of MCMC is its ability to draw samples from distributions even when the only knowledge about the distribution is how to calculate the density for different samples. MCMC is also very useful in Bayesian inference, since it focuses on posterior distributions that are difficult to examine analytically [139]. The most popular MCMC algorithm is the *Metropolis-Hastings* method.

3.4.1 Metropolis-Hastings MCMC Algorithm

Metropolis-Hastings (MH) algorithm was initially studied by Metropolis [140] and was later reestablished by Hastings [141]. Its strength lies in the ability to produce samples from distributions that are difficult to sample from, which can be useful in Bayesian inference. Assuming a target distribution $\pi(\cdot)$, MH simulates a Markov Chain that has $\pi(\cdot)$ as its stationary distribution. Thus, after a large number of iterations $t = 1, 2, ...$, samples from the Markov Chain will approximate samples from $\pi(\cdot)$.

For a set of random variables $\mathcal{X}$ that need to be estimated, MH starts at iteration $t = 0$ by randomly initializing them as $\mathcal{X}^0 \sim p(\mathcal{X})$, usually by sampling from their respective prior distributions. Then, the MH simulates candidate values $\mathcal{X}^*$

for random variables from a proposal distribution $q(\cdot)$, which is usually symmetric ($q(\mathcal{X}^*|\mathcal{X}^t) = q(\mathcal{X}^t|\mathcal{X}^*)$). Normal distribution represents the most popular choice of symmetric distribution, in which case MH is also known as *random-walk MH*. After simulating $\mathcal{X}^*$, the algorithm calculates the *acceptance probability* as:

$$\alpha = \min\left(1, \frac{\pi(\mathcal{X}^*)q(\mathcal{X}^*|\mathcal{X}^t)}{\pi(\mathcal{X}^t)q(\mathcal{X}^t|\mathcal{X}^*)}\right)$$

The algorithm accepts the candidate values $\mathcal{X}^*$ with probability α, that is $\mathcal{X}^{t+1} = \mathcal{X}^*$, otherwise $\mathcal{X}^{t+1} = \mathcal{X}^t$. The acceptance probability should be neither too high nor too low for achieving good mixing and accurate Markov chain convergence. Exhaustive research has proven that the ideal acceptance rate value should be between 20-40%.

3.5 Maximum Likelihood Estimation

Consider a $D \times T$ set of observations $\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., y_T\}$, where each $\boldsymbol{y}_t, t \in [1, 2, ..., T]$ represents a multidimensional observation vector at time t, $\boldsymbol{y}_t = \{y_t^1, y_t^2, ..., y_t^D\}$. Assuming that these observations are controlled by a set of parameters $\boldsymbol{\theta}$, the likelihood function for $\boldsymbol{\theta}$ is:

$$L(\boldsymbol{Y}|\boldsymbol{\theta}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{\theta}). \tag{3.3}$$

This function demonstrates the likelihood of observing $\boldsymbol{Y}$ given parameters $\boldsymbol{\theta}$. The *maximum likelihood estimation (MLE)* can then be used to infer the parameter values $\hat{\boldsymbol{\theta}}$ that maximize the likelihood function as:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, L(\boldsymbol{Y}|\boldsymbol{\theta}).$$

To avoid numerical underflow, it is recommended to maximize the logarithm of the likelihood, or *log-likelihood* function instead of Eq. 3.3 directly:

$$\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}) \equiv \log L(\boldsymbol{Y}|\boldsymbol{\theta}) = \sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\boldsymbol{\theta}). \tag{3.4}$$

Since $log(\cdot)$ is a strictly increasing function, the value of $\boldsymbol{\theta}$ which maximizes $p(\boldsymbol{y}_t|\boldsymbol{\theta})$ also maximizes $\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta})$.

3.6 Expectation-Maximization Algorithm

MLE function is a robust method for parameter estimation, but it requires fully observed data. When latent variables are considered, MLE computation can be achieved using an iterative process called the *Expectation-Maximization (EM)* algorithm. On each EM algorithm iteration, two steps, namely the E-step, and the M-step, are applied. During expectation, or the E-step, the latent variables (degradation states) are estimated given the observed data (sensor observations) and the current estimate of the model parameters. Then, in the M-step, the likelihood function is maximized given the latest estimates of the latent variables as obtained from the E-step.

The EM algorithm represents an iterative procedure that maximizes $\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta})$. Assuming that after ith iteration is complete, the current parameter vector is $\boldsymbol{\theta}_i$. In order to maximize likelihood, or the equivalent log-likelihood, function $\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta})$, the parameter vector $\boldsymbol{\theta}$ needs to be updated, such as

$$\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}) \geq \hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}_i).$$

Similarly, the difference

$$\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}) - \hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}_i)$$

must be maximized. Using a realization of the latent variables $\boldsymbol{x}$, total probability $p(\boldsymbol{y}_t|\boldsymbol{\theta})$ can be written as

$$p(\boldsymbol{y}_t|\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) p(\boldsymbol{x}|\boldsymbol{\theta}),$$

therefore

$$\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}) - \hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}_i) = \log \sum_{t=1}^{T} \sum_{\boldsymbol{x}} p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) p(\boldsymbol{x}|\boldsymbol{\theta}) - \sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\boldsymbol{\theta}_i). \tag{3.5}$$

Considering Jensen's inequality, Eq. 3.5 can be written as:

$$\begin{aligned}
\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}) - \hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}_i) &= \log \sum_{t=1}^{T} \Big(\sum_{\boldsymbol{x}} p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) p(\boldsymbol{x}|\boldsymbol{\theta}) - \log p(\boldsymbol{y}_t|\boldsymbol{\theta}_i) \Big) \\
&= \log \sum_{t=1}^{T} \Big(\sum_{\boldsymbol{x}} p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) p(\boldsymbol{x}|\boldsymbol{\theta}) \frac{p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i)}{p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i)} - \log p(\boldsymbol{y}_t|\boldsymbol{\theta}_i) \Big) \\
&= \log \sum_{t=1}^{T} \Big(\sum_{\boldsymbol{x}} p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i) \frac{p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) p(\boldsymbol{x}|\boldsymbol{\theta})}{p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i)} - \log p(\boldsymbol{y}_t|\boldsymbol{\theta}_i) \Big) \\
&\geq \sum_{t=1}^{T} \Big(\sum_{\boldsymbol{x}} p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i) \log \frac{p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) p(\boldsymbol{x}|\boldsymbol{\theta})}{p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i)} - \log p(\boldsymbol{y}_t|\boldsymbol{\theta}_i) \Big) \\
&= \sum_{t=1}^{T} \sum_{\boldsymbol{x}} p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i) \log \frac{p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) p(\boldsymbol{x}|\boldsymbol{\theta})}{p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i) p(\boldsymbol{y}_t|\boldsymbol{\theta}_i)} \\
&= \Delta(\boldsymbol{\theta}|\boldsymbol{\theta}_i).
\end{aligned} \tag{3.6}$$

In the same manner, Eq. 3.6 can be written as:

$$\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}) \geq \hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}_i) + \Delta(\boldsymbol{\theta}|\boldsymbol{\theta}_i) = l(\boldsymbol{\theta}|\boldsymbol{\theta}_i)$$

EM algorithm's objective is to select the parameter values $\boldsymbol{\theta}$ that maximize $\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta})$, or, equivalently, maximize $l(\boldsymbol{\theta}|\boldsymbol{\theta}_i)$, which is upper bounded by the log-likelihood function. The updated values are denoted as $\boldsymbol{\theta}_{i+1}$, for which the following equation holds:

$$\boldsymbol{\theta}_{i+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, l(\boldsymbol{\theta}|\boldsymbol{\theta}_i)$$

$$
\begin{aligned}
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \Big(\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}_i) + \Delta(\boldsymbol{\theta}|\boldsymbol{\theta}_i) \Big) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \Big(\hat{L}(\boldsymbol{Y}|\boldsymbol{\theta}_i) + \sum_{t=1}^{T} \sum_{\boldsymbol{x}} p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i) \log \frac{p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) p(\boldsymbol{x}|\boldsymbol{\theta})}{p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i) p(\boldsymbol{y}_t|\boldsymbol{\theta}_i)} \Big) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \Big(\sum_{t=1}^{T} \sum_{\boldsymbol{x}} p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i) \log \big[p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) p(\boldsymbol{x}|\boldsymbol{\theta}) \big] \Big) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \Big(\sum_{t=1}^{T} \sum_{\boldsymbol{x}} p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i) \log \big[\frac{p(\boldsymbol{y}_t, \boldsymbol{x}, \boldsymbol{\theta})}{p(\boldsymbol{x}, \boldsymbol{\theta})} \frac{p(\boldsymbol{x}, \boldsymbol{\theta})}{p(\boldsymbol{\theta})} \big] \Big) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \Big(\sum_{t=1}^{T} \sum_{\boldsymbol{x}} p(\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}_i) \log p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) \Big) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \Big(\sum_{t=1}^{T} \mathbb{E}_{\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}} \log p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta}) \Big).
\end{aligned}
$$

Eventually, the concept of EM algorithm can be summarized in the two following steps that occur iteratively:

1. Ascertain the sum of conditional probabilities $\sum_{t=1}^{T} \mathbb{E}_{\boldsymbol{x}|\boldsymbol{y}_t, \boldsymbol{\theta}} \log p(\boldsymbol{y}_t, \boldsymbol{x}|\boldsymbol{\theta})$.

2. Maximize previous expression w.r.t. $\boldsymbol{\theta}$.

3.7 Nelder-Mead Simplex Method

The Nelder-Mead simplex method was introduced in [142] and is a popular algorithm for multidimensional, unconstrained, and derivative-free optimization. Similarly to the famous Dantzig's simplex method, the Nelder-Mead algorithm is a special polytope of $n+1$ vertices in n dimensions. Assuming a function $f(x), x \in \mathbb{R}^n$ that needs to be minimized, and having $x_1, x_2, ..., x_{n+1}$ as our current test points, the Nelder-Mead algorithm proceeds as follows:

1. **Order function values according to vertices:**

   $f(x_1) \leq f(x_2) \leq ... \leq f(x_{n+1})$.

2. **Calculate** centroid x_o of all test points, except x_{n+1}.

3. **Reflection:** Set $\alpha > 0$ and calculate reflection point $x_r = x_o + \alpha(x_o - x_{n+1})$. if $f(x_1) \leq f(x_r) \leq f(x_{n+1})$, set $x_{n+1} = x_r$ and go to Step 1.

4. **Expansion:** If $f(x_r) < f(x_1)$, compute expand point $x_e = x_o + \gamma(x_r - x_o), \gamma > 1$. if $f(x_e) < f(x_r)$, set $x_{n+1} = x_e$ and go to Step 1; else set $x_{n+1} = x_o$ and go to Step 1.

5. **Contraction:** Calculate contracted point $x_c = x_o + \rho(x_{n+1} - x_o), 0 \leq \rho \leq 0.5$. if $f(x_c) < f(x_{n+1})$ then $x_{n+1} = x_c$, and go to Step 1.

6. **Shrink:** Except best point x_1, replace all points as: $x_i = x_1 + \sigma(x_i - x_1)$ and go to Step 1

A comprehensive study regarding the convergence of the Nelder-Mead algorithm can be found in [143], where its convergence properties for strictly convex one- and two-dimensional functions are presented.

3.8 Artificial Neural Networks

Artificial neural networks (ANNs), or *multi-layer perceptrons*, are supervised learning algorithms used for both classification and regression problems. They were initially inspired by the function of biological neural networks that exist in animal and human brains and use a network form of interconnected processing elements, or neurons. The advantage of ANNs is their ability to learn performance tasks without being pre-programmed by specific task-abiding rules. An ANN is a collection of artificial neurons distributed in sequential and densely connected layers: the input layer,

hidden layer(s), and output layer. The neurons in each layer collect input signals in the form of real numbers and produce outputs by utilizing an activation function over the sum of all inputs. Because the information transmission always goes onto the next level, ANNs are also known as *feed-forward* neural networks. The connections between layers are associated with weights that need to be tuned properly. This tuning process is known as ANN training that requires labeled training datasets.

3.8.1 Single-Layer Artificial Neural Networks

Assuming d-dimensional input signals $\boldsymbol{\xi} \equiv \xi_{1:d}$, and m-dimensional output variables $\boldsymbol{y} \equiv y_{1:m}$, a single-layer ANN is defined below for a single hidden layer and a single training sample. Using n hidden layer neurons, hidden layer outputs are computed as:

$$l_j = g(\sum_{d'=1}^{d} a_{jd'} \cdot \xi_{d'}), j = 1, 2, ..., n,$$

where $\{\mathrm{a}_{j1}, \mathrm{a}_{j2}, ..., \mathrm{a}_{jd}\}$ is the weight vector from the input layer to the jth hidden node. The second step relates the neurons between the hidden and the output layers as:

$$\hat{y}_{m'} = f\big(\sum_{j=1}^{n} \delta_{jm'}(g(\sum_{d'=1}^{d} a_{jd'} \cdot \xi_{d'}))\big),\ m' = 1, ..., m,$$

where vector $\boldsymbol{\delta}_j = [\delta_{1j}, ...\delta_{mj}]$ represents the output weights for hidden neuron j. The weights are randomly initialized at the beginning of ANN training. Functions $g(\cdot), f(\cdot)$ are activation functions that can be either linear or non-linear. The most common types of activation functions are the sigmoid and the radial basis functions, although other types, such as the rectified linear unit (ReLU), have recently increased in popularity. The outputs of all neurons are propagated to the next level, where the same process repeats for the neurons in this layer until the overall network outputs

are produced in the end. This process is known as forward pass. Assuming n training samples with respective actual labels $\boldsymbol{y}_n = y_{mn}$, and denoting the estimated outputs from the forward pass as $\hat{\boldsymbol{y}}_n = \hat{y}_{mn}$, the sum of squared error between the target, and estimated outputs is computed as:

$$SSE = \sum_{k=1}^{n} \sum_{m'=1}^{m} (y_{m'k} - \hat{y}_{m'k})^2.$$

SSE represents a *cost function* that measures the dissimilarity between the two values. The purpose of the training process is the minimization of SSE by following an iterative gradient decent procedure known as *backpropagation algorithm* ([144]). Neuron weights are adjusted with every iteration in the direction of the SSEs decreasing values. The weights are updated as:

$$\boldsymbol{w}(i+1) = \boldsymbol{w}(i) + \rho \nabla(i), i \in \{1, 2, ...\}$$

where $\boldsymbol{w}$ is a generic weight vector, i is the iteration number, ρ is the learning rate and $\nabla(i)$ is the gradient vector of the SSE function. Backpropagation repeats for a predefined number of iterations (epochs) until the error converges to a low value or the maximum number of epochs is reached. ANNs are considered *universal approximators*, meaning that they can theoretically approximate any non-linear continuous function, given a finite number of hidden neurons under mild assumptions on the activation function [145]. Fig. 4 presents an outline of an ANN. Despite their efficiency to approximate functions of any complexity, ANNs face potential issues during training that can potentially lead to overfit training data, thus demonstrating poor performance when new, unlabeled, data are obtained. The main reasons for overfitting include when little training data is provided or if the number of hidden neurons is too large. Therefore, multiple trial-and-error steps are necessary to find an efficient design of an ANN.

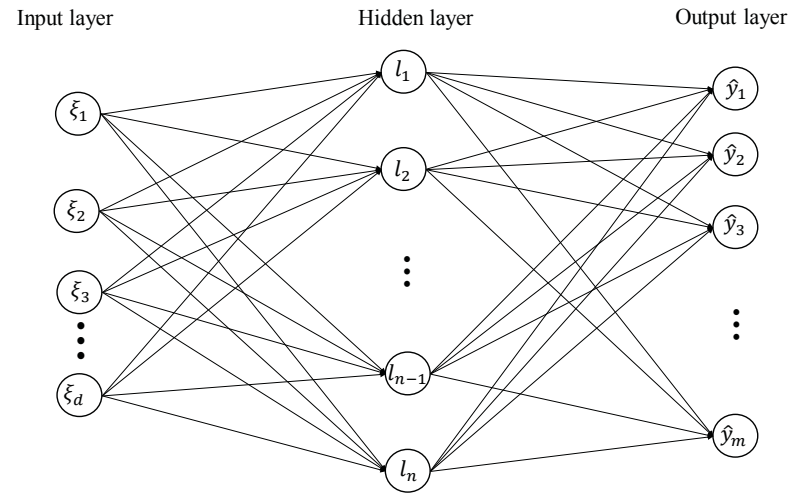


Figure 4: Outline of an ANN with d inputs, n hidden neurons, and m outputs.

The most common ANN topology requires one hidden layer, which is known as a *shallow network*. Conversely, deep learning, as the name suggests, requires neural networks with multiple hidden layers. In this case, the ANN becomes a *deep neural network*.

3.8.2 Deep Neural Network

Fully connected deep neural networks (DNNs) are similar to ANNs, except that they have multiple hidden layers that give DNNs a greater capacity for approximating highly complex functions. Hidden layers can have different numbers of neurons, a common practice in deep learning for better generalization and less overfitting. However, this can possibly lead to underfitting since, by dropping neurons, a part of the information stored in the training data is discarded. Furthermore, more hidden layers contain high numbers of hidden neurons, and therefore more free variables (weights) that need to be tuned through back-propagation. This architecture can lead to higher computational solution times, or other significant issues, such as *vanishing gradients*

[130]. Advancements in computer hardware and faster algorithms, such as long short-term memory [146], are capable of efficiently overcoming these issues. Fig. 5 presents an outline of a DNN with two hidden layers.

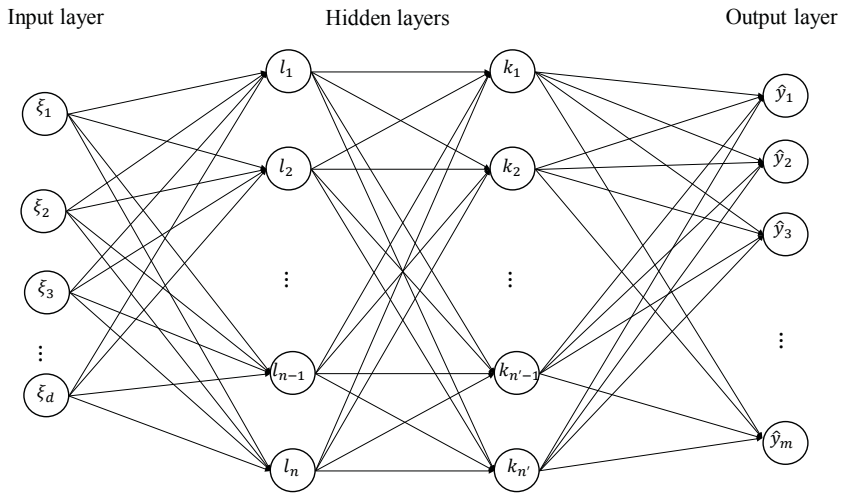


Figure 5: Outline of a deep neural network with 2 hidden layers.

3.8.3 Recurrent Neural Network

Recurrent neural networks (RNNs) are deep neural networks specifically designed for processing sequential data. Connections between units form directed sequential graphs that allow them to display temporal dynamic behavior. What sets apart RNNs from the other neural network architectures is that the latter can map given inputs to target outputs using only forward propagation. In contrast, RNNs can map the entire input history to the target outputs by storing the memory of all these inputs within the network [147].

A plethora of RNN architectures have been introduced in the literature. In its basic form, an RNN takes as input a sequence of data points and, at each time step,

applies a simple RNN unit to both a single data point and to the networks output from the previous time step [148]. Given an input in the form of $\boldsymbol{\xi}^1_{1:d}, ..., \boldsymbol{\xi}^\tau_{1:d}$, with $\boldsymbol{\xi}^{1:\tau}_{1:d}$ denoting a $d \times \tau$ tensor representing d-dimensional input sequences at times $t = 1, .., \tau$, a simple RNN applies an activation function $f_h(\cdot)$ iteratively. This process creates a latent level $\mathbf{h}_t$:

$$\boldsymbol{h}_t = f_h(\boldsymbol{\xi}_t, \boldsymbol{h}_{t-1}) = \sigma(\boldsymbol{\xi}_t \cdot \mathbf{W}_h + \boldsymbol{h}_{t-1} \cdot \boldsymbol{U}_h + b_h).$$

$\boldsymbol{W}_h, \boldsymbol{U}_h$ are neuron weight matrices associated with the inputs at time t and the feedback outputs from time $t-1$, respectively, b_h is the bias, and $\sigma(\cdot)$ represents the logistic sigmoid activation function. The output at time t can then be defined as:

$$\hat{\boldsymbol{y}}_t = f_t(\boldsymbol{h}_t \cdot \boldsymbol{W}_t + b_t),$$

where again $\boldsymbol{W}_y$ denotes a neuron weight matrix associated with the outputs at time t and b_y the output bias. Fig. 6 presents the layout of a vanilla RNN. Note the "unfolding" of the hidden layers, which illustrates the function of recurrence within RNNs.

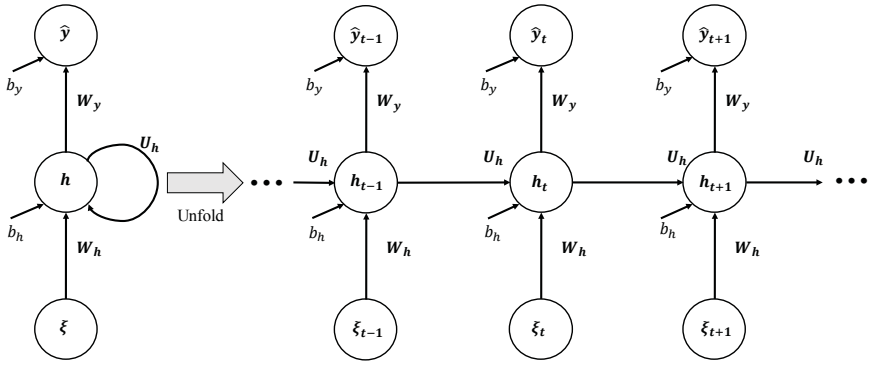


Figure 6: Outline of a recurrent neural network.

3.8.4 Extreme Learning Machine

Backpropagation generally requires a large number of training epochs to accurately tune the neuron weights, leading to long training times - even for neural networks of moderate sizes. The Extreme Learning Machine (ELM) was introduced in [149] with the purpose of addressing this training time issue. ELM has the same structure as a hidden single-layer feed-forward neural network, though its learning speed is significantly higher due to the fact that the neuron weights between inputs and the hidden layer are randomly assigned. Therefore, they are independent of the training data. As a result, input-to-hidden layer connection weights do not need tuning, leading to much faster training speeds.

It must be noted here that, while connection weights between the input and the hidden layers are randomly set, those between the hidden and the output layers are tuned in with past data through activation functions. ELM can be trained based on the same data multiple times with different random input weights and still obtain the values that give the lowest training error. The application of ELMs is given in Chapter 5.

3.9 Reinforcement Learning

Reinforcement learning (RL) can be considered to belong to the area of machine learning, although it can also be considered a separate statistical learning approach alongside supervised and unsupervised learning. The debate for this distinction is still ongoing. The objective of RL is to let an autonomous software *agent*, that acts within an *environment*, perform a certain set of *actions* that maximize some

form of a cumulative reward. RL algorithms are formulated as *Markov decision processes* (MDPs), since the feedback provided from the environment satisfies the *Markov property*. MDPs are described by a 5-tuple $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{X} = \{\boldsymbol{x}_t\}$ is the set of states, $\mathcal{A} = \{\alpha^1, ..., \alpha^Z\}$ is the set of Z different control actions, $\mathcal{P}$ represents the probability transition equation $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \alpha_t)$, $\mathcal{R}(\boldsymbol{x}_t, \alpha_t)$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor emphasizing on immediate rewards. A generic representation of the RL is presented in Fig. 7, with the blue arrow denoting the action taken by the agent. The green and brown arrows denote the next environment state and the reward obtained from the performed action that are given as feedback to the agent.

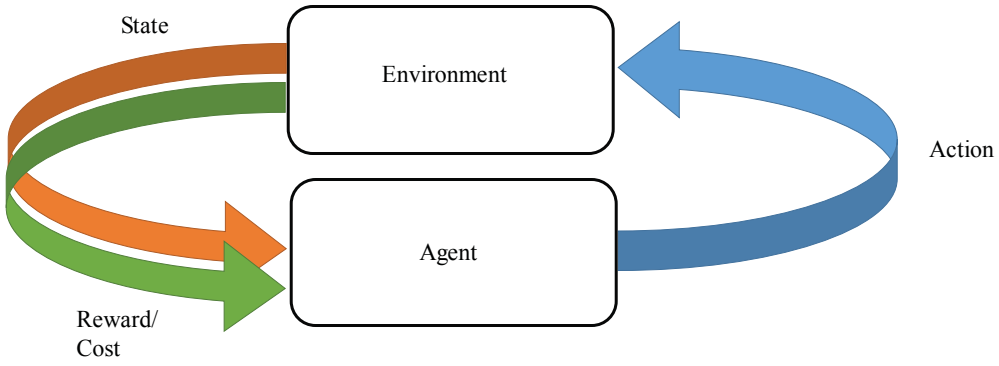


Figure 7: Reinforcement learning process.

Agent training is iterative with agent-environment interactions occurring at discrete time intervals. At interval t, the agent receives state $\boldsymbol{x}_t$ from the environment and chooses a control action α_t. The system reacts to the action and traverses to the next state $\boldsymbol{x}_{t+1}$, where it observes a reward r_{t+1}. Both $\boldsymbol{x}_{t+1}$ and r_{t+1} are provided to the agent as feedbacks from the environment. Rewards are associated with agent transitions and are determined *a priori*. At each training iteration, the agent updates either the *value function* $\mathcal{V}(\boldsymbol{x})$ or the *action-value function* $\mathcal{Q}(\boldsymbol{x}, \alpha)$ based

on a specific *policy* that maps states $\boldsymbol{x} \in \mathcal{X}$ to actions $\alpha \in \mathcal{A}$ [150]. When a value function is considered, the expected outcome of being at state $\boldsymbol{x}$ following policy π is the cumulative reward for visiting all future states, discounted by γ:

$$\mathcal{V}^{\pi}(\boldsymbol{x}) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{T} \gamma^{k} r_{t+k} | \boldsymbol{x}_t = \boldsymbol{x} \right). \tag{3.7}$$

Accordingly, when an action-state-value function is considered, the expected outcome of being at state $\boldsymbol{x}$, taking action α following policy π, is the cumulative discounted reward for all future action-state-value pairs:

$$\mathcal{Q}^{\pi}(\boldsymbol{x}, \alpha) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{T} \gamma^{k} r_{t+k} | \boldsymbol{x}_t = \boldsymbol{x}, \alpha_t = \alpha \right). \tag{3.8}$$

Since the environment is formulated as a MDP, RL can be solved using dynamic programming, using either i) a *policy iteration*, where the process starts from an initial policy and improves iteratively, or ii) a *value iteration*, where the process, starting from an arbitrary value function, improves the estimated values of a state- or action-state-value function leading to an optimal policy. Using the *Bellman equation* [151], the state-value function can be estimated as:

$$\mathcal{V}^{\pi}(\boldsymbol{x}) = \mathbb{E}_{\pi}\left(r_t + \gamma \mathcal{V}^{\pi}(\boldsymbol{x}_{t+1}) | \boldsymbol{x}_t = \boldsymbol{x} \right). \tag{3.9}$$

The optimal policy π^* is obtained by greedily implementing actions that maximize the state-value function. In a similar manner, when an action-state-value function is considered, π^* is obtained through

$$\mathcal{Q}^{\pi}(\boldsymbol{x}, \alpha) = \mathbb{E}_{\pi}\left(r_t + \gamma \mathcal{Q}^{\pi}(\boldsymbol{x}_{t+1}, \alpha_{t+1}) | \boldsymbol{x}_t = \boldsymbol{x}, \alpha_t = \alpha \right). \tag{3.10}$$

Several RL solution techniques have been introduced in the literature, such as Monte Carlo, temporal-difference (TD) and station-action-reward-state-action (SARSA) [152].

The development of the off-policy $\mathcal{Q}$-*learning* method provided a breakthrough in RL, since with $\mathcal{Q}$-learning, the action-state-value function directly approximates its optimal value regardless of the policy followed. This concept provides a significant simplification of the algorithm analysis and provides early convergence [152].

In $\mathcal{Q}$-learning, all value function values are stored in a predefined matrix ($\mathcal{Q}$-table), where each cell represents all possible states of the environment. However, storing separate value functions for every possible state is inefficient and computationally intractable in real-world problems where very large state-spaces are considered. Generalization through *function approximators*, such as ANNs and DNNs, has proven to be effective in obtaining the optimal policy. The combination of DNNs and RL led to the development of *deep reinforcement learning (DRL)* algorithms. A notable example can be found in [132], where the *deep* $\mathcal{Q}$*-learning* technique was established. A neural network as an approximator for the action-value function can be utilized as:

$$\hat{\mathcal{Q}}(\boldsymbol{x}, \alpha; \boldsymbol{\theta}) \approx \mathcal{Q}(\boldsymbol{x}, \alpha),$$

where $\boldsymbol{\theta}$ denotes the neuron weights. This approximation, known as $\mathcal{Q}$*-network* [153], improves the action-value function through minimizing a series of loss functions

$$\mathcal{L}(\boldsymbol{\theta}_j) = \mathbb{E}_{\boldsymbol{x},\alpha}\Big[(\mathcal{Y}_j - \hat{\mathcal{Q}}(\boldsymbol{x}, \alpha; \boldsymbol{\theta}_j))^2\Big], \tag{3.11}$$

where $\mathcal{Y}_j = \mathbb{E}_{\boldsymbol{x}}\big[r_j + \gamma \max_{\alpha'} \hat{\mathcal{Q}}(\boldsymbol{x}', \alpha'; \boldsymbol{\theta_{j-1}})|\boldsymbol{x}, \alpha\big]$ is the network target of iteration j. These targets directly depend on the neuron weights and are not predefined, as in the case of supervised learning. In the deep reinforcement learning approach, targets are also known as *target networks*. Through gradient descent, Eq. 3.11 is minimized as:

$$\nabla_{\boldsymbol{\theta}_j}\mathcal{L}(\boldsymbol{\theta}_j) = \mathbb{E}_{\boldsymbol{x},\alpha}\Big[\Big(r_j + \gamma \max_{\alpha'} \hat{\mathcal{Q}}(\boldsymbol{x}', \alpha'; \boldsymbol{\theta_{j-1}}) - \hat{\mathcal{Q}}(\boldsymbol{x}, \alpha; \boldsymbol{\theta}_j)\Big)\nabla_{\boldsymbol{\theta}_j}\mathcal{Q}(\boldsymbol{x}, \alpha; \boldsymbol{\theta}_j)\Big]. \tag{3.12}$$

When deep neural networks are used, numerous parameters ($\boldsymbol{\theta}$) must be tuned, and therefore stochastic gradient descent is an efficient alternative method for loss minimization. For notational convenience, $\mathcal{Q}$ instead of $\hat{\mathcal{Q}}$ is used to refer to the action-value function.

CHAPTER 4

Monitoring and Control of Systems with Linear and Gaussian Dynamics

Most modern industrial systems are controlled by complex, non-linear dynamics. There are, however, many systems that are controlled by simpler linear dynamics, such as batteries and their state of charge. The framework in this Chapter provides an introductory model that operates under specific assumptions concerning system dynamics, such as linear degradation and observation processes contaminated with Gaussian noise. This model is not a part of the overall decision-making structure that is proposed in this thesis, but forms the basis for frameworks that can describe more complex systems (see Chapter 5).

4.1 The Main Model

A dynamic system at time t is characterized by a latent degradation process x_t and an observation process $\boldsymbol{y}_t$. A hazard process λ_t, representing the conditional probability of failure over time as a function of the latent degradation state x_t, can also be considered. Therefore, the overall working status o_t of the system at time t is defined as a random variable with a distribution dependent on λ_t. The proposed system dynamics structure, presented in Figure 8, consists of a latent degradation state x_t,

a quasi-latent hazard state λ_t, the sensor observations process $\boldsymbol{y}_t$, and the observable working condition o_t. Both processes $\boldsymbol{y}_t$ and o_t are conditionally independent given x_t. The design of the structure helps with inferring latent states by utilizing observable data collected over time. Consequently, before proceeding to the actual reliability analysis, the overall model needs to be trained using already obtained historical data. This process is described in Section 4.1.1.

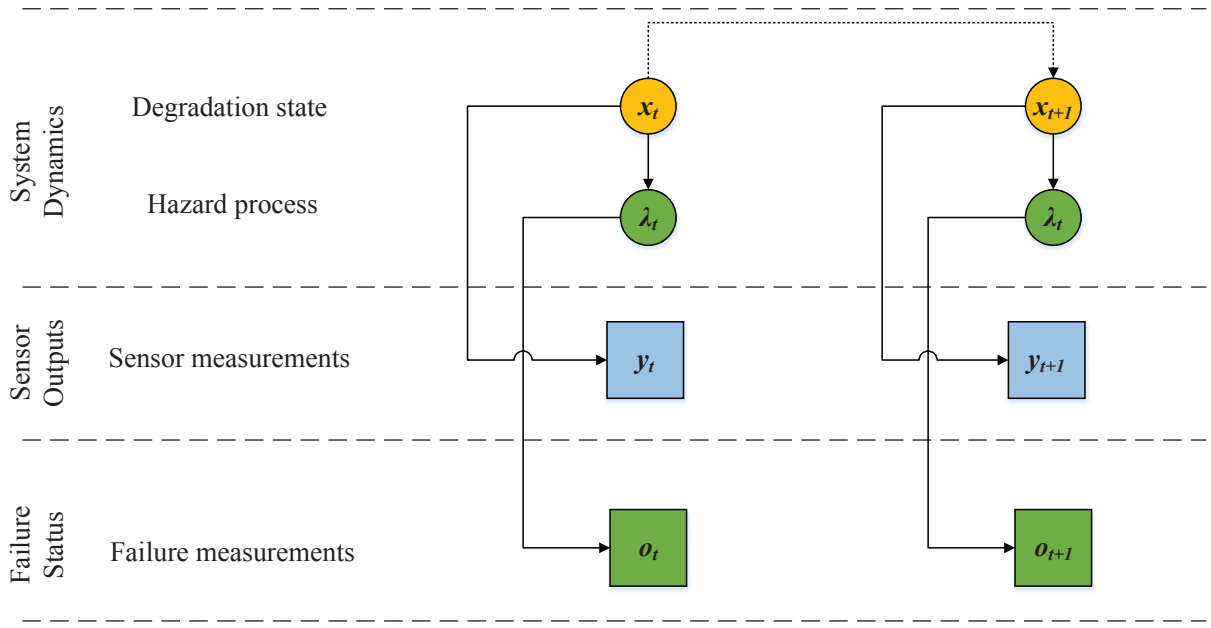


Figure 8: The stochastic framework of the proposed structure.

4.1.1 Inference on the Approximate Closed-form solution of the Marginal Likelihood Function

Assuming a single degrading system with lifetime T, sensor observations $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_T$, and working condition observations $o_1, \ldots, o_T$, the marginal likelihood can be applied for parameter estimation of the proposed structure. The parameter vector is denoted

by $\boldsymbol{\theta}$, where $\boldsymbol{\theta} = \{\boldsymbol{F}_t, \boldsymbol{H}_t, \boldsymbol{Q}_t, \boldsymbol{R}_t, \alpha, \beta, \beta_0\}$. The likelihood function can be expressed as follows:

$$p(\boldsymbol{o}, \boldsymbol{y}|\boldsymbol{\theta}) = p(\boldsymbol{o}|\boldsymbol{y}, \boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{\theta}). \tag{4.1}$$

For brevity, $\boldsymbol{\theta}$ will be omitted. The degradation state is assumed to be one-dimensional, and therefore the posterior state estimate $\hat{\boldsymbol{x}}_{t|t}$ and posterior error covariance $\boldsymbol{P}_{t|t}$ are replaced by their scalar equivalents $\hat{x}_{t|t}$ and $P_{t|t}$, respectively. Probability $p(\boldsymbol{y})$ denotes the marginal likelihood of Kalman filter, which can be factored as the product of probability of each observation given previous observations as:

$$\begin{aligned} p(\boldsymbol{y} = \boldsymbol{y}_0, \cdots, \boldsymbol{y}_T) &= \prod_{t=0}^{T} p(\boldsymbol{y}_t|\boldsymbol{y}_{t-1}, ..., \boldsymbol{y}_0) \\ &= \prod_{t=0}^{T} \int p(\boldsymbol{y}_t|x_t)p(x_t|\boldsymbol{y}_{t-1}, ..., \boldsymbol{y}_0)dx_t \\ &= \prod_{t=0}^{T} \int \mathcal{N}(\boldsymbol{y}_t; \boldsymbol{H}_t x_t, \boldsymbol{R}_t) \cdot \mathcal{N}(x_t; \hat{x}_{t|t-1}, P_{t|t-1})dx_t \\ &= \prod_{t=0}^{T} \mathcal{N}(\boldsymbol{y}_t; \boldsymbol{H}_t \hat{x}_{t|t-1}, \boldsymbol{S}_t). \end{aligned} \tag{4.2}$$

Probability $p(\boldsymbol{o}|\boldsymbol{y})$ can also be simplified as:

$$p(\boldsymbol{o}|\boldsymbol{y}) = p(o_0|\boldsymbol{y}_0) \cdot \prod_{t=1}^{T} p(o_t|\boldsymbol{y}_{1:t}). \tag{4.3}$$

Based on the fact that $\boldsymbol{y}_t$ and o_t are conditionally independent given x_t, then

$$p(o_0|\boldsymbol{y}_0) \cdot \prod_{t=1}^{T} p(o_t|\boldsymbol{y}_{1:t}) = \int p(o_0|x_0)p(x_0|\boldsymbol{y}_0)dx_0 \cdot \prod_{t=1}^{T} \int p(o_t|x_t)p(x_t|\bar{\boldsymbol{y}}_t)dx_t, \tag{4.4}$$

where $\bar{\boldsymbol{y}}_t \equiv \boldsymbol{y}_{1:t}$. The probability of the system functioning at time t, given the hidden state x_t, is denoted by $p(o_t|x_t)$, whereas term $p(x_t|\boldsymbol{y}_t)$ represents the PDF of the hidden state x_t, given the observations up to this point. Analytically solving Eq. 4.4 can be computationally challenging and potentially impossible to complete in a

reasonable amount of time, and therefore a closed-form solution is preferable using the nice properties of the *Probit* function. Term $p(o_t|x_t), \forall t = \{0, 1, .., T\}$ represents the logistic regression function described in Section 3.2, but since the convolution of logistic sigmoid and Gaussian distribution is intractable [154], the Probit function can be used to approximate $p(o_0|x_0)$ and $p(o_t|x_t)$. Thus, term $p(o_t|x_t), \forall t = \{0, 1, .., T\}$ can be written in a Probit form as:

$$\sigma(\tau) \approx \Phi(\xi\tau).$$

Coefficient ξ ($\xi = \sqrt{\frac{\pi}{8}}$) guarantees the same slope at origin for the two functions. $\Phi(\tau)$ represents the cumulative distribution function of the standard Gaussian distribution and is given in the following equation:

$$\Phi(\tau) = \int_{-\infty}^{\tau} \mathcal{N}(x|0, 1)dx.$$

Since system dynamics are linear and Gaussian, the second terms of both integrals of Eq. 4.4 can be expressed as

$$p(x_0|y_0) = \mathcal{N}(x_0|\hat{x}_{0|0}, P_{0|0}), \tag{4.5}$$

$$p(x_t|\bar{\boldsymbol{y}}_t) = \mathcal{N}(x_t|\hat{x}_{t|t}, P_{t|t}). \tag{4.6}$$

Regarding the Probit and the Gaussian distribution, the following relationship holds true [155, 156]:

$$\int \Phi(\xi\tau)\mathcal{N}(\tau|\mu, \sigma^2)d\tau = \Phi(\frac{\mu}{\sqrt{\xi^{-2} + \sigma^2}}) \approx \sigma\Big(\frac{\mu}{\sqrt{1 + \xi^2\sigma^2}}\Big), \tag{4.7}$$

where $\sigma(\frac{\mu}{\sqrt{1+\xi^2\sigma^2}})$ is the logistic sigmoid, in a form that can approximate the value of the Probit. Using the known properties of adding and multiplying constants to a

random Gaussian variable, the expressions for mean and variance can be updated. The results of transforming the integrals in Eq. 4.4 are shown below:

$$\begin{aligned}\int p(o_0|x_0)p(x_0|\boldsymbol{y}_0)dx_0 &= \int \Phi(\xi(\alpha x_0+\beta\cdot 0+\beta_0))\,\mathcal{N}(x_0|\hat{x}_{0|0},P_{0|0})dx_0\\ &= \int \Phi(\xi(\alpha x_0+\beta_0))\mathcal{N}(\xi(\alpha x_0+\beta_0)|\alpha\hat{x}_{0|0}+\beta_0,\alpha^2P_{0|0})dx_0\\ &= \sigma\Big(\frac{\alpha\hat{x}_{0|0}+\beta_0}{\sqrt{1+\xi^2\alpha^2P_{0|0}}}\Big),\end{aligned} \tag{4.8}$$

$$\begin{aligned}\prod_{t=1}^{T}\int p(o_t|x_t)p(x_t|\bar{\boldsymbol{y}}_t)dx_t &= \prod_{k=1}^{T}\int \Phi(\xi(\alpha x_t+\beta\cdot t+\beta_0))\mathcal{N}(x_t|\hat{x}_{t|t},P_{t|t})dx_t\\ &= \prod_{t=1}^{T}\int \Phi(\xi(\alpha x_t+\beta\cdot t+\beta_0))\cdot\mathcal{N}(\xi(\alpha x_t+\beta\cdot t+\beta_0)|\alpha\hat{x}_{t|t}+\beta\cdot t+\beta_0,a^2P_{t|t})dx_t\\ &= \prod_{t=1}^{T}\sigma\Big((-1)^{\mathbf{1}_{\{t=T\}}}\frac{\alpha\hat{x}_{t|t}+\beta\cdot t+\beta_0}{\sqrt{1+\xi^2\alpha^2P_{t|t}}}\Big),\end{aligned} \tag{4.9}$$

where $\mathbf{1}_{\{*\}}$ denotes the indicator function. Substituting all the aforementioned expressions, Eq. 4.1 can be rewritten as

$$p(\boldsymbol{o},\boldsymbol{y}) = \sigma\Big(\frac{\alpha\hat{x}_{0|0}+\beta_0}{\sqrt{1+\xi^2\alpha^2P_{0|0}}}\Big)\cdot\prod_{t=1}^{T}\sigma\Big((-1)^{\mathbf{1}_{\{t=T\}}}\frac{\alpha\hat{x}_{t|t}+\beta\cdot t+\beta_0}{\sqrt{1+\xi^2\alpha^2P_{t|t}}}\Big)\cdot\prod_{t=0}^{T}\mathcal{N}(\boldsymbol{y}_t;\boldsymbol{H}_t\hat{x}_{t|t-1},\boldsymbol{S}_t). \tag{4.10}$$

Eq. 4.10 illustrates the closed-form solution of the marginal likelihood function from Eq. 4.1, and can be used for parameter estimation. Due to possible occurences of arithmetic underflow (see Section 3.5), the logarithm version of Eq. 4.10, $\ell = \log(p(\boldsymbol{o},\boldsymbol{y}|\boldsymbol{\theta}))$, is used instead. The complete marginal likelihood estimation function is presented below.

$$\begin{aligned}\mathcal{L} = \sum_{t=0}^{T}\Bigg\{&\mathbf{1}_{\{t=0\}}\log\Bigg\{\sigma\Big(\frac{\alpha\hat{x}_{0|0}+\beta_0}{\sqrt{1+\xi^2\alpha^2P_{0|0}}}\Big)\Bigg\}\\ &+\mathbf{1}_{\{t>0\}}\log\Bigg\{\sigma\Big((-1)^{\mathbf{1}_{\{t=T\}}}\frac{\alpha\hat{x}_{t|t}+\beta\cdot t+\beta_0}{\sqrt{1+\xi^2\alpha^2P_{t|t}}}\Big)\Bigg\}\end{aligned}$$

$$+ \log \left\{ \mathcal{N}(\boldsymbol{y}_t; \boldsymbol{H}_t \hat{x}_{t|t-1}, \boldsymbol{S}_t) \right\} \Bigg\}. \tag{4.11}$$

The unknown parameters can be estimated by maximizing 4.11 over $\boldsymbol{\theta}$. Given M degrading systems with an equal number of observation sequences, the complete marginal likelihood estimation function becomes

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{j=1}^{M} \mathcal{L}_j(\boldsymbol{\theta}).$$

4.2 Predictive Analytics

Three data-driven measures that provide useful information on the current and future health status of the device under monitoring are described here. These measures can be divided into diagnostic measures, which deal with the current health status of the device, and prognostic measures, which deal with the future health status of the device, and are: conditional reliability, degradation level, and remaining useful life.

4.2.1 Reliability Measures

One of the most important measures in prognostics is the conditional reliability, which represents the probability that a device continues its operation until a future time point t', given that the device has not failed at time t, $(t' \geq t + 1)$, and $\bar{\boldsymbol{y}}_t$ is observed. Future reliability estimation must occur, assuming that no observation signals are observable after time t. An analytical solution for the distribution of the degradation state at time t' by propagating the mean and the variance of the state in all future cycles before t' is provided. Based on Kalman filter recursive equations,

the state estimates $\hat{x}_{t'|t}$ and covariance estimates $P_{t'|t}$ can be computed as

$$\hat{x}_{t'|t} = \prod_{i=t+1}^{t'} F_i \hat{x}_{t|t}, t' > t, \tag{4.12}$$

$$P_{t'|t} = (\prod_{i=t+1}^{t'} F_i)^2 P_{t|t} + \sum_{j=t+1}^{t} (\prod_{i=j+1}^{t'} F_i)^2 Q_j, t' > t. \tag{4.13}$$

Since one-dimensional degradation state is considered, matrices $\boldsymbol{F}_t$ and $\boldsymbol{Q}_t$ take a scalar form F_t and Q_t, respectively. The probability of the system being at state $x_{t'|t}$ at t', given observations up to time t and assuming the system is still operational at time t, is

$$p(x_{t'|t}|\bar{\boldsymbol{y}}_t) = \mathcal{N}(x_{t'|t}; \hat{x}_{t'|t}, P_{t'|t}), t' > t.$$

Three important measures are defined:

1. $h_{t'|t}$: conditional probability of failure at the t'th cycle, given the survival up to cycle $t'-1$ and observations $\bar{\boldsymbol{y}}_t$

$$\begin{aligned} h_{t'|t} = p(o_{t'} = 1|\bar{o}_{t'-1} = 0, \bar{\boldsymbol{y}}_t) &= \int \Phi(\xi(\alpha x_t + \beta \cdot t + \beta_0))\mathcal{N}(x_t|\hat{x}_{t'|t}, P_{t'|t})dx_t \\ &= \sigma\Big(-\frac{\alpha\hat{x}_{t'|t} + \beta \cdot t + \beta_0}{\sqrt{1+\xi^2\alpha^2 P_{t'|t}}}\Big). \end{aligned} \tag{4.14}$$

2. $f_{t'|t}$: conditional probability distribution function of the failure at the t'th cycle, given observations $\bar{\boldsymbol{y}}_t$

$$f_{t'|t} = p(o_t = 1, \bar{o}_{t'-1} = 0|\bar{o}_t = 0, \bar{\boldsymbol{y}}_t). \tag{4.15}$$

3. $r_{t'|t}$: probability of being in working state at the t'th cycle, given $\bar{\boldsymbol{y}}_t$

$$r_{t'|t} = p(\bar{o}_{t'} = 0|\bar{o}_t = 0, \bar{\boldsymbol{y}}_t). \tag{4.16}$$

Term $\bar{o}_t = 0$ denotes the survival of the system up to time t. The following recursive relationships hold true between f, h, and r as

$$f_{t'|t} = r_{t'-1|t} \times h_{t'|t},$$

$$r_{t'|t} = 1 - \sum_{g=t+1}^{t'} f_{g|t}.$$

Algorithm 1 simplifies the process of deriving $r_{t'|t}$ for a single observation sequence $\bar{\boldsymbol{y}}_t$.

Algorithm 1 Algorithm to calculate $r_{t'|t}$, $f_{t'|t}$, and $h_{t'|t}$.

procedure RELIABILITY
  Input: Sequence of observations $\bar{\boldsymbol{y}}_t$, parameters $F_t, \boldsymbol{H}_t, Q_t, \boldsymbol{R}_t, \alpha, \beta, \beta_0$
  for$(g \leftarrow t+1 : t')$ **repeat**:
    $h_{g|t} = \sigma(-\frac{\alpha\hat{x}_{g|t}+\beta\cdot g+\beta_0}{\sqrt{1+\xi^2\alpha^2 P_{g|t}}})$
  end_for
  Initialize: $f_{t+1|t} \leftarrow h_{t+1|t}$, given that $r_{t|t} \leftarrow 1 - h_{t+1|t}$
  for$(t'_0 \leftarrow t+2 : t')$ **repeat**:
    $r_{t'_0-1|t} = 1 - \sum_{g=t+1}^{t'_0} f_{g|t}$
    $f_{t'_0|t} = r_{t'_0-1|t} \times h_{t'_0|t}$
  end_for
  $r_{t'|t} = 1 - \sum_{g=t+1}^{t'} f_{g|t}$, $f_{t'|t} = r_{t'-1|t} \times h_{t'|t}$, $h_{t'|t} = \sigma\Big(-\frac{\alpha\hat{x}_{t'|t}+\beta\cdot t+\beta_0}{\sqrt{1+\xi^2\alpha^2 P_{t'|t}}}\Big)$.
  Output: $r_{t'|t}$, $f_{t'|t}$, $h_{t'|t}$
end procedure

4.2.2 Degradation Level

The monotonicity of the degradation process has always been a challenging topic in real-world condition monitoring applications. The distribution of $\hat{x}_t$ at time t, given observations $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t$ is shown in Eq. 4.6. This measure can be utilized for latent degradation state monitoring, but it is possible that due to its high variance, it will be ineffective for real-time condition monitoring. Aside from those measurements,

the average overall health status at time t given $\bar{\boldsymbol{y}}_t$ can also be defined as follows:

$$\begin{aligned} A_t &= 0 \times p(o_t = 0|\bar{\boldsymbol{y}}_t, \bar{o}_{t-1} = 0) + 1 \times p(o_t = 1|\bar{\boldsymbol{y}}_t, \bar{o}_{t-1} = 0) \\ &= p(o_t = 1|\bar{\boldsymbol{y}}_t, \bar{o}_{t-1} = 0) \\ &= h_{t|t} \end{aligned}$$

that can be simplified to

$$p(o_t = 1|\bar{\boldsymbol{y}}_t, \bar{o}_{t-1} = 0) = \int p(o_t = 1|x_t)p(x_t|\bar{\boldsymbol{y}}_t, \bar{o}_{t-1} = 0)dx_t = \sigma(-\frac{\alpha\hat{x}_{t|t} + \beta \cdot t + \beta_0}{\sqrt{1 + \xi^2\alpha^2 P_{t|t}}}).$$

All the aforementioned measures belong to the class of diagnostics, as they provide information regarding only the current status of the system's health.

4.2.3 Remaining Useful Life

The conditional RUL is defined as a random variable that represents the number of cycles until the failure point is reached, given past observations. Let RUL_t denote the conditional remaining life computed at time t, as:

$$p(\text{RUL}_t = d) = p(o_{t+d} = 1, \bar{o}_{t+d-1} = 0|\bar{\boldsymbol{y}}_t, \bar{o}_{t-1} = 0) = f_{t+d|t}, \tag{4.17}$$

where $d > 0$. Based on the above distribution, the mean residual life MRL_t, or expected remaining useful life of the system given its survival up to time t, can be defined from the following equation:

$$\text{MRL}_t = \mathbb{E}(\text{RUL}_t) = \sum_{d=0}^{\infty} d \times f_{t+d|t}. \tag{4.18}$$

4.2.4 Real-Time Decision Making

A cost-effective replacement policy with regards to the system degradation structure is presented here. Such policies generally require two kinds of costs, namely a

cost of replacement c_r and a cost of failure c_f. As the names suggest, the former is set exclusively as the cost of the equipment that is about to fail (or have failed already) and it is to be removed and replaced by a new unit. In other words, c_r is considered as the cost of purchasing the new equipment. On the other hand, the failure cost adds to c_r all relevant charges such as extra labor, repair of damage to the rest of the system, cost of downtime, etc. Therefore, it is safe to assume that $c_f > c_r$, which is reasonable for real-world systems. System sensors provide condition monitoring signals at discrete time points, or *observation epochs*. Let l be the true lifetime of the system, L be the random variable representing the lifetime, and T_γ be the replacement time associated with a policy γ. Based on this policy, the system is replaced at T_γ or at failure, whichever occurs first. In the proposed framework, a dynamic replacement policy where T_γ is updated at each decision epoch based on the most recent observations is considered. To incorporate this to the system replacement policy, let $T_{\gamma|t}$ represent the suggested replacement time obtained at time t. The expected average cost per unit time of this policy calculated at the tth cycle can be computed as

$$\varphi^{T_{\gamma|t}} = \frac{c_r + c_f \cdot p(L < T_{\gamma|t} | \bar{o}_t = 0, \bar{\boldsymbol{y}}_t)}{\mathbb{E}[\min(L, T_{\gamma|t})]}, T_{\gamma|t} > t, \tag{4.19}$$

where $p(L < T_{\gamma|t} | \bar{o}_t = 0, \bar{\boldsymbol{y}}_t)$ is the conditional probability of a failure replacement given $\bar{\boldsymbol{y}}_t$, that is the system fails before time $T_{\gamma|t}$, and $\mathbb{E}[\min(L, T_{\gamma|t})]$ is the expected effective lifetime, that is the minimum between the system's age (L) and suggested replacement time ($T_{\gamma|t}$). Based on 4.15, it can be concluded that

$$p(L < T_{\gamma|t} | \bar{o}_t = 0, \bar{\boldsymbol{y}}_t) = 1 - r_{T_{\gamma|t}|t}.$$

The expected effective lifetime can be expressed as

$$\mathbb{E}[\min(L, T_{\gamma|t})] = \sum_{x=t+1}^{T_{\gamma|t}-1} (x \times f_{x|t}) + T_{\gamma|t} \times r_{T_{\gamma|t}|t}.$$

The main objective of the proposed process is to reach the optimal value of $T_{\gamma|t}$, while minimizing the unit average cost given in Eq. 4.19. Therefore, for each time t a future time horizon $[t+1, ..., T_u]$ is considered, where T_u is an arbitrary maximum lifetime where the system has already failed. Using exhaustive search, $T^*_{\gamma|t} \in [k+1, ..., T_u]$ is obtained from

$$T^*_{\gamma|t} = \underset{d \in [t+1,...,T_u]}{\operatorname{argmin}} \varphi^d.$$

At time t, the cost of immediate replacement $\frac{c_r}{t}$, using the optimal value of the cost $\varphi^{T^*_{\gamma|t}}$, is computed. Then, if

$$T^*_{\gamma|t} \geq (t+1) \quad \& \quad \varphi^{T^*_{\gamma|t}} < \frac{c_r}{t},$$

that is a future replacement time with a lower cost than immediate replacement exists, then the best choice is to wait until the next cycle. Otherwise, if

$$\varphi^{T^*_{\gamma|t}} \geq \frac{c_r}{t},$$

or if the cost of immediate replacement is lower than the minimum cost of replacement in any future time point, then an immediate replacement is the best choice. It should also be noted that, regardless of the above conditions, the system is always replaced at failure. Based on this dynamic policy, the effective replacement time is

$$T^*_{\gamma} = \min\{L, \inf\{t : \frac{c_r}{t} \leq \varphi^{T^*_{\gamma|t}}\}\}.$$

For simplicity, a simple control index $I_k = \varphi^{T^*_{\gamma|t}} - \frac{c_r}{t}$ and threshold 0 are defined, meaning that the operation is terminated when I_t exceeds zero. Algorithm 2 summarizes the main steps in the dynamic cost-effective replacement policy described above.

Algorithm 2 Summary of the Steps for Dynamic Cost Policy.

procedure DYNAMIC COST POLICY
  Step 1:
    Set t:= 0 and C:=0. Decision action at this point is "Do Nothing" (DN).
  Update:
  Step 2:
    Set $t := t + 1$.
    if$(o_t = 1)$
      Let $C = c_f$ be the total cost.
    go to *Step 4*.
    else
      Collect the condition monitoring signal ($\boldsymbol{y}_t$).
      Compute $\varphi^{T^*_{\gamma|t}}$.
  Step 3:
    Define a control index $I_t = \varphi^{T^*_{\gamma|t}} - \frac{c_r}{t}$.
    if$(\varphi^{T^*_{\gamma|t}} - \frac{c_r}{t} \geq 0)$ STOP (i.e., terminate the operation if $I_t \geq 0$).
      go to *Step 4*.
    else go to *Step 2*.
  Step 4:
    Set $C := C + c_r$.
  Unit average cost for this system is $\frac{C}{t}$; the effective replacement time is t.
end procedure

4.3 Numerical Experiments

Numerous simulation-based numerical experiments were performed for assessing the fidelity of the proposed framework, its application in real-time monitoring, and its advantage over conventional models with regards to decision-making.

4.3.1 Description of the Simulated Dataset

A fully stochastic framework was designed to simulate time-series data for the numerical experiments. According to the proposed structure, multiple trajectories of run-to-failure samples were simulated based on a single-unit degrading system with a one-dimensional hidden degradation process x and a one-dimensional observation process y. Since both x and y are one-dimensional, all parameters of Kalman filter and logistic regressions are expressed in a scalar format and are considered to be time-invariant. The true parameter values used in the simulations are as follows:

$$F = 1.02,\ H = 2,\ Q = 1,\ R = 1,\ \alpha = 2,\ \beta = 4 \text{ and } \beta_0 = -150.$$

For each sample, simulation of the degradation state x and then simulation of the observation signal y over time, were carried out by the following equations:

$$x_t = F \cdot x_{t-1} + \mathcal{N}(0, Q), \tag{4.20}$$

$$y_t = H \cdot x_t + \mathcal{N}(0, R). \tag{4.21}$$

Determining survival time l_i for each sample i and given the degradation state x_t, a random number v_i from $\mathcal{U}(0, 1)$ is drawn at first. The failure time l_i defines the time point where the conditional probability of failure $p(o_t = 1|x_t)$ obtained from λ_t exceeds v. Figure 9 shows plots for processes x, y, as well as the logarithm of

$p(o_t = 1|x_t)$ for a randomly-chosen simulated sample. It can be defined that y is a noisy version of x and that the probability of failure increases monotonically over time. In real-time applications, only the observation data y are available for monitoring the health of the system. Nevertheless, they are enough to estimate the latent process x and the probability of failure $p(o_t = 1|x_t)$. Furthermore, it can be deduced that the log-hazard process follows a linear pattern, where the value is very small when the system is in the as-good-as-new state and linearly involves until system failure. Due to the linear nature of system dynamics, this linear pattern never experiences sudden behavioral changes. While this linearity is mathematically convenient, it is not realistic for most of the practical systems.

4.3.2 Simulated Data and Relations with Real Industrial Settings

It must be emphasized that while the framework assumptions may be valid for a class of industrial systems, its performance will deteriorate if these assumptions are not fully met. As can be seen in [157], Kalman filter diverges in the presence of an improper system model, false modeled processes and/or measurement noise, or unexpected sudden changes of the state vectors. Thus, if any of the assumptions made regarding degradation and measurement processes (e.g., Gaussian noise) are violated, the proposed model may perform poorly. Nevertheless, there are methods (e.g., residual-based testing) to check on the performance of Kalman filter and hazard models in the presence or absence of truth data and whether they adequately represent system dynamics and lifetime.

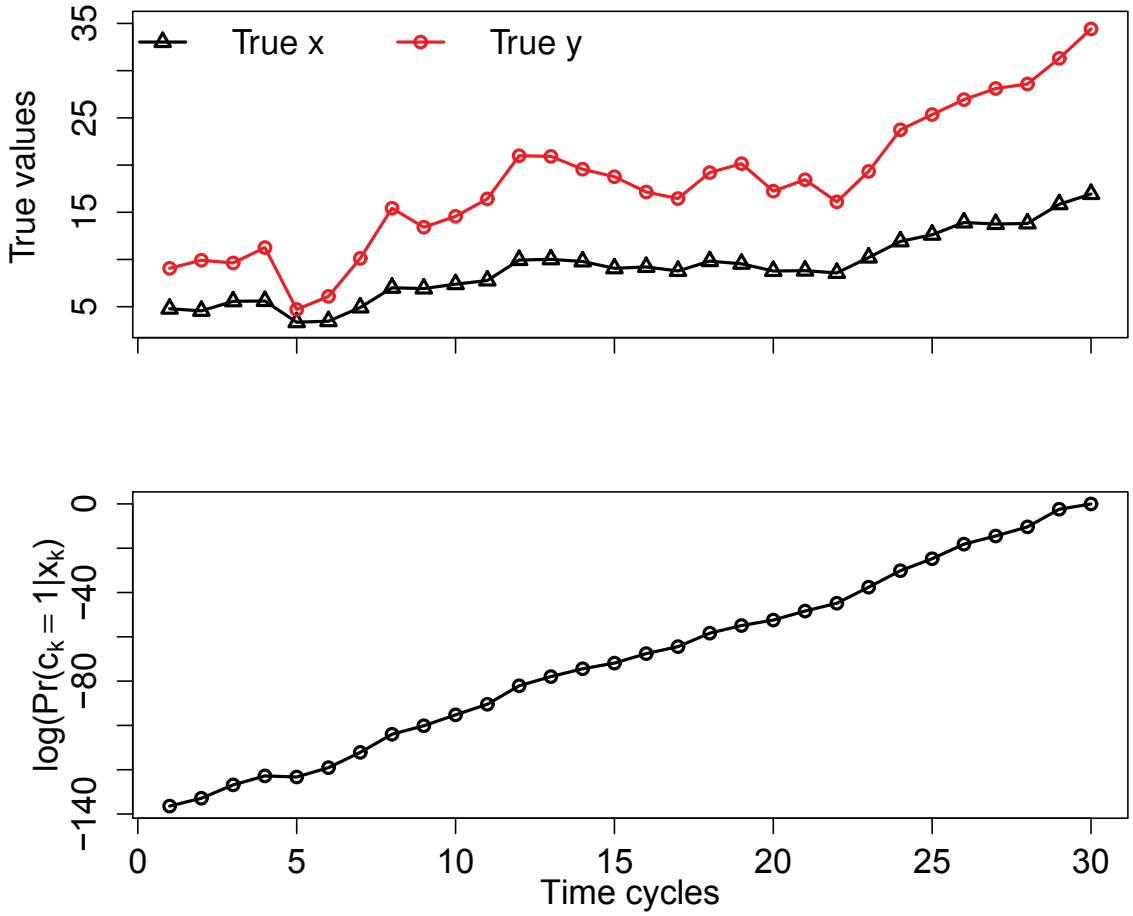


Figure 9: A sample trajectory of x, y, and the logarithm of $p(o_t = 1|x_t)$ for a simulated system.

The system simulated in Section 4.3.1 is assumed to be subject to a single latent degradation process and only indirect information is available through sensor observations. This partially-observed degradation structure is a very typical case of a degrading system that has been considered many times in the literature (e.g., [70, 71, 69, 44]). It should be pointed out that a one-dimensional observation process is considered in all simulation-based numerical experiments. However, the proposed framework can handle more complex cases where multi-dimensional observation processes are used as data for system diagnostics and prognostics.

4.3.3 Parameter Estimation

The efficiency of the parameter estimation procedure was evaluated on a set of multiple trajectories of run-to-failure data. Parameters are categorized into two sets: (a) Kalman filter parameters, namely the transition matrix $\boldsymbol{F}_t$, the observation matrix $\boldsymbol{H}_t$, the state and observation covariance noises $\boldsymbol{Q}_t$ and $\boldsymbol{R}_t$, respectively, and (b) logistic regression parameters, namely the regression coefficient α, time coefficient β, and intercept β_0. For evaluating the effect of the number of samples in the estimation results and assessing the convergence rate to the underlying true values, four cases of N simulated samples ($N \in [10, 20, 40, 80]$), were considered. For each N, the samples were simulated one hundred times (i.e., for 100 runs) and the mean and standard deviation of each parameter estimate over all runs were calculated. The results are shown in Figures 10-13 and 14-15, where for each parameter and each N, the boxplots of the estimates over one hundred runs are presented. The boxplots prove that as the number of samples (N) increases, the variance of each estimate decreases. Furthermore, Table 2 presents the true values of the parameters, as well as the means, standard deviations, and the root-mean-square errors (RMSE) of the estimated values. The results verify that parameter estimates are very close to true values, particularly when the number of samples increases. Also, results verify that the error of estimation approximates zero as N increases.

Table 2: Means and standard deviations (SD) of estimated values (100 runs).

Number of samples		*N=10*			*N=20*			*N=40*			*N=80*		
Parameter	True	Mean	SD	RMSE	Mean	SD	RMSE	Mean	SD	RMSE	Mean	SD	RMSE
F	1.02	1.016	0.013	0.014	1.018	0.001	0.001	1.017	0.007	0.007	1.018	0.005	0.005
H	2	2.012	0.143	0.143	1.985	0.102	0.103	2.008	0.07	0.07	2.002	0.045	0.045
Q	1	1.048	0.295	0.298	1.062	0.282	0.287	1.05	0.193	0.198	1.052	0.175	0.182
R	1	0.903	0.525	0.532	0.92	0.474	0.478	0.879	0.370	0.388	0.892	0.325	0.341
α	2	2.015	0.232	0.231	1.982	0.165	0.165	2.015	0.113	0.114	2.008	0.074	0.074
β	4	3.988	0.103	0.103	3.986	0.065	0.066	3.988	0.045	0.046	3.982	0.03	0.034

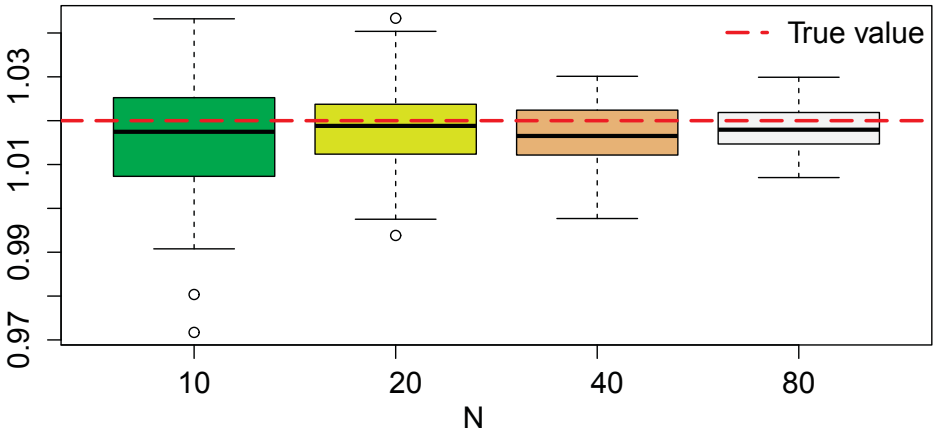


Figure 10: Transition parameter F.

4.3.4 Residual Life Prediction

For each sample, it is assumed that observations are obtained until a certain proportion of its true life (such as 60%,65%,70%,75%,80%,85%,90% and 95%). Based on these observations, the RUL probability distribution presented in Eq. 4.17 is estimated, and the mean residual life shown in Eq. 4.18 is calculated. For each N and each point in the lifetime, the means and standard deviations of true and estimated remaining lives, the root mean square error, and the relative error between the true and the estimated remaining useful life of all samples, are reported (see Table 3). The relative error demonstrates the average absolute difference between true and estimated RUL of the samples. These results corroborate that the mean residual life estimates are relatively close to the true residual life and that the error decreases when the estimates are made closer to the end of the lifetime (since more condition monitoring data are available). The results also show that as the devices age, the prediction uncertainty decreases.

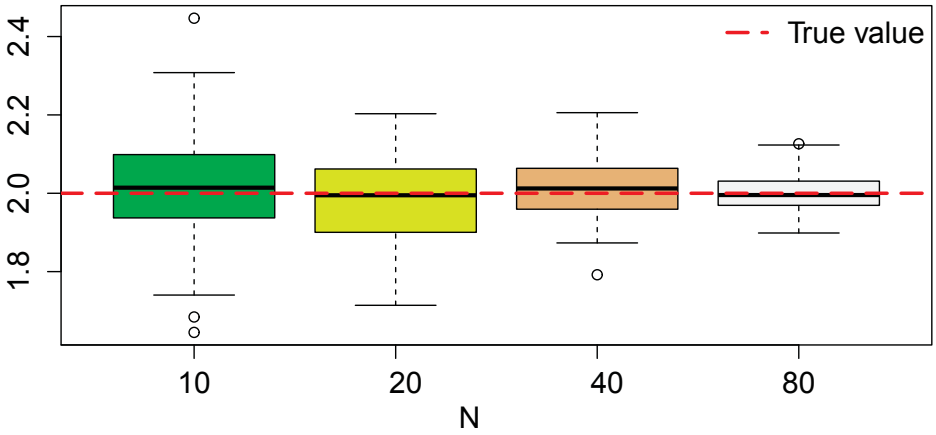


Figure 11: Observation parameter H.

4.3.5 Dynamic Replacement Policy

The replacement policy described in Section 4.2.4 is compared with two other types of cost policies: i) a preventive maintenance policy, in which the inspection and replacement of equipment occur in fixed time intervals regardless of the state of the equipment, and ii) a corrective maintenance policy, in which replacements occur after failure. Regarding the former case, the cost of scheduled replacement, which can occur at any time point before the true life of the sample as given in Eq. 4.22, is calculated. Considering N samples, l_i denotes the true lifetime of each system $i \in [1, ..., N]$, L denotes a random variable representing the lifetime, and T_γ denotes the replacement time associated with a cost-effective replacement policy γ ($T_\gamma \in [1, ..., \max(l_i)]$). The expected average cost per unit time of this policy calculated at the tth cycle can be computed as

$$\varphi^{T_{\gamma|t}} = \frac{(N-J)\cdot c_r + J\cdot(c_r + c_f)}{\mathbb{E}[\min(L, T_{\gamma|t})]}, \tag{4.22}$$

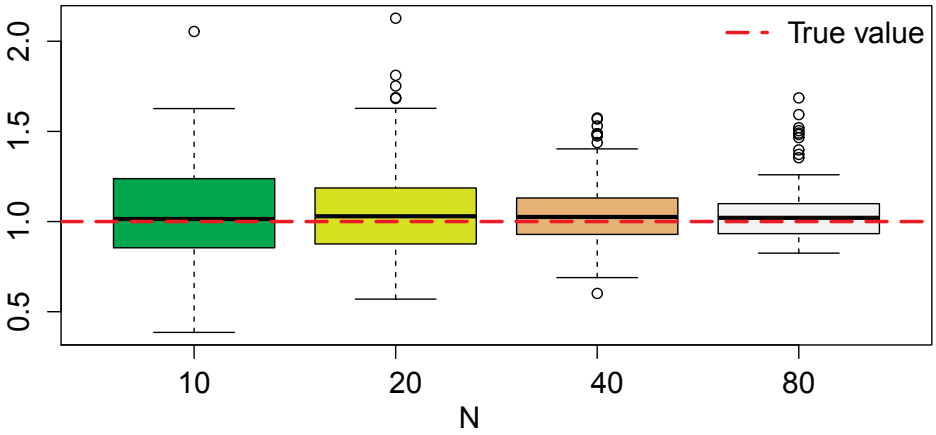


Figure 12: Latent process noise variance Q.

where $J = \sum_{i=1}^{N} \mathbf{1}_{\{l_i \leq T_{\gamma|t}\}}$. The expected effective lifetime in this case is expressed as

$$\mathbb{E}[\min(L, T_{\gamma|k})] = \sum_{i=1}^{N} \{l_i \times \mathbf{1}_{\{l_i \leq T_{\gamma|k}\}}\} + T_{\gamma|k}(N - J).$$

Finally, the corrective maintenance suggests that all samples fail without any maintenance actions taking place. Here, the expected cost can be expressed as follows:

$$\varphi^{T_{\gamma|t}} = \frac{N \cdot (c_r + c_f)}{\mathbb{E}(L)}.$$

Figure 16 shows the application of each maintenance policy in real time on three different samples. For each sample, the true failure time and the recommended maintenance times from the preventive maintenance and the proposed policies are shown as vertical lines. The control index I_k plotted in each figure is the result of our proposed framework, that is the system is replaced when this control index exceeds zero or the system fails, whichever occurs first. Note that the preventive maintenance time point is not affected by each sample's real time degradation condition and is fixed in all samples. For sample 1, the preventive maintenance model recommends

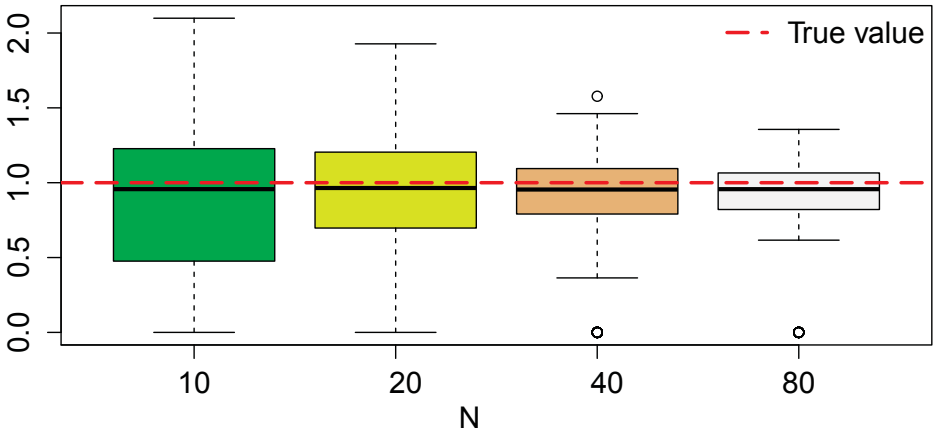


Figure 13: Observation process noise variance R.

replacement too early while the proposed policy favors terminating the operation for maintenance only five cycles before failure. For sample 2, the opposite occurs, that is the preventive maintenance model recommends a replacement later than the proposed model. Finally, for sample 3, both preventive and the proposed models fail to prevent failure. Cost analysis experiments were conducted for $N = 5000$ samples and the results are shown in Table 4. On average, the proposed policy provides the lowest average cost compared to the other two policies. Furthermore, it yields a lower failure rate than both preventive and corrective replacement policies. The proposed method provides a higher effective lifetime than the preventive maintenance policy and a lower lifetime than the corrective maintenance case. In addition, raising the cost of failure replacement c_f with respect to the cost of replacement c_r, increases the average cost while the failure rate decreases. The proposed policy becomes similar to the preventive maintenance policy, when the cost of failure decreases. Furthermore, both the proposed and preventive maintenance models perform significantly better than corrective maintenance with respect to total costs and failure preven-

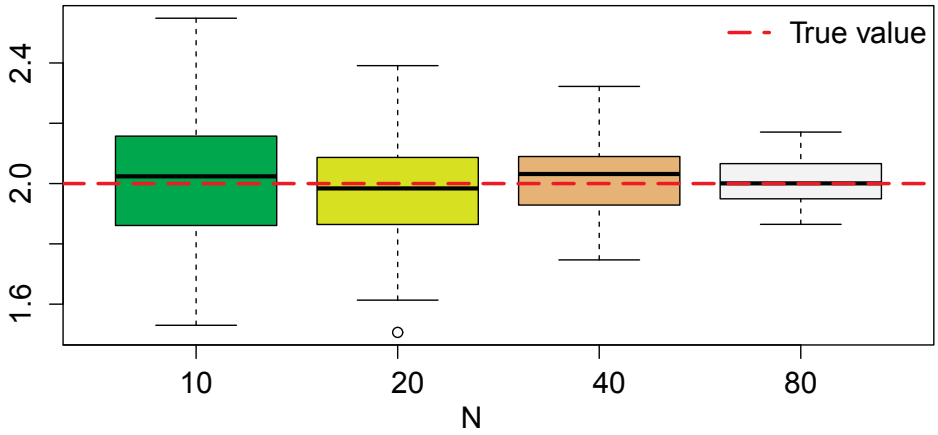


Figure 14: Regression coefficient α.

tion. This numerical analysis has a great value, because it reveals how much a CBM framework and monitoring systems health condition in real time can help decrease the average maintenance cost of the system and unexpected failures in the long-run. The numerical analysis also assists decision-makers in determining the proper balance between losing effective lifetimes due to early replacement/repair, preventing catastrophic failures, and decreasing long-run average cost. The results of the cost-effective replacement policy can also be visualized by another mechanism that highlights the trade-off between the percentage of missing operating time and the percentage of unexpected failures for the N simulated samples and various combinations of c_r and c_f (see Figure 17). In order to test the sensitivity of the different cost policies, a combination where $c_r = c_f$ is also considered. The percentage of missing operating time is the fraction of the total potential operating time when the sample does not operate due to early maintenance recommended by the proposed model. The percentage of unexpected failures is the fraction of replacements that occur as a result of an inefficient suggested replacement time, that is, the number of failures that oc-

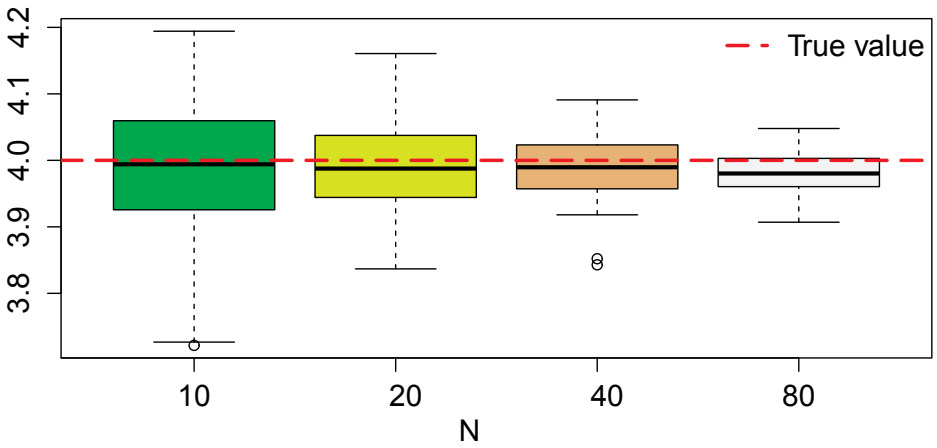


Figure 15: Time coefficient β.

cur while the sample is still operating divided by the total number of samples. As the cost ratio increases, the proposed policy tends to recommend earlier replacement time, resulting in lower failure percentage but higher missing operating time. An analysis like this can be used to determine the desired trade-off between important performance measures and as a sensitivity of the results for this trade-off. It can also be used to compare the potential cost-saving of a CBM framework with the actual cost of implementing it (e.g., acquiring sensors), so that a better-informed decision can be made regarding applying CBM as the best strategy for maintenance.

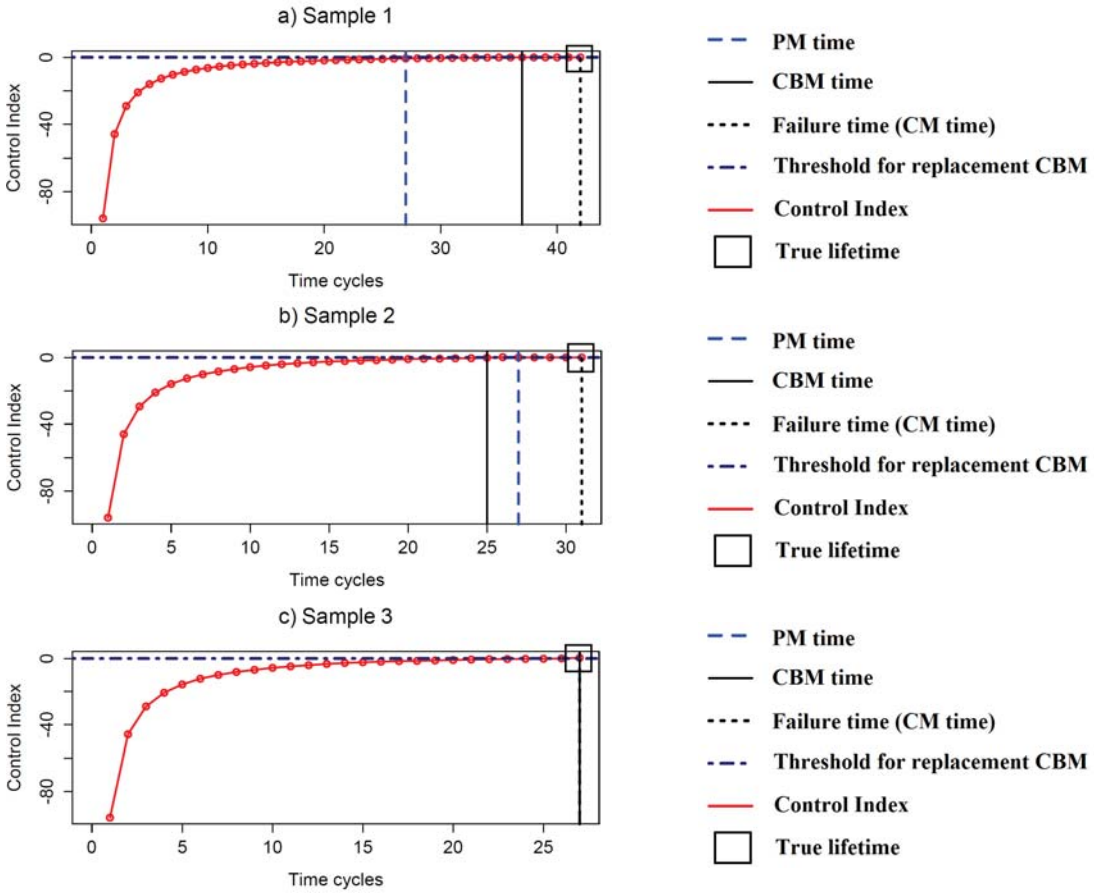


Figure 16: Application of each maintenance policies in three simulated samples.

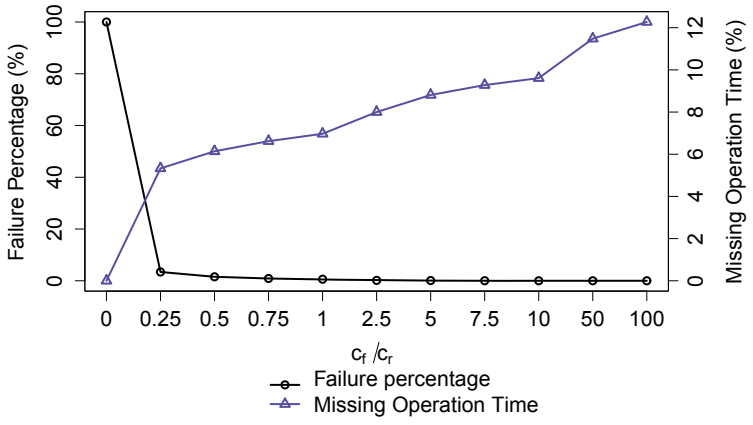


Figure 17: Trade-off between the average percentage of missing operating time and the average percentage of unexpected failures for various combinations of $\frac{c_f}{c_r}$ for 5000 simulated samples.

4.4 Summary

The proposed framework was developed by combining Bayesian filtering and traditional machine learning classification models that can potentially be extended to other applications and for other purposes, such as time-series classifications. The framework yields very good results for systems that are controlled by linear and Gaussian dynamics, but it performs rather poorly when these assumptions are not met. Furthermore, the log-hazard process is assumed to follow a smooth path until system failure without considering sudden changes in the degradation pattern, which are always possible throughout the systems operation. This concept is addressed in the next Chapter.

The outputs of this framework are latent state estimates that can be later used for RUL estimation and for dynamic maintenance decision-making. Theoretically, this framework represents Phase I of the proposed structure, and its outputs can be used to establish an environment to train DRL agent for maintenance cost minimization. However, because of its inability to describe non-linear systems, Phase I must be represented by a framework that describes more complex system dynamics. Therefore, a particle filter-based framework, as described in the next Chapter, will be used.

Table 3: RUL results for the testing simulated dataset.

% of true life	Mean True RUL	Mean Est. RUL	SD True RUL	SD Est. RUL	RMSE	Relative Error
			N=10			
60	14.4	13.46	1.35	1.42	1.53	0.04
65	12.7	11.94	1.06	1.53	1.49	0.039
70	10.9	10.35	1.1	1.64	1.4	0.029
75	9.2	8.89	0.79	1.13	0.82	0.018
80	7.3	6.49	0.67	0.93	1.08	0.025
85	5.6	5.14	0.52	0.88	0.79	0.019
90	3.9	3.8	0.32	0.84	0.71	0.018
95	2	1.89	0	0.73	0.7	0.017
			N=20			
60	14	12.83	1.65	1.77	2.05	0.045
65	12.4	11.5	1.35	1.8	1.8	0.04
70	10.75	10.12	1.33	1.72	1.66	0.036
75	8.95	8.41	1.19	1.54	1.47	0.033
80	7.3	6.96	0.92	1.23	1.15	0.025
85	5.6	5.35	0.68	1.09	1	0.023
90	3.9	3.75	0.55	1.08	1	0.023
95	2.05	2.19	0.22	0.58	0.69	0.015
			N=40			
60	13.85	12.45	1.73	1.55	2.21	0.048
65	12.18	11	1.4	1.36	1.81	0.041
70	10.45	9.62	1.24	1.34	1.43	0.034
75	8.73	8	0.96	1.44	1.52	0.038
80	7.15	6.7	0.86	1.4	1.43	0.035
85	5.43	5.3	0.68	1.17	1.2	0.029
90	3.85	3.72	0.43	0.99	0.93	0.023
95	2.03	2.12	0.16	0.58	0.56	0.014
			N=80			
60	13.87	12.82	1.5	1.52	2.32	0.052
65	12.2	11.3	1.33	1.49	2.18	0.049
70	10.53	9.74	1.24	1.39	1.99	0.043
75	8.83	8.22	1	1.29	1.75	0.037
80	7.05	6.62	0.78	1.24	1.44	0.033
85	5.49	5.22	0.62	1.03	1.17	0.028
90	3.85	3.73	0.48	0.92	0.98	0.022
95	2.08	2.11	0.27	0.66	0.64	0.016

Table 4: Results for the testing simulated dataset.

	Average Cost	Failure Rate	Effective Lifetime
$c_r = 100, c_f = 100$			
CBM (Our model)	3.222	0.58%	31.21
Preventive Maintenace	3.843	3.58%	26.977
Corrective Maintenance	5.96	100%	33.55
$c_r = 100, c_f = 500$			
CBM (Our model)	3.28	0.06%	30.576
Preventive Maintenance	4.109	0.52%	24.998
Corrective Maintenance	17.883	100%	33.55
$c_r = 100, c_f = 1000$			
CBM (Our model)	3.299	0%	30.306
Preventive Maintenace	4.213	0.01%	23.999
Corrective Maintenance	32.78	100%	33.55
$c_r = 100, c_f = 10000$			
CBM (Our model)	3.399	0%	29.413
Preventive Maintenace	4.589	0.01%	23.999
Corrective Maintenance	301.029	100%	33.55

CHAPTER 5

Monitoring and Control of Hybrid Non-Linear Systems

This Chapter presents Part I of the proposed decision-making structure. This framework introduces a condition-based maintenance method for degradation monitoring and RUL estimation that considers an SSM with multiple latent and observable layers of system dynamics. SSMs of this type are also known as *hybrid SSMs* (HSSM). Most of the previously designed SSMs assume parametric processes relating latent states with sensor observations, which are both difficult to establish and hard to empirically validate - especially in the presence of multivariate sensor observations. So far, no approach has been proposed for dealing with the joint estimation of states and parameters for multi-layer SSMs without considering parametric or distributional assumptions between latent states and multidimensional observations. The proposed HSSM is fully hierarchical and its stochastic layers include: i) a latent degradation process that represents the degradation progression of a system, ii) a binary latent operating condition process that denotes whether a system operates normally, or a fault occurred during its operation, iii) a failure hazard process that represents the probability of failure every time an observation signal is collected, iv) a multidimensional

observation process that represents sensor outputs, and v) a binary working condition process that shows whether a system operates or has failed (stopped working).

5.1 Main Framework

The proposed HSSM takes into account all the information collected through multiple sensors and has an interpretable hierarchical structure that considers causal dependencies between data elements and data sources. The HSSM expands the classic SSM [72] into including a discrete hidden state c_t that represents the latent health condition, or *mode*, of the system at time t, along with the continuous latent state $\boldsymbol{x}_t$. The mode is either binary (normal/faulty), or discrete/categorical (normal, partially faulty, and faulty). The sensor observation process $\boldsymbol{y}_t$, at time t, is represented by an $m-$dimensional vector, where m is the number of condition monitoring sensors. Following the same concept as in Chapter 4, a dynamic hazard process λ_t denotes the conditional probability of failure between the tth and $(t+1)$th intervals as a function of the degradation state. However, unlike the previous framework, the hazard process here is fully stochastic. The model also includes an observable variable that represents the survival/working status of the system at any time point. This process has two potential outcomes, $o_t = 0$ and $o_t = 1$, denoting normal operation and failure, respectively. Each can be considered as a random variable with a distribution dependent on λ_t. The inclusion of hazard and working status processes establishes an entirely stochastic system breakdown, depending only on the hazard process without having to define fixed failure thresholds.

Finally, the HSSM considers a layer of d-dimensional operating inputs representing known information to the system over time. These inputs are denoted as $\boldsymbol{u}_t$ and

represent both control inputs at time t, and/or other known but uncontrollable inputs, such as environmental conditions. Given $\boldsymbol{u}_t$, the continuous hidden degradation state process $\boldsymbol{x}_t$, the discrete hidden mode process c_t, the hazard process λ_t, an m-dimensional observation process $\boldsymbol{y}_t$, and the failure process o_t, the proposed HSSM will be:

$$c_t = f_c(c_{t-1}, \boldsymbol{u}_t, \boldsymbol{\theta}_c), \tag{5.1}$$

$$\boldsymbol{x}_t = f_x(\boldsymbol{x}_{t-1}, c_t, \boldsymbol{u}_t, \boldsymbol{\theta}_x), \tag{5.2}$$

$$\lambda_t = f_\lambda(\boldsymbol{x}_t, c_t, \boldsymbol{u}_t, \boldsymbol{\theta}_\lambda), \tag{5.3}$$

$$\boldsymbol{y}_t = f_y(\boldsymbol{x}_t, c_t, \boldsymbol{u}_t, \boldsymbol{\theta}_y), \tag{5.4}$$

$$o_t = f_o(\lambda_t, \boldsymbol{\theta}_o). \tag{5.5}$$

Stochastic functions $f_{\cdot}$ relate variables and parameters either within the same level (latent or observable), or between different levels, and vector $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_c, \boldsymbol{\theta}_x, \boldsymbol{\theta}_\lambda, \boldsymbol{\theta}_y, \boldsymbol{\theta}_o\}$ represents the cardinality of model parameters needed to characterize those functions. The HSSM complexity depends significantly on the parametric formulas that are selected to represent these functions. Figure 18 provides an overview of the proposed HSSM, where the arrows denote the conditional dependencies between latent and observed variables. The continuous latent degradation process, similar to [158], is assumed to be one-dimensional (x_t) and is considered as a health index of the system that degrades exponentially over time. The system is assumed to be in a perfect health when $x_t \simeq 10$, although this value cannot be considered as a hard limit due to uncertainty. Thus, even in the case of perfect system health, x_t can fluctuate slightly over or under 10. Except the sensor observation process that is described

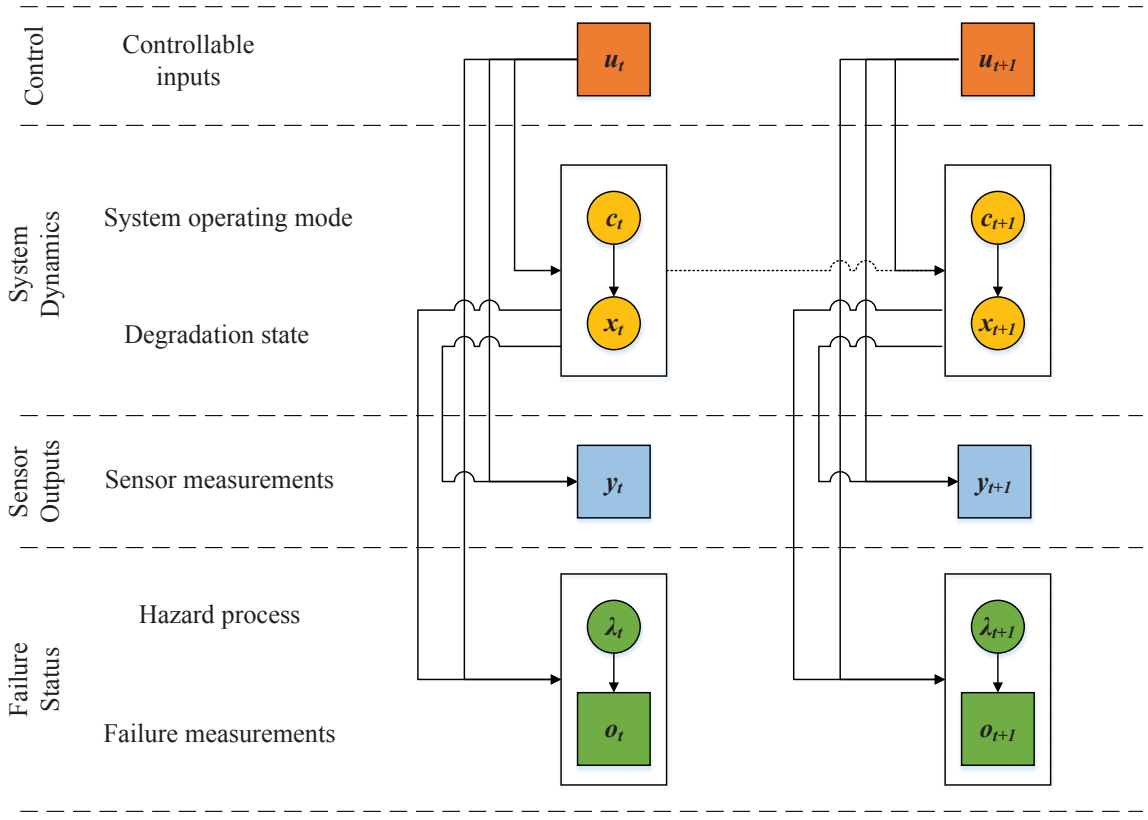


Figure 18: The stochastic framework of the proposed structure.

later in Section 5.1.1, a detailed description of the other processes on both latent and observable levels are described in detail below:

- **Evolution of the Hidden Mode Process (c_t):** The first latent level of the proposed HSSM represents the operating condition of the system. The parameters that define this process, denoted by $\boldsymbol{\theta}_c$, are estimated during the training phase. A simple case of a binary discrete latent state c_t, where $c_t = 0$ denotes normal system operation and $c_t = 1$ means a faulty condition, is utilized. Transitions between normal and faulty condition states are modeled with a Hidden Markov Model (HMM) and are described by a Markov transition matrix.

- **Evolution of the Degradation Process (x_t):** Eq. 5.2 shows how the degradation process is affected by the mode at time t. Data-driven SSMs/HSSMs do not require any specific form of the function f_x that describes the evolution of the continuous latent degradation, except in cases where the system dynamics are of linear nature (see Chapter 4). For most of the real-world systems, an exact form of f_x is very difficult, or even impossible, to exist. In the absence of any previous studies for a similar type of system, f_x can be either one of the commonly used and previously studied statistical models (e.g., linear, exponential, and power law functions) or it can be a completely new mathematical function that reasonably represents the degradation process.

- **Evolution of Hazard (λ_t) and Failure Processes (o_t):** Hazard process $\lambda_t \in \{0, 1\}$ represents the conditional probability of failure between time $t-1$ and time t, given the system's survival up to time $t-1$, and depends on latent states x_t, c_t, and inputs $\boldsymbol{u}_t$ (and potentially other covariates). The logistic regression was utilized to characterize f_λ, due to its simple and flexible mathematical structure. Logistic regression is able to formulate the probability of failure under the binary assumption of being either at a normal or a failed state given a set of covariates. The structure of the hazard process f_λ for the proposed framework is:

$$\lambda_t = \frac{1}{1 + \exp[-(\alpha x_t + \beta c_t + \boldsymbol{\gamma}\boldsymbol{u}_t + \beta_0)]},$$

where regression coefficients $\boldsymbol{\theta}_\lambda = \{\alpha, \beta, \boldsymbol{\gamma}\}$ are associated with latent states x_t, c_t, and operating inputs $\boldsymbol{u}_t$, respectively. Coefficient β_0 is the intercept. Unlike the framework in Chapter 4, this hazard process is directly dependent on the

mode process and the hazard pattern can experience sudden changes during system operation.

Regarding the failure process, the probability of the system being in the failure state at time t ($o_t = 1$), is λ_t, or

$$p(o_t = 1 | o_{1:t-1} = \mathbf{0}) = \lambda_t.$$

Since no other parameters are used to related λ_t to o_t, the parameter set $\boldsymbol{\theta}_o$ is an empty set, and thus, not part of the model parameters.

Fig. 19 presents sample plots for the five layers of system dynamics from a simulated system with one-dimensional sensor observations. It can be seen that the system becomes faulty at around time $t = 20$. For clearer illustration of the hazard trend, the logarithm values of the actual hazard are shown.

5.1.1 An Extreme Learning Machine for Sensor Data

The most significant contribution of this framework is the non-parametric form of the observation process. The majority of SSMs in the literature assume parametric, time-invariant forms for f_y [159]. Modern systems generate multidimensional condition monitoring measurements, and therefore fixed parametric relationships or distributional assumptions (with possibly many unknown parameters) between observations and latent states are unrealistic and often do not exist. To then represent, the stochastic distribution given in Eq. 5.4, the proposed framework utilizes a relatively new data-driven method called *Extreme Learning Machine* (ELM). The important property of ELM is the non-iterative linear solution for the output weights [160],

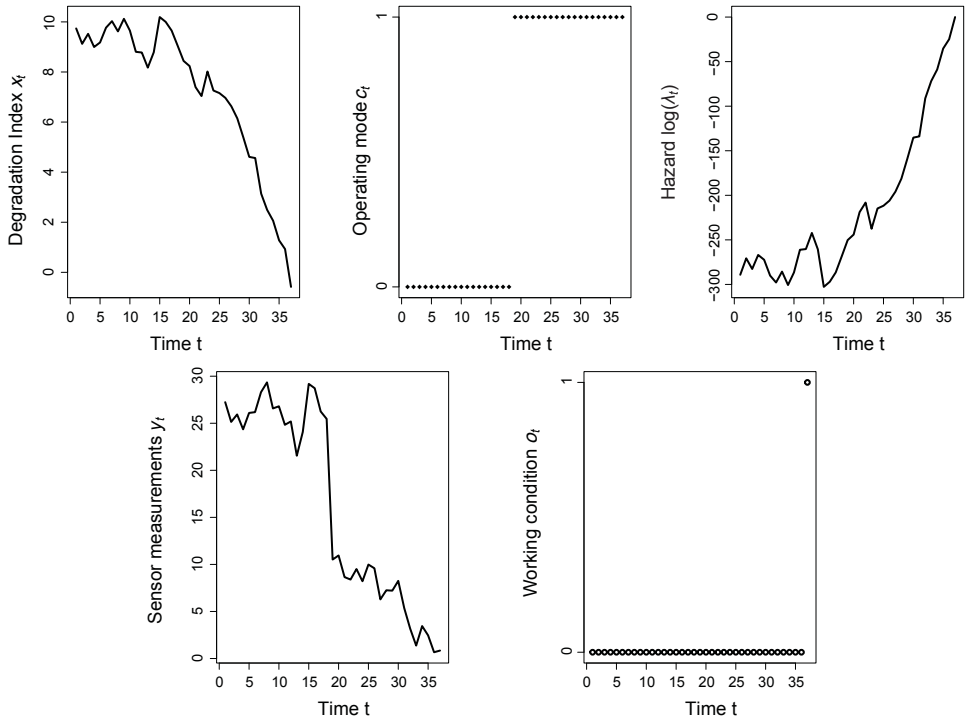


Figure 19: Sample plots of x_t, c_t, λ_t, y_t, and o_t.

which makes it a good candidate to represent the relationship between the multi-dimensional observation data and other latent states.

The ELM-based observation process is explained below with function f_y input variables denoted by ξ_1 to ξ_{d+2} (i.e., x_t, c_t and $\boldsymbol{u}_t$). The variables at the output layers are y_1 to y_m, for m-dimensional sensor measurements. The ELM for a single sample is defined with inputs $\boldsymbol{\xi}_t = \{x_t, c_t, \boldsymbol{u}_t\}$ and outputs $\boldsymbol{y}_t$. For $\hat{N}$ hidden neurons, the hidden layer outputs are computed using a non-linear activation function $g(.)$ as:

$$\tau_j = g(\mathbf{a}_j \cdot \boldsymbol{\xi}_t + b_j), j = 1, 2, ..., \hat{N},$$

where $\mathbf{a}_j = (\mathrm{a}_{j1}, \mathrm{a}_{j2}, ..., \mathrm{a}_{jd+2})$ denotes the weight vector between inputs and the jth hidden node and b_j is the bias. The hidden layer to the output layer are then related

as:

$$v_{m'} = \sum_{j=1}^{\hat{N}} \delta_{jm'}\big(g(\mathbf{a}_j \cdot \boldsymbol{\xi}_t + b_j)\big) = \sum_{j=1}^{\hat{N}} \delta_{jm'}\tau_j,\ m' = 1, ..., m,$$

where $\boldsymbol{\delta}_j = [\delta_{j1}, ...\delta_{jm}]$ represents the output weight vector for each hidden neuron j. Set $[\mathbf{a}_j, b_j, \boldsymbol{\delta}_j]$ for $j = 1, 2, ..., \hat{N}$ represents parameter vector $\boldsymbol{\theta}_y$. ELM training is equivalent to a typical least square problem with target value y_{tm} and estimated values $v_{m'} = \sum_{j=1}^{\hat{N}} \delta_{jm'}\tau_j$ using available training data. The sensor signal is assumed to follow a normal distribution with the ELM output as mean, and a standard deviation σ that can be approximated by the root of the sample error (true value - estimated value from ELM) variance in the training data. Finally, the stochastic relationship between the input layers and the mth observation measurement is summarized as:

$$y_{tm'} \sim \mathcal{N}\Big(\sum_{j=1}^{\hat{N}} \delta_{jm'}\tau_j, \sigma^2\Big), m' = 1, 2, ..., m.$$

5.2 Model Training Framework

Given a single system with lifetime T, a sequence of continuous sensor observations $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_T$, and discrete working condition observations $o_1, \ldots, o_T$, the maximum likelihood estimation is utilized for jointly estimating the latent states $\boldsymbol{z}_{1:t} = \{c_{1:t}, x_{1:t}, \lambda_{1:t}\}$, and unknown parameters $\boldsymbol{\Theta}$. The results can be unequivocally extended to multiple cases. For brevity, operating inputs $\boldsymbol{u}_{1:t}$ are removed.

5.2.1 An Iterative Approach for Model Training

The likelihood function based on observable data is expressed by the joint likelihood of observations:

$$p_{\boldsymbol{\Theta}}(\boldsymbol{y}_{1:T}, o_{1:T}) = \Big\{p_{\boldsymbol{\Theta}}(\boldsymbol{y}_1)\prod_{t=2}^{T} p_{\boldsymbol{\Theta}}(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})\Big\}\Big\{p_{\boldsymbol{\Theta}}(o_1)\prod_{t=2}^{T} p_{\boldsymbol{\Theta}}(o_t|o_{1:t-1})\Big\}, \tag{5.6}$$

where $p_{\boldsymbol{\Theta}}$ denotes a probability density function given parameter set $\boldsymbol{\Theta}$. Equivalently, the log-likelihood will be:

$$\begin{aligned}\mathcal{L}_{\boldsymbol{\Theta}}(\boldsymbol{y}_{1:T}, o_{1:T}) &= \log p_{\boldsymbol{\Theta}}(\boldsymbol{y}_{1:T}, o_{1:T}) \\ &= \log\{p_{\boldsymbol{\Theta}}(\boldsymbol{y}_1)\} + \log\{\prod_{t=2}^{T} p_{\boldsymbol{\Theta}}(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})\} + \log\{p_{\boldsymbol{\Theta}}(o_1)\} \\ &+ \log\{\prod_{t=2}^{T} p_{\boldsymbol{\Theta}}(o_t|o_{1:t-1})\}.\end{aligned} \tag{5.7}$$

Using the law of total probability and the Markov property of the HSSM, the densities in Eq. 5.7 can be analyzed as:

$$p_{\boldsymbol{\Theta}}(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}) = \int p_{\boldsymbol{\Theta}}(\boldsymbol{y}_t|\boldsymbol{z}_t) p_{\boldsymbol{\Theta}}(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) d\boldsymbol{z}_t, \tag{5.8}$$

$$p_{\boldsymbol{\Theta}}(o_t|o_{1:t-1}) = \int p_{\boldsymbol{\Theta}}(o_t|\boldsymbol{z}_t) p_{\boldsymbol{\Theta}}(\boldsymbol{z}_t|o_{1:t-1}) d\boldsymbol{z}_t. \tag{5.9}$$

The (unknown) HSSM parameters can be estimated through maximization of Eq. 5.7 considering Eqs. 5.8-5.9 over $\boldsymbol{\Theta}$. Since latent states $\boldsymbol{z}_t$ are unknown, the log-likelihood function cannot be directly maximized and the Expectation-Maximization (EM) algorithm can be used, due to its ability to perform joint state-parameter estimation. The log-likelihood is approximated as $\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}') \approx \mathcal{L}_{\boldsymbol{\Theta}}(\boldsymbol{y}_{1:T}, o_{1:T})$, based on observations $\boldsymbol{y}_{1:T}, o_{1:T}$ and a given parameter set $\boldsymbol{\Theta}'$. The function $\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}')$ is the conditional mean of the log-likelihood function:

$$\begin{aligned}\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}') &= \mathbb{E}_{\boldsymbol{\Theta}'}\{\mathcal{L}_{\boldsymbol{\Theta}}(\boldsymbol{z}_{1:T}, \boldsymbol{y}_{1:T}, o_{1:T}) | (\boldsymbol{y}_{1:T}, o_{1:T})\} \\ &= \int \mathcal{L}_{\boldsymbol{\Theta}}(\boldsymbol{z}_{1:T}, \boldsymbol{y}_{1:T}, o_{1:T}) p_{\boldsymbol{\Theta}'}(\boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T}, o_{1:T}) d\boldsymbol{z}_{1:T},\end{aligned} \tag{5.10}$$

where

$$\mathcal{L}_{\boldsymbol{\Theta}}(\boldsymbol{z}_{1:T}, \mathbf{y}_{1:T}, o_{1:T}) = \log p_{\boldsymbol{\Theta}}(\boldsymbol{z}_{1:T}, \mathbf{y}_{1:T}, o_{1:T}). \tag{5.11}$$

The authors in [161] proved that iteratively maximizing $\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}')$ leads to increasing the likelihood function. Therefore, instead of directly optimizing Eq. 5.7 by considering a random initial estimate $\boldsymbol{\Theta}_0$, the joint likelihood function can be iteratively improved by maximizing Eq. 5.10. This iterative procedure terminates using a stopping criterion, such as no improvement in the likelihood function or the change of parameters after each iteration is lower than a specified tolerance. The EM process is presented in Algorithm 3. The main challenges of applying EM algorithm in the proposed framework are i) computing the cost function $\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}')$ in the E-step, ii) initialization of latent states due to the non-parametric form of f_y, and iii) maximizing $\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}')$ over multi-dimensional parameter set $\boldsymbol{\Theta}_0$ in the M-step. All of them are addressed in Sections 5.2.1.1-5.2.1.3, respectively.

Algorithm 3 Expectation-Maximization Algorithm.

Input: Initial values of parameters $\boldsymbol{\Theta}_0$ and latent variables.
while *stopping criterion not satisfied* **do**
  Expectation (E-step): Compute $\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}')$.
  Maximization (M-step): Calculate: $\boldsymbol{\Theta}_{new} = \text{argmax}_{\boldsymbol{\Theta}}\, \mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}')$.
  if *algorithm converges* **then**
    $\boldsymbol{\Theta}^* = \boldsymbol{\Theta}_{new}$. Go to **Output**.
  else
    $\boldsymbol{\Theta}' = \boldsymbol{\Theta}_{new}$. Go to *Expectation*.
  end if
end while
Output: Final estimates of parameters $\boldsymbol{\Theta}^*$.

5.2.1.1 Computing Function $\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}')$ with Particle Filtering

Using Bayes' rule, the Markov property of the HSSM, and the generative structure between variables at different layers, the joint-likelihood function in Eq. 5.11 becomes:

$$\mathcal{L}_{\boldsymbol{\Theta}} = \log p_{\boldsymbol{\Theta}}(\boldsymbol{y}_{1:T}|c_{1:T}, x_{1:T}, \lambda_{1:T}) + \log p_{\boldsymbol{\Theta}}(o_{1:t}|\lambda_{1:T}) \tag{5.12}$$

$$
\begin{aligned}
&+ \log p_{\boldsymbol{\Theta}}(c_{1:T}) + \log p_{\boldsymbol{\Theta}}(\lambda_{1:T}) + \log p_{\boldsymbol{\Theta}}(x_{1:T}) \\
&= \log p_{\boldsymbol{\Theta}}(c_1) + \log p_{\boldsymbol{\Theta}}(x_1) + \log p_{\boldsymbol{\Theta}}(\lambda_1) \\
&+ \sum_{t=2}^{T-1} \log p_{\boldsymbol{\Theta}}(c_t|c_{t-1}) + \sum_{t=2}^{T-1} \log p_{\boldsymbol{\Theta}}(x_t|x_{t-1}, c_t) \\
&+ \sum_{t=1}^{T} \log p_{\boldsymbol{\Theta}}(\lambda_t|c_t, x_t) \\
&+ \sum_{t=1}^{T} \log p_{\boldsymbol{\Theta}}(o_t|\lambda_t) + \sum_{k=1}^{T} \log p_{\boldsymbol{\Theta}}(\boldsymbol{y}_t|c_t, x_t, \lambda_t).
\end{aligned}
$$

For approximating the log-likelihood using the EM algorithm, the conditional expectation $\mathbb{E}_{\boldsymbol{\Theta}_t}\{\cdot|\boldsymbol{y}_{1:T}, o_{1:T}\}$ on both sides of Eq. 5.12 is applied as:

$$
\begin{aligned}
\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}') &= \int \log p_{\boldsymbol{\Theta}}(c_1) p_{\boldsymbol{\Theta}'}(c_1|\boldsymbol{y}_{1:T}, o_{1:T}) dc_1 \\
&+ \int \log p_{\boldsymbol{\Theta}}(x_1|c_1) p_{\boldsymbol{\Theta}'}(x_1, c_1|\boldsymbol{y}_{1:T}, o_{1:T}) dx_1 \\
&+ \int \log p_{\boldsymbol{\Theta}}(\lambda_1|x_1, c_1) p_{\boldsymbol{\Theta}'}(\lambda_1, x_1, c_1|\boldsymbol{y}_{1:T}, o_{1:T}) d\lambda_1 \\
&+ \sum_{t=2}^{T} \int \log p_{\boldsymbol{\Theta}}(c_t|c_{t-1}) p_{\boldsymbol{\Theta}'}(c_t, c_{t-1}|\mathbf{y}_{1:T}, o_{1:T}) dc_t dc_{t-1} \\
&+ \sum_{t=2}^{T} \int \log p_{\boldsymbol{\Theta}}(x_t|x_{t-1}, c_t) p_{\boldsymbol{\Theta}'}(x_t, x_{t-1}, c_t|\boldsymbol{y}_{1:T}, o_{1:T}) dx_t dx_{t-1} \\
&+ \sum_{t=1}^{T} \int \log p_{\boldsymbol{\Theta}}(\lambda_t|x_t, c_t) p_{\boldsymbol{\Theta}'}(\lambda_t, x_t, c_t|\boldsymbol{y}_{1:T}, o_{1:T}) d\lambda_t \\
&+ \sum_{t=1}^{T} \int \log p_{\boldsymbol{\Theta}}(\boldsymbol{y}_t|\boldsymbol{z}_t) p_{\boldsymbol{\Theta}'}(\boldsymbol{z}_t|\boldsymbol{y}_{1:T}, o_{1:T}) d\boldsymbol{z}_t \\
&+ \sum_{t=1}^{T} \int \log p_{\boldsymbol{\Theta}}(o_t|\lambda_t) p_{\boldsymbol{\Theta}'}(\lambda_t|\boldsymbol{y}_{1:T}, o_{1:T}) d\lambda_t. \qquad (5.13)
\end{aligned}
$$

The integrals in the previous equations are intractable and cannot be calculated analytically. Instead, an approximate approach using Bayesian filtering is considered. More specifically, a *hybrid particle filter* (HPF) is used to numerically approximate function $\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}')$. The HPF represents an expansion of the particle filter that was

shown in Section 3.3 with multiple latent and/or observable state processes. The posterior distribution of the latent variables can be computed by a weighted sum of N_s samples drawn from the posterior distribution as

$$p_{\boldsymbol{\Theta}_t}(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}, o_{1:t}) \approx \sum_{i=1}^{N_s} \frac{1}{N_s} \delta_{\boldsymbol{z}_t^i}(\boldsymbol{z}_t) \equiv \hat{p}_{\boldsymbol{\Theta}_t}(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}, o_{1:t}), \tag{5.14}$$

where $\delta_{\boldsymbol{z}_t^i}(.)$ denotes the Dirac delta function and $\boldsymbol{z}_t^i, i = 1, ..., N_s$ are assumed to be independent and identically distributed samples drawn from $p_{\boldsymbol{\Theta}_t}(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}, o_{1:t})$. Since it is impossible to sample from the true posterior, particles $\boldsymbol{z}_t^i$ are instead sampled from a proposal distribution, or importance density function, denoted by $q(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}, o_{1:t})$. Thus, the posterior distribution will be

$$\hat{p}_{\boldsymbol{\Theta}_t}(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}, o_{1:t}) = \sum_{i=1}^{N_s} w_t^i \delta_{\boldsymbol{z}_t^i}(\boldsymbol{z}_t), \tag{5.15}$$

with w_t^i denoting the particle weight associated with particle $\boldsymbol{z}_t^i$, which can be computed recursively as

$$\begin{aligned} w_t^i &= \frac{p(x_{1:t}^i, c_{1:t}^i, \lambda_t^i|\boldsymbol{y}_{1:t}, o_{1:t})}{q(x_{1:t}^i, c_{1:t}^i, \lambda_t^i|\boldsymbol{y}_{1:t}, o_{1:t})} \\ &= \frac{p(\boldsymbol{y}_t|x_t^i, c_t^i, \lambda_t^i)p(o_t|\lambda_t^i)p(x_t^i|x_{t-1}^i, c_t^i)p(c_t^i|c_{t-1}^i)p(\lambda_t^i|c_t^i, x_t^i)p(x_{1:t-1}^i, c_{1:t-1}^i, \lambda_{t-1}^i|\boldsymbol{y}_{1:t-1}, o_{1:t-1})}{q(x_t^i|x_{t-1}^i, c_t^i)q(c_t^i|c_{t-1}^i)q(\lambda_t^i|c_t^i, x_t^i)q(x_{1:t-1}^i, c_{1:t-1}^i, \lambda_{t-1}^i|\boldsymbol{y}_{1:t-1}, o_{1:t-1})} \\ &= w_{t-1}^i \frac{p(\boldsymbol{y}_t|x_t^i, c_t^i, \lambda_t^i)p(o_t|\lambda_t^i)p(x_t^i|x_{t-1}^i, c_t^i)p(c_t^i|c_{t-1}^i)p(\lambda_t^i|c_t^i, x_t^i)}{q(x_t^i|x_{t-1}^i, c_t^i)q(c_t^i|c_{t-1}^i)q(\lambda_t^i|c_t^i, x_t^i)} \end{aligned} \tag{5.16}$$

As the time index t increases, most of the particles are given weights equal to zero due to the skewness of weight distribution. This problem is known as *particle degeneracy* and can be remedied using a resampling step at the end of each step. During this step, the particles are updated using resampling with replacement N_s times from the set z_i^t, with probability of resampling for each particle z_i^k proportional to its weight w_t^i. All particle weights then become equal ($w_{t-1}^i = 1/N_s$). Using prior densities

$p(x_t|x_{t-1}, c_t), p(c_t|c_{t-1})$, and $p(\lambda_t|c_t, x_t)$ as proposal distributions and applying resampling at every t, Eq. 5.16 can be simplified to

$$w_t^i \propto p(\boldsymbol{y}_t|x_t^i, c_t^i, \lambda_t^i)p(o_t|\lambda_t^i), \tag{5.17}$$

Particle weights resemble probability values associated with the likelihood of each particle to represent the latent state. Therefore, some particle weights can become very small and, due to limitations in computational efficiency, may become zero (not to be confused with the zero particle weights due to their distribution). For that reason, the particle weight in Eq. 5.17 can be written in its logarithmic form.

$$\hat{w}_t^i \propto \log(p(\boldsymbol{y}_t|x_t^i, c_t^i, \lambda_t^i)) + \log(p(o_t|\lambda_t^i)), \tag{5.18}$$

Both Eqs. 5.17 and 5.18 are equivalent, that is $\hat{w}_t^i \equiv w_t^i$. The weights can also be artificially increased, without biasing the resampling procedure, by adding to every particle weight value at time t the value of the maximum particle weight at time t:

$$\hat{w}_t^i = \hat{w}_t^i + \max(\hat{\boldsymbol{w}}_t), \hat{\boldsymbol{w}}_t = \{\hat{w}_t^1, \hat{w}_t^2, ..., \hat{w}_t^{N_s}\}, i = 1, ..., N_s.$$

Finally, the weights are returned to their original form:

$$\hat{w}_t^i = e^{\hat{w}_t^i}.$$

After all particles and their weights are obtained, function $\hat{\mathcal{Q}}(\boldsymbol{\Theta}, \boldsymbol{\Theta}_t)$ can be approximated as:

$$\begin{aligned}
&\hat{\mathcal{Q}}(\boldsymbol{\Theta}, \boldsymbol{\Theta}_t) \approx \\
&\sum_{i=1}^{N_s} w_1^i \log p_{\boldsymbol{\Theta}}(c_1^i) + \sum_{i=1}^{N_s} w_1^i \log p_{\boldsymbol{\Theta}}(x_1^i) \\
&+ \sum_{i=1}^{N_s} w_1^i \log p_{\boldsymbol{\Theta}}(\lambda_1^i) + \sum_{t=2}^{T} \sum_{i=1}^{N_s} w_t^i \log p_{\boldsymbol{\Theta}}(c_t^i|c_{t-1}^i)
\end{aligned}$$

$$+\sum_{t=2}^{T}\sum_{i=1}^{N_s} w_t^i \log p_{\boldsymbol{\Theta}}(x_t^i|x_{t-1}^i, c_t^i)$$
$$+\sum_{t=1}^{T}\sum_{i=1}^{N_s} w_t^i \log p_{\boldsymbol{\Theta}}(\lambda_t^i|x_t^i, c_t^i)$$
$$+\sum_{t=1}^{T}\sum_{i=1}^{N_s} w_t^i \log p_{\boldsymbol{\Theta}}(o_t|\lambda_t^i)$$
$$+\sum_{t=1}^{T}\sum_{i=1}^{N_s} w_t^i \log p_{\boldsymbol{\Theta}}(\boldsymbol{y}_t|x_t^i, c_t^i, \lambda_t^i). \tag{5.19}$$

Algorithm 4 presents the hybrid particle filter process that generates a set of N_s particles $(x_{1:T}^{1:N_s}, c_{1:T}^{1:N_s}, \lambda_{1:T}^{1:N_s})$ and their weights.

5.2.1.2 Training Process Initialization

The unknown parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_c, \boldsymbol{\theta}_x, \boldsymbol{\theta}_\lambda, \boldsymbol{\theta}_y\}$ need to be estimated. Typically, these parameters and the state of particles at time zero are initialized either randomly or, if available, with a prior knowledge. However, ELM also requires a prior knowledge of latent states that must be used as inputs to estimate $\boldsymbol{\theta}_y$, and which are unavailable at time zero. Therefore, in the beginning a modified version of the HPF (Algorithm 4-Part (b)) is used to generate a set of latent states for ELM training. These states, along with the sensor observations and the working condition observations are then used for the first-time training of the ELM. The difference between the modified HPF used for initialization and the actual HPF is that in the former, the important weights are calculated only based on $p(o_t|\lambda_t^i, \boldsymbol{\theta}_o)$. This is mandatory since ELM is initially untrained and thus $p(\boldsymbol{y}_t|x_t^i, c_t^i, \mathbf{u}_t, \boldsymbol{\theta}_y)$ cannot be computed at the beginning. This initialization step provides randomly generated parameters $\{\boldsymbol{\theta}_c, \boldsymbol{\theta}_x, \boldsymbol{\theta}_\lambda\}$ and initialized neuron weights $\boldsymbol{\theta}_y$ through training the ELM network for the first time.

Algorithm 4 (a) Hybrid Particle Filter and (b) modified Particle Filter used for Initialization in the EM algorithm.

Input: Sequences of observations $\boldsymbol{y}_{1:t}$, $o_{1:t}$, parameter vector $\boldsymbol{\Theta}$, number of particles (N_s).

(a) Hybrid Particle Filter

1: **for** $t' = 1$ to t **do**
2: **for** $i = 1$ to N_s **do**
Sample $c_{t'}$, $x_{t'}$, $\lambda_{t'}$ from the followings distributions:
3: $c^i_{t'} \sim p(c_{t'}|c^i_{t'-1}, \boldsymbol{u}_{t'}, \boldsymbol{\theta}_c)$,
4: $x^i_{t'} \sim p(x_{t'}|x^i_{t-1}, c^i_{t'}, \boldsymbol{u}_{t'}, \boldsymbol{\theta}_x)$,
5: $\lambda^i_{t'} \sim p(\lambda_{t'}|x'^i_t, c^i_{t'}, \boldsymbol{u}_{t'}, \boldsymbol{\theta}_\lambda)$
6: **end for**
7: **for** $i = 1$ to N_s **do**
Calculate the importance weights as:
8: $w^i_{t'} = p(\boldsymbol{y}_{t'}|x^i_{t'}, c^i_{t'}, \boldsymbol{u}_{t'}, \boldsymbol{\theta}_y)p(o_{t'}|\lambda^i_{t'}, \boldsymbol{\theta}_o)$
Normalize the importance weights: $\tilde{w}^i_{t'} = \frac{w^i_{t'}}{\sum_{j=1}^{N_s} w^j_{t'}}$
9: **end for**
10: **for** $j = 1$ to N_s **do**
Draw new particles $c^j_{t'}, x^j_{t'}, \lambda^j_{t'}$ with replacement such that:
11: $p(c^j_{t'} = c^i_{t'}) = \tilde{w}'^i_t$, $j \in \{1, ..., N_s\}$
12: $p(x^j_{t'} = x^i_{t'}) = \tilde{w}^i_t$, $j \in \{1, ..., N_s\}$
13: $p(\lambda^j_{t'} = \lambda^i_{t'}) = \tilde{w}^i_t$, $j \in \{1, ..., N_s\}$
14:
Set $w^i_{t'} = \frac{1}{N_s}$ for $i \in \{1, ..., N_s\}$.
15: **end for**
16: **end for**

Output: $c^i_{1:t}$, $x^i_{1:t}$, $\lambda^i_{1:t}$, $w^i_{1:t}$ for $i \in \{1, ..., N_s\}$.

(b) Modified Hybrid Particle Filter for Initialization

All the steps are the same as part (a) except for line 6 where the following formula is used to calculate the importance weights:

$w^i_{t'} = p(o_{t'}|\lambda^i_{t'}, \boldsymbol{\theta}_o)$.

5.2.1.3 M-step in the EM algorithm

During the M-step, Eq. 5.19 is maximized with respect to the current parameter values $\boldsymbol{\Theta}'$, that is

$$\boldsymbol{\Theta}'_{new} = \underset{\boldsymbol{\Theta}}{\operatorname{argmax}}\, \mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}').$$

Eq. 5.19 shows that the terms associated with $\boldsymbol{\theta}_c$, $\boldsymbol{\theta}_x$, $\boldsymbol{\theta}_\lambda$, and $\boldsymbol{\theta}_y$ can be independently maximized in the M-step. Due to the nature and complexity of the terms associated with each group of unknown parameters, different optimization methods are used in each M-step iteration for parameter inference. The parameters included in $\boldsymbol{\theta}_c$ are estimated using the *Nelder-Mead* simplex method, whereas for those in $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_\lambda$, are estimated using the random walk *Metropolis-Hastings* (MH) algorithm. The choice of MH for estimating $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_\lambda$ is due to two main reasons: i) fast convergence and ii) the ability to control parameter estimates by carefully selecting prior distributions. The MH algorithm for these two parameter sets is shown in Algorithm 5. Finally, the ELM is utilized for estimating parameters in $\boldsymbol{\theta}_y$. Estimation of the parameters on each group is conducted while the parameters on the other subsets are kept fixed on their current values. The ELM network is trained in each EM algorithm iteration. Despite the fact that ELM training is computationally much faster than a regular ANN, it is both computationally expensive and unnecessary to use the entire data to train the ELM. For these reasons, only a subset of the training data is used to train the ELM. A random selection of N' ($N' \subseteq N$) training samples takes place in order to be used as the ***ELM training set*** for the ELM (N is the total number of time-series in the training data). The required data for training the ELM in the input layer are degradation states $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_T\}$, operating condition states $\{c_1, ..., c_T\}$, and

Algorithm 5 Random Walk Metropolis-Hastings for Parameter Set $\boldsymbol{\theta} \in \{\boldsymbol{\theta}_x, \boldsymbol{\theta}_\lambda\}$.

Input: An initial vector of parameter values $\boldsymbol{\theta}^0$.
for iteration $t = 1, 2, ...$ **do**
  Generate candidate $\boldsymbol{\theta}^*$ from a symmetric proposal distribution $q(\boldsymbol{\theta}^t|\boldsymbol{\theta}^{t-1})$.
  Calculate acceptance probability:
  $\alpha = \min\left(1, \frac{\hat{\mathcal{Q}}(\boldsymbol{\theta},\boldsymbol{\Theta}'|\boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*)}{\hat{\mathcal{Q}}(\boldsymbol{\theta},\boldsymbol{\Theta}'|\boldsymbol{\theta}^{t-1})p(\boldsymbol{\theta}^{t-1})}\right)$,
  where $\hat{\mathcal{Q}}(\boldsymbol{\theta}, \boldsymbol{\Theta}'|\boldsymbol{\theta}^*)$ and $\hat{\mathcal{Q}}(\boldsymbol{\theta}, \boldsymbol{\Theta}'|\boldsymbol{\theta}^*)$ refer to the terms in Eq. 5.19 that contain parameter set $\boldsymbol{\theta}$, given $\boldsymbol{\theta} = \boldsymbol{\theta}^*$, and $\boldsymbol{\theta} = \boldsymbol{\theta}^{t-1}$, respectively.
  Draw a random number $u \sim \text{Uniform}(0, 1)$.
  if $(u \leq \alpha)$ **then**
    Accept $\boldsymbol{\theta}$ and set $\boldsymbol{\theta}^t = \boldsymbol{\theta}$.
  else
    Set $\boldsymbol{\theta}^t = \boldsymbol{\theta}^{t-1}$.
  end if
end for
Output: A final set of parameter estimates $\boldsymbol{\theta}^t$.

operating inputs $\{\boldsymbol{u}_1, ..., \boldsymbol{u}_T\}$ for N' samples. It should be pointed out that the N' samples employed for ELM training are still used for the rest of the EM algorithm.

5.2.2 Summary of proposed approach for model training

The framework's overall process is summarized here. The training data is characterized by N life trajectories with a sequence of sensor observations $\{\boldsymbol{y}_1^{(i)}, \ldots, \boldsymbol{y}_T^{(i)}\}$ and discrete working status observations $\{o_1^{(i)}, \ldots, o_T^{(i)}\}$, along with known lifetimes $T^{(i)}$ for $i \in \{1...N\}$. At first, N' samples are randomly selected for ELM training, and the initialization step is applied to generate a set of initial state and parameter values. In the E-step, the function $\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}'))$ is updated; then, in the M-step, a new set of parameters is obtained using appropriate optimization methods for each set (see Section 5.2.1.3). The process is repeated for an arbitrarily large number of iterations until a predefined convergence criterion is met. The overview of the model training framework is presented in Fig. 20.

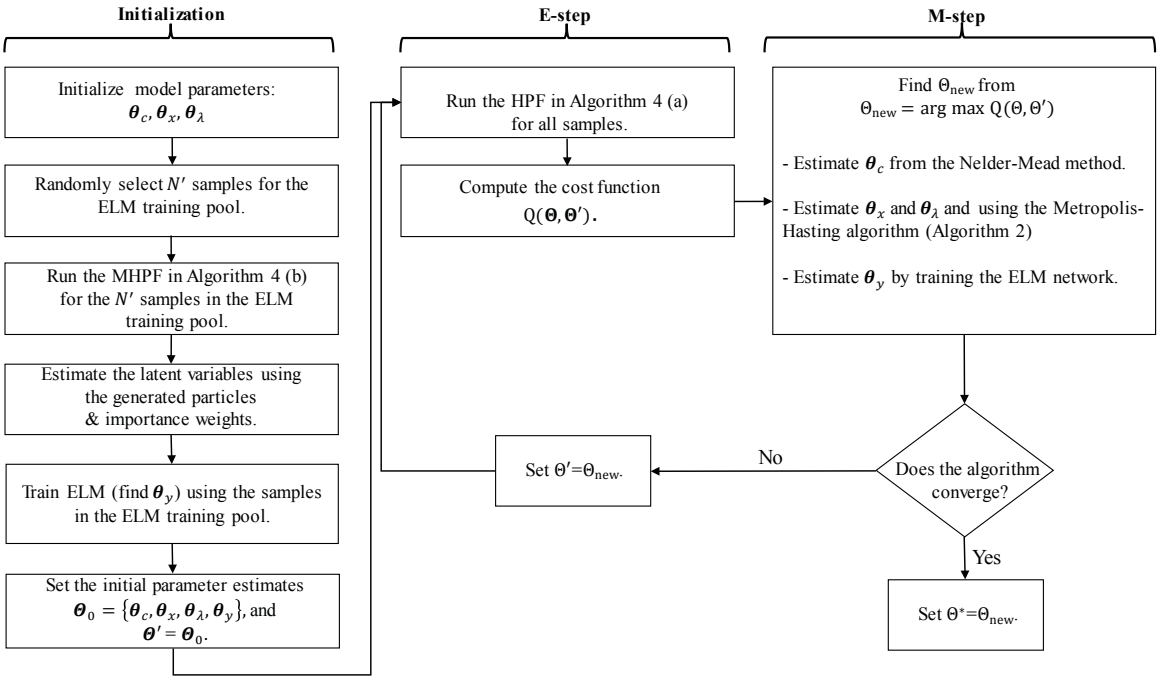


Figure 20: Flowchart of the proposed model training framework.

5.3 Remaining Useful Life estimation

A random variable r_t is defined, denoting the conditional RUL at time t given survival up to time t and past sensor measurements $\boldsymbol{y}_1, ..., \boldsymbol{y}_t$. Using the HPF at time t, particle values $x^i_{1:t}$, $c^i_{1:t}$, and $\lambda^i_{1:t}$, for $i \in \{1, ..., N_s\}$ are obtained. The age of the system, calculated at time t based on particle i, is characterized by a variable l^i_t, and it can be determined by projecting particle values into the future for time $t + \tau, \tau \geq 1$, until system failure. Each particle has a failure point that can be obtained using the projected values of $\lambda^i_{t+\tau}, \tau \geq 1$, since they represent the probability of failure. After particle $\lambda^i_{t+\tau}$ is projected at time $t + \tau$, a random number ψ is drawn from a uniform distribution $\mathcal{U}(0, 1)$, and if $\psi \leq \lambda^i_{t+\tau}$, particle i has reached its failure point and, thus $l^i_t = t + \tau$. This estimate is associated with an uncertainty that

stems from the particle weight calculated at time t (w_t^i). It should be emphasized that particle projections in future times are conducted using the trained transition functions $f_{\cdot}$. The process repeats for a predefined number of times N_{iter}, in order to accommodate for the randomness of future particle projections. Denoting $l_t^{i,n}$ as the age of the system computed at time t based on the ith particle and the nth iteration, the expected value of the remaining useful life can be calculated as

$$\mathbb{E}\{r_t\} = \frac{1}{N_{iter}} \sum_{n=1}^{N_{iter}} \sum_{i=1}^{N_s} w_t^i \, l_t^{i,n} - t. \tag{5.20}$$

The probability distribution of the conditional remaining useful life can be calculated as

$$\begin{aligned} p(r_t = \tau) &= p(o_{t+\tau} = 1, o_{t+\tau-1} = 0 | \mathbf{y}_t, o_t = 0) \\ &= \frac{1}{N_{iter}} \sum_{n=1}^{N_{iter}} \sum_{i=1}^{N_s} w_t^i \, \mathbf{1}_{\{l_t^{i,n} = t+\tau\}}, \quad \tau \geq 1. \end{aligned} \tag{5.21}$$

Algorithm 6 gives a step-by-step presentation of the procedure to calculate the distribution of the conditional RUL.

5.4 Numerical Experiments

The validity of the framework was tested using multiple numerical experiments on both synthetic and real world data. Section 5.4.1 is based on synthetic data and Section 5.4.2 is based on the C-MAPSS dataset obtained from the NASA Prognostics Data Repository [162].

Algorithm 6 RUL and its probability distribution computed at time k.

Input: Parameter vector Θ, number of iterations N_{iter}, number of particles N_s, and sensor measurements $\boldsymbol{y}_1, ..., \boldsymbol{y}_t$.
for $i \in \{1, ..., N_s\}$ **do**
Run the developed HPF and obtain particle values $\boldsymbol{z}_t^i$ and their weights w_t^i.
end for
Set $\tau = 1$.
for $n \in \{1, ..., N_{iter}\}$ **do**
  for $i \in \{1, ..., N_s\}$ **do**
    while $o_{t+\tau-1}^{i,n} = 0$ **do**
      Calculate the projected values $\boldsymbol{z}_{t+\tau}^i$.
      Randomly generate ψ from $\mathcal{U}(0, 1)$.
      if $(\psi \leq \lambda_t^{i,n})$ **then**
        The system is considered failed.
        $l_t^{i,n} = t + \tau$ and $o_t^{i,n} = 1$.
      else
        $o_t^{i,n} = 0$.
      end if
      $\tau = \tau + 1$.
    end while
  end for
end for
Estimate the expected RUL from Eq. 5.20.
Estimate the probability distribution of RUL from Eq. 5.21.
Output: The probability distribution of the Remaining Useful Life.

5.4.1 Simulation Experiments

Following a similar approach for generating synthetic data as the previous Chapter, a fully stochastic framework was designed to simulate data for model evaluation, and multiple trajectories of run-to-failure samples were generated for a single-unit degrading system according to the structure given in Eqs. 5.1-5.5. The simulated time-series assumed a one-dimensional latent degradation process x_t based on the health index presented in [162, 158, 163], slightly modified to incorporate all latent variables and operating conditions. The evolution of the system over time is represented by a binary mode process c_t at time t, a $2-D$ observation process $\boldsymbol{y}_t$, a $1-D$

operating input u_t, a $1-D$ hazard process λ_t, and a $1-D$ working condition process o_t. The equations of the corresponding HSSM are presented below:

$$p(c_t = j | c_{t-1} = i) = p_{ij}, \tag{5.22}$$

$$x_t = x_{t-1} - e^{(-b_{c_t})} - g \cdot u_t + v, \tag{5.23}$$

$$\boldsymbol{y}_t = \mathcal{N}(\boldsymbol{H}^{c_t} x_t, \boldsymbol{R}), \tag{5.24}$$

$$\lambda_t = \frac{1}{1 + e^{-(\alpha x_t + \beta_0)}}, \tag{5.25}$$

$$\Pr(o_t = o) = \begin{cases} 1 - \lambda_t, & \text{if } o = 0 \\ \lambda_t, & \text{if } o = 1 \end{cases}, \tag{5.26}$$

where $\boldsymbol{H}^{c_t}$ is a $\{2 \times 1\}$ vector that maps the (true) latent state x_t into the observed space depending on the operating mode at time t. The variable v represents the health index noise that follows a normal distribution with zero mean and variance q^2. Markov transition matrix $\mathbf{P}_{ij}$ denotes the transition probabilities from mode i to mode j. Variable $\boldsymbol{R}$ represents the covariance matrix for the multi-dimensional sensor observation process, which also follows a multivariate normal distribution. Parameters b_{c_t} and g characterize the latent degradation state's transition and α and β_0 characterize the hazard process. The simulation process follows the assumptions used in the C-MAPSS dataset, that is, when a system transits to the faulty operating condition (mode), it remains there for the rest of its lifetime. For that reason, transition matrix $\mathbf{P}$ is expressed as

$$\mathbf{P}_{ij} = \begin{bmatrix} p_{11} & 1 - p_{11} \\ 0 & 1 \end{bmatrix}, \forall k. \tag{5.27}$$

The true values chosen for the parameter vector $\boldsymbol{\Theta}$ are given below:

$$b_0 = 9, b_1 = 0.9, g = 0.001, \boldsymbol{H}^0 = \begin{bmatrix} 2.5 \\ 2.72 \end{bmatrix}, \boldsymbol{H}^1 = \begin{bmatrix} 1.76 \\ 1.09 \end{bmatrix}$$

$$p_{11} = 0.9q = 0.001, \alpha = -30, \beta_0 = 3, \mathbf{R} = \begin{bmatrix} 0.09 & 0 \\ 0 & 0.09 \end{bmatrix}.$$

The simulation process starts from initial values $c_o = 0$ and $x_0 = 10$ (i.e., the system is at the as-good-as-new condition) and continues as follows: at time t, the mode c_t is generated by Eq. 5.22, followed by the health index x_t (Eq. 5.23), and the observation signal $\boldsymbol{y}_t$ (Eq. 5.24). Then, the probability of failure λ_t is calculated and a random number $\zeta_t \sim \mathcal{U}(0, 1)$ is drawn. If $\zeta_t \leq \lambda_t$, the system fails ($o_t = 1$), otherwise $o_t = 0$ and the process continues. The system starts its operation normally, and at some point, a fault occurs (the operating mode transits to $o_t = 1$).

5.4.1.1 ELM Performance

The ability of the ELM in mapping latent states to sensor observations (Eq. 5.24) is demonstrated here. A dataset of $N = 50$ samples was generated, with $N' = 40$ of them selected at random to be used for ELM training (see Section 5.2.1.3). Fifty neurons were chosen for the ELM's hidden layer. For comparison purposes, the latent states using Eq. 5.24 were also estimated using the HPF. Estimates for both degradation and mode states regarding two of the degrading systems can be seen in Fig. 21, where ELM and the observation process from Eq. 5.24 were used. The true latent state values are shown with solid lines. It can be seen that using ELM as the observation process performs almost the same as using the true f_y formula. Fig. 22 presents the mean squared error (MSE) between the mean values of true and

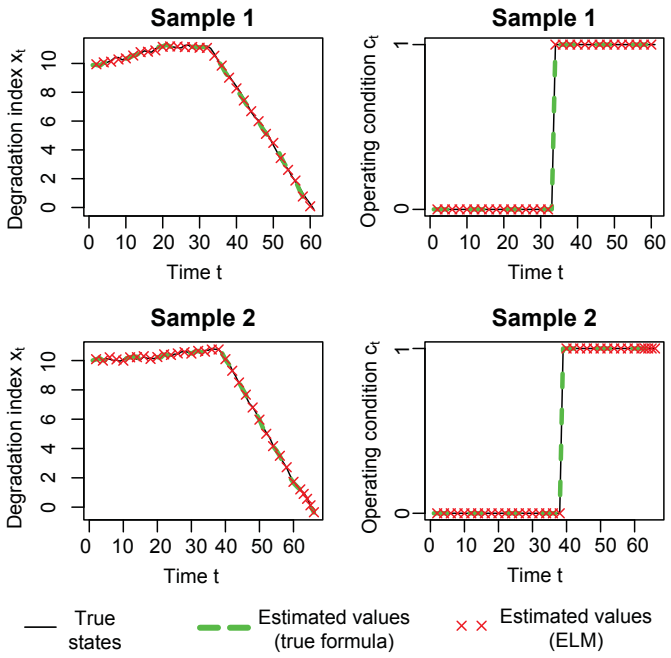


Figure 21: Comparison between true latent states x_t (left column) and c_t (right column) shown by the solid lines and and their estimates using the true formula for f_y and ELM for two random samples.

estimated observations that are obtained after each EM iteration for both training (N' samples) and testing sets ($N - N'$ samples). ELM converges very fast and provides relatively low MSEs in both sets. The results indicate that a properly tuned ELM can provide a reasonable non-parametric equivalent for the observation process f_y. Fig. 23 shows that the estimates for the observation measurements $\hat{\boldsymbol{y}}$ obtained from the ELM are almost identical to those obtained using Eq. 5.24 and their actual values. This outcome suggests a significant accomplishment that addresses the challenge of finding a true parametric distribution for the observation process.

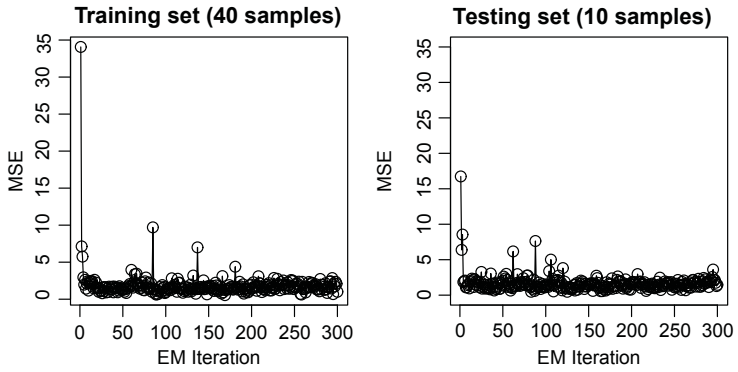


Figure 22: MSE of true v.s. estimated $\boldsymbol{y}$ for samples in the training set (left) and testing set (right).

5.4.1.2 Parameter Estimation

For the rest of the experiments, 100 samples for model training and 900 for testing were generated. From the first $N = 100$ samples that were used specifically for parameter estimation, a subsample $N' = 20 \subseteq N$ was randomly selected for ELM training. In order to better capture the data variance, a 5-fold cross-validation was employed for selecting the ELM training samples N' and the parameter estimation results were averaged over all folds. The random-walk MH was used for estimating the unknown parameters b_0, b_1, g, α, and β_0 using uniform priors. The EM algorithm ran for 300 iterations, and on each iteration the MH algorithm was executed for 1,000 iterations. From the total 300,000 MCMC iterations ($300 \times 1,000$), the first 250,000 were considered as *burn-in* time. Table 5 shows the mean parameter estimates, their standard deviation, and the root mean squared error between true and estimated values. Results show that the estimated values are very close to the true values and the variations between parameter estimates are low.

The process of parameter estimation was also conducted using randomly generated latent states x and c based on their true stochastic distributions and the random

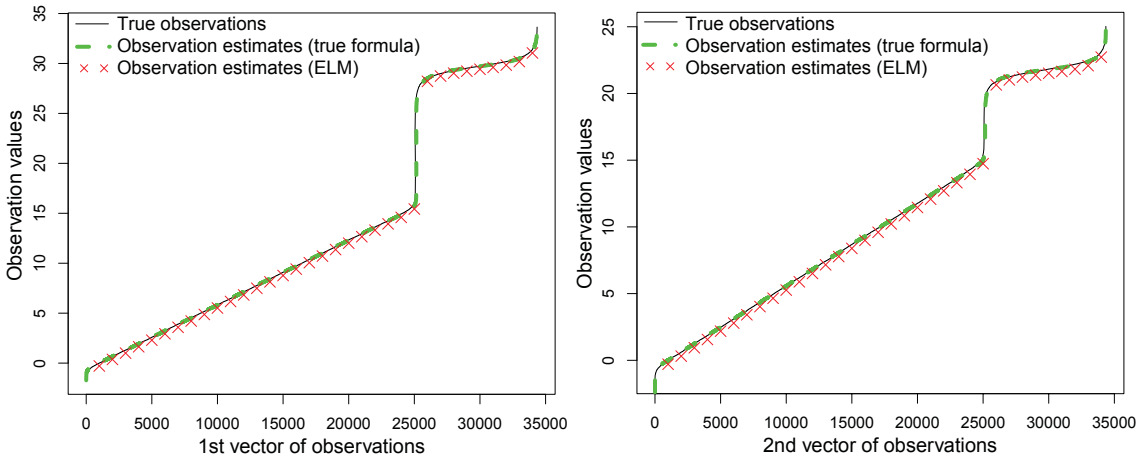


Figure 23: Comparison between the true 2-dimensional $\boldsymbol{y}$ and its estimates obtained from ELM and the known formula given in Eq. 5.24. The values are sorted based on the true $\boldsymbol{y}$.

initial parameters (instead of the modified HPF presented above), for comparison purposes. Results verified that the ELM algorithm took 15 fewer iterations to converge, while parameter estimates were relatively better than the ones obtained from random initialization.

5.4.1.3 RUL Prediction

The RUL estimation results of the remaining 900 simulated samples is demonstrated here. Note that the true lifetimes of these 900 samples are known, and the purpose is to prove that the RUL estimation procedure presented in Section 5.3 can yield results that closely approximate the true lifetimes. RUL predictions were made at multiple points during the lifetime of each sample, starting from when only 5% of true lifetime was left, and ended at 90% of the system's age, on 5% increments. Fig. 24, provides boxplots comparing the true vs. the estimated RULs in true lifetime percentages. It can be seen that percentages of the estimated RULs are approximating the true ones when the predictions are made closer to system failure (e.g., at

Table 5: Parameter estimation results.

	ELM Initialization					Random Initialization				
Measure	$\boldsymbol{b_0}$	$\boldsymbol{b_1}$	$\boldsymbol{g}$	α	β_0	$\boldsymbol{b_0}$	$\boldsymbol{b_1}$	$\boldsymbol{g}$	α	β_0
True	9	0.9	0.001	-30	3	9	0.9	0.001	-30	3
Estimate	8.95	0.89	0.003	-31.6	2.94	8.54	0.53	0.009	-28.04	1.002
SD	0.585	0.03	0.002	2.8	0.58	0.81	0.021	0.001	0.23	0.24
RMSE	0.586	0.029	0.003	2.9	0.59	0.91	0.37	0.009	1.98	2

5%-15%). The predictions deviate the earlier they are made, which is intuitively reasonable since the level of uncertainty regarding future system degradation is higher at the early stages of system operation (i.e., less condition monitoring data available). In real-world applications, the true remaining life is unnecessary for RUL prediction, which can be made at any time instance throughout the life cycle. The only data needed for prediction is the observable sensor data up to the point where the prediction is made.

5.4.2 Application in the C-MAPSS Dataset

The same experiments were carried out using one of the NASA C-MAPSS dataset, which was developed in the NASA Ames Research Center for simulating turbofan engine degradation [163]. The dataset contains 260 multidimensional time-series training data for a $90,000$ lb thrust class turbofan engine, with 21 observation features ($\boldsymbol{y}$) and three operating inputs ($\boldsymbol{u}$). Each engine begins its function having an unknown level of initial wear and suffers from high-pressure compressor degradation.

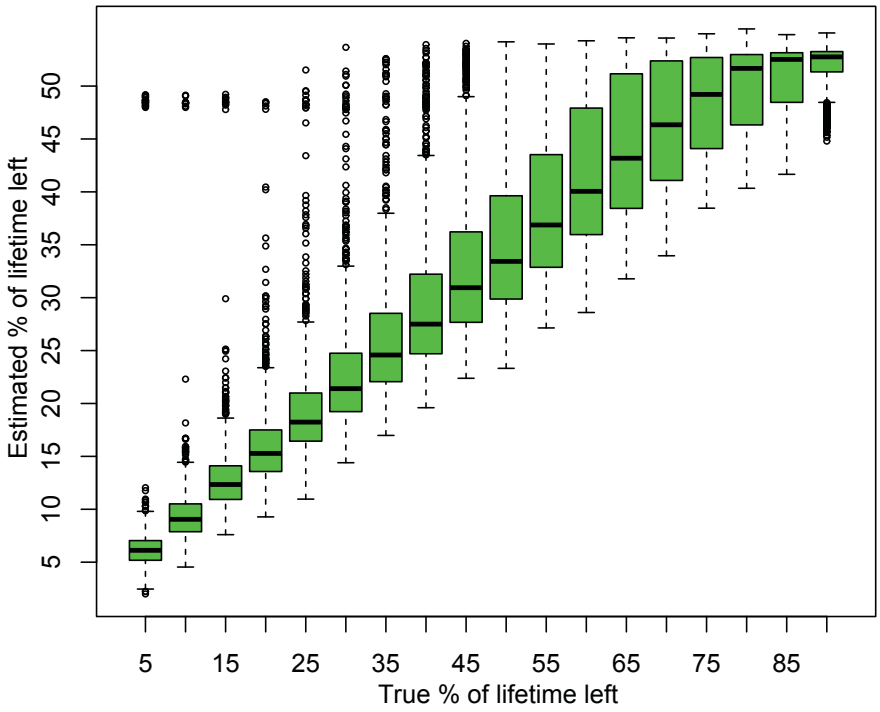


Figure 24: Boxplots of true and estimated percentage of remaining lifetime (900 engines).

5.4.2.1 Parameter Estimation

A random selection of $N = 100$ engines for training and 160 for testing was made. From the 100 engines in the training dataset, $N' = 20$ engines were also used for training ELM, again using a 5-fold cross-validation and averaging the results over all folds. Fifty neurons were used for the ELM hidden layer. The following assumptions were made for the rest of the numerical experiments:

- The transition matrix shown in Eq. 5.27 holds for the mode process.
- The engines are assumed to be functioning without faults for at least 50% of their lifetimes. This assumption was made for numerical stability and to avert any transitions from the healthy state to the faulty state early at a system's lifetime. No changes in the results were observed after substituting 50% to

values 5%-90% with 5% increments. However, for extreme cases (i.e., 1% or 99%), the mode process c estimation became biased towards either 0 or 1 for all time instances. In real applications, a fixed number based on past experience (preferably low) can be used as a lower point where the fault transition can happen.

- The observations were normalized between 0 and 1 to accommodate high variance between different sensor outputs.
- The observations for each engine were averaged every three cycles.
- The parameters of the model are $\boldsymbol{\theta}_c = \{p_{11}\}$, $\boldsymbol{\theta}_x = \{b_0, b_1, g_1, g_2, g_3\}$, and $\boldsymbol{\theta}_\lambda = \{\alpha, \beta_0\}$.

Uniform priors for the random-walk MH were used for parameters $\boldsymbol{\theta}_x$=$\{b_0, b_1, g_1, g_2, g_3\}$. The EM and the MH ran for 10 and 150,000 iterations, respectively, and the first 130,000 MH iterations served as *burn-in time*. Fig. 25 presents the MH samples for the parameters of $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_\lambda$ on the last 20,000 MH iterations. The mean estimated values are:

$$\hat{\boldsymbol{\Theta}} = \{0.899, 3.48, 1.99, 0.012, 0.022, 0.052, -24.8, 4.05\}.$$

The results show relatively good mixing and convergence. Fig. 26 shows the estimated x_t and c_t for three sample engines. The results in all engines show that the systems develop a fault at some time point during their operation and fail when the degradation index x_t approaches zero. It can also be seen that the degradation index is monotonically decreasing. Hence, it can be concluded that the trained HSSM can reasonably represent the latent degradation and operating mode states.

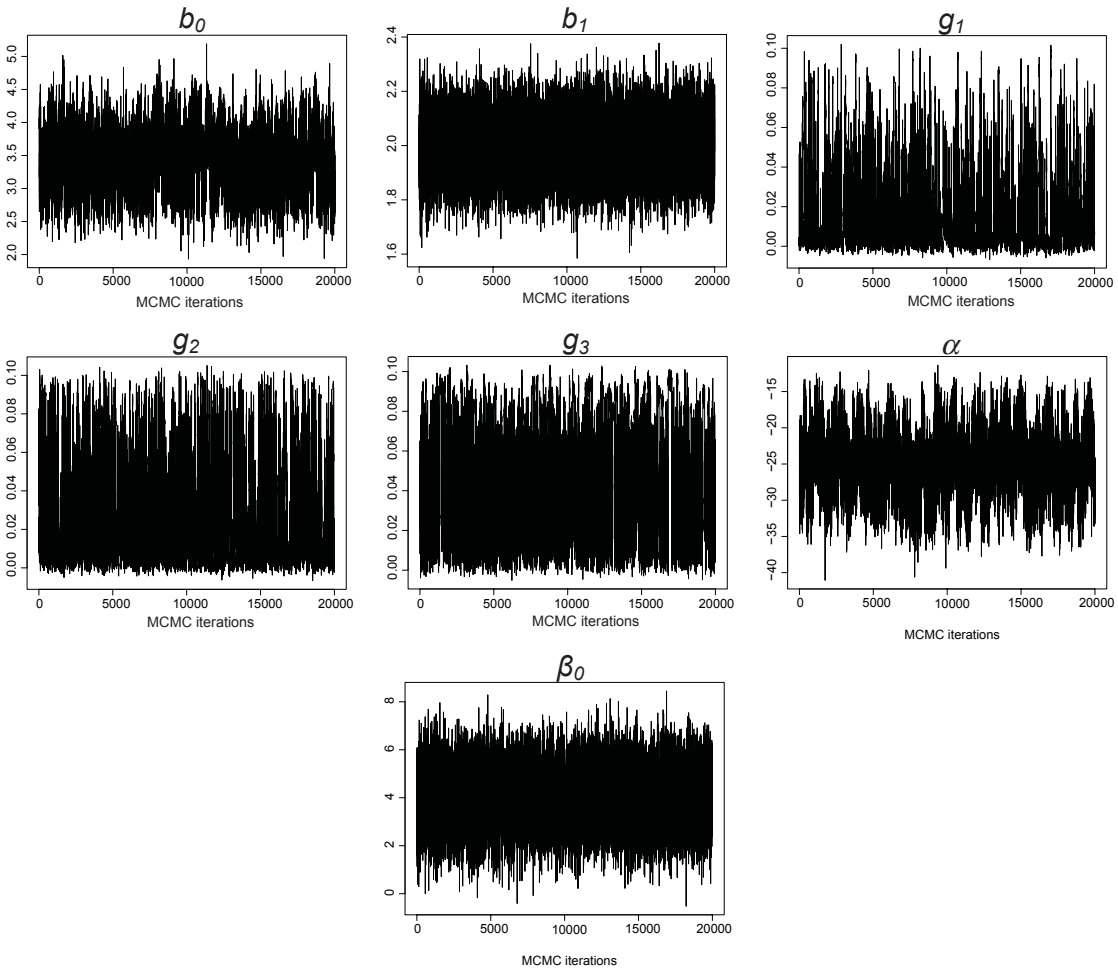


Figure 25: The MCMC samples for the parameters for $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_\lambda$.

5.4.3 RUL Prediction

Fig. 27 shows the comparison between true and estimated RULs for 12 randomly selected engines from the testing set. The results indicate that the model can provide very good estimates for the remaining useful life, particularly when the prediction is made closer to the failure point. Table 6 reveals the estimated mean RUL, the root mean squared error (RMSE) of estimates, and their standard deviation (SD), which are made when the true RUL was in the range of 1 to 15 cycles before failure for a

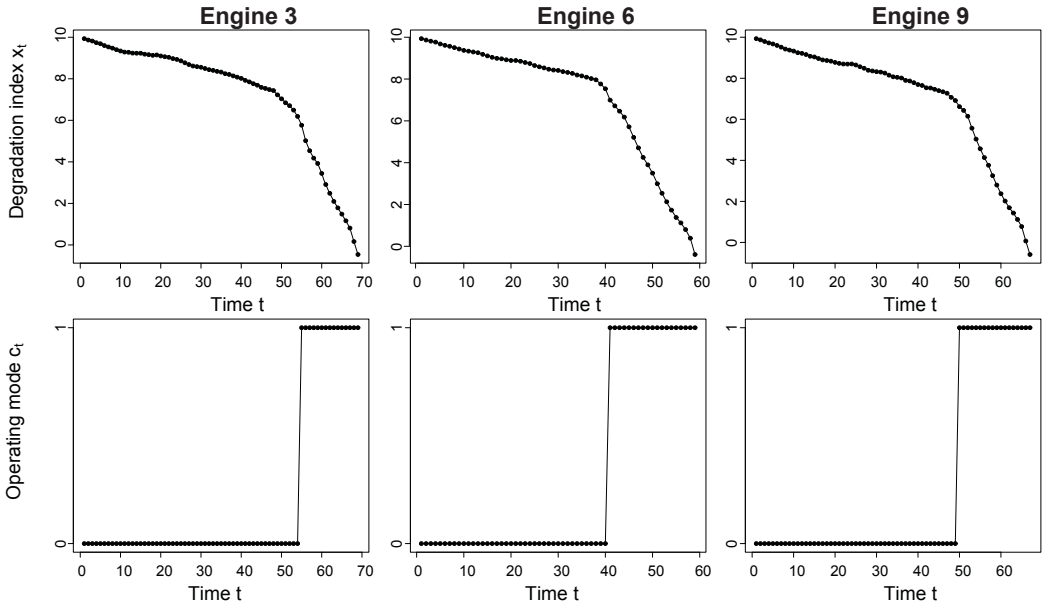


Figure 26: The estimates for the latent states using our model for 3 sample engines.

subset of 100 engines in the testing set. The estimates for mean RUL are close to the true values, especially when the estimations are made very close to the failure point. The decrease in the RMSE and SD values also suggests a low level of uncertainty. The boxplots of RUL estimates together with the true RUL are presented in Fig. 28. Results show that the model can provide reasonable RUL estimates using sensor measurements.

5.4.4 RUL Accuracy and Comparison with Other Models

The C-MAPSS dataset RUL estimates were compared with the same estimates obtained from applying the framework presented in Chapter 4 (**Model a**), and a model with similar structure to the HSSM presented in this Chapter, assuming a multivariate normal distribution for Eq. 5.4 (**Model b**). Neither model used ELM for mapping latent states to sensor observations. The boxplots in Fig. 29 indicate

Table 6: Means and standard deviations (SD) of estimated RUL.

True RUL	Mean Est. RUL	RMSE Est. RUL	SD Est. RUL
15	17.37	5.14	4.58
14	16.32	5.18	4.65
13	15.04	5.06	4.65
12	14.24	5.51	5.06
11	12.92	5.27	4.94
10	11.64	5.13	4.89
9	10.59	4.51	4.24
8	9.23	4.32	4.17
7	8.42	4.58	4.37
6	7.27	4.46	4.29
5	5.98	4.06	3.96
4	5.01	3.75	3.63
3	4.17	3.57	3.38
2	3.20	2.59	2.31
1	2.04	2.38	1.68

that the proposed framework provides much better estimates due to its ability to address more complex system dynamics with its multi-level generative structure.

5.4.5 Computational Complexity

The most important drawback of this framework is its computational complexity. As was shown in Section 5.2, the EM algorithm is a combination of time-intensive steps. This is especially true during the M-step, where the Metropolis-Hastings algorithm, which is utilized for the parameter sets of the degradation and the hazard processes, requires tens thousands of steps on every EM iteration. All these processes run for a predefined number of EM iterations. For example, for the CMAPSS data, the EM algorithm ran for 10 iterations and the Metropolis-Hastings algorithm ran 15,000 times for both parameter sets $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_\lambda$, for a total $150,000 + 150,000 = 300,000$ Metropolis-Hastings iterations. The model required 8.6 hours of CPU time to finish

training for a medium-sized dataset. This long training time could be due to the fact that there was no feature selection or any other data preprocessing steps. The extra noise introduced by features that do not otherwise contribute to the provided information, or to the model in general, can increase the amount of time for model training significantly. That makes the model application expensive, therefore reducing its operational spectrum only to large and complex systems with very high costs of maintenance.

For comparison purposes, the total number of Metropolis-Hastings iterations for the simulated dataset was $300,000 + 300,000 = 600,000$. The model in this case required 8.7 hours of CPU time for training. While this is almost equal to the CMAPSS case, the number of Metropolis-Hastings iterations was doubled here. Since the simulated dataset had only two-dimensional sensor observations, it can be derived that a lower complexity of the input data can lead to a significant reduction of model training time. Future work must focus on feature selection and feature generation in order to reduce the training time and to make the model suitable for less complex systems.

5.5 Summary

The framework on this Chapter represents an upgraded version of the model presented in Chapter 4, which can analyze systems controlled by any kind of dynamics, either linear or non-linear. Its fully hierarchical structure makes it able to take into consideration multiple levels of dynamics. It also relaxes unrealistic parametric assumptions commonly made to define the stochastic relationship between latent states

and sensor observations. Furthermore, a hazard process was defined that removes the need for predefined failure thresholds.

An iterative procedure was designed for an unsupervised model training and RUL estimation. The results obtained from numerical experiments on synthetic and real sets demonstrate the ability of the proposed framework for model training and accurate estimation of the system's latent dynamics, as well as its remaining useful life. This framework represents Phase I of the proposed structure, and its outputs will be used as inputs for the control and decision-making approach described in the next Chapter.

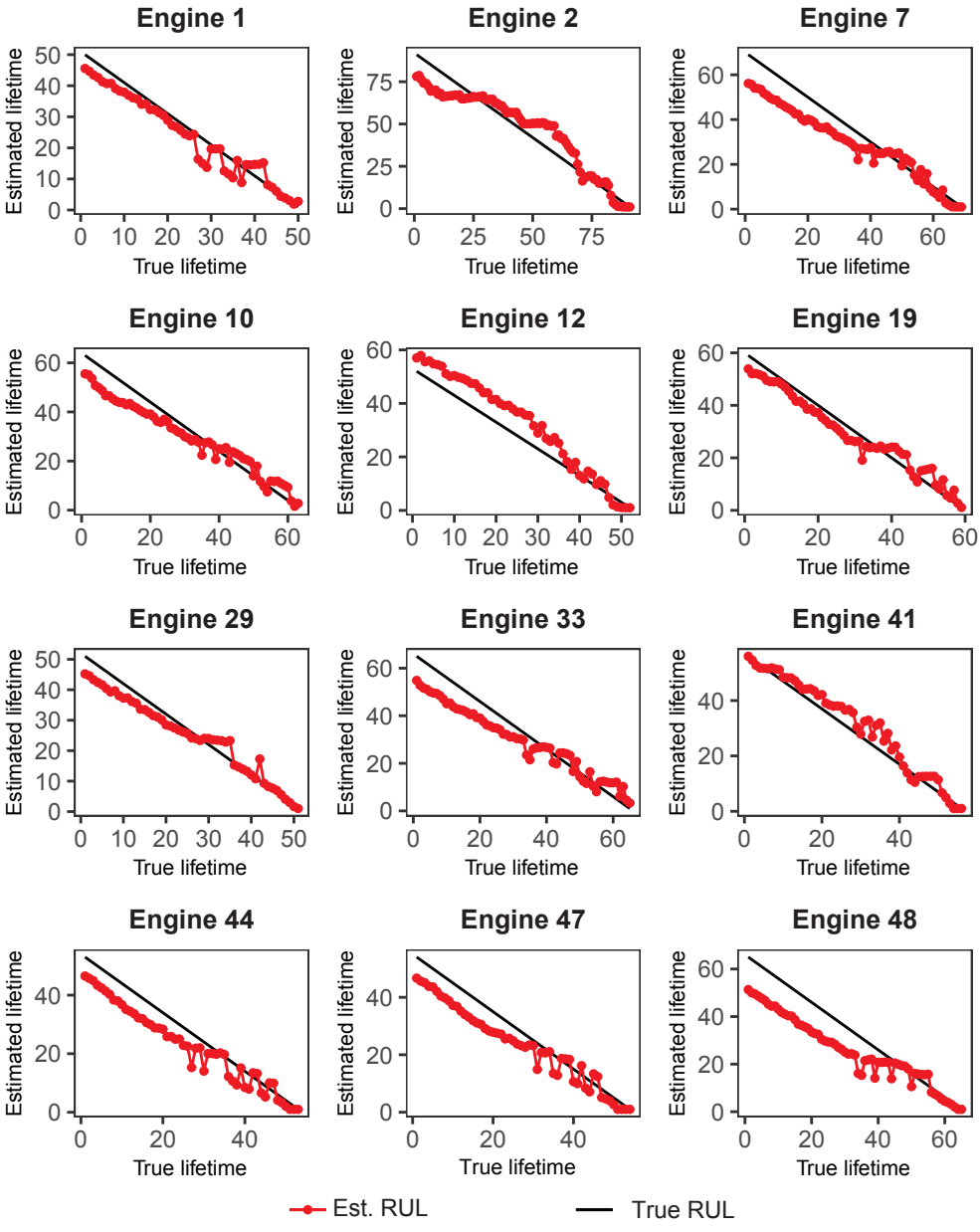


Figure 27: True v.s. estimated RULs for 12 engines.

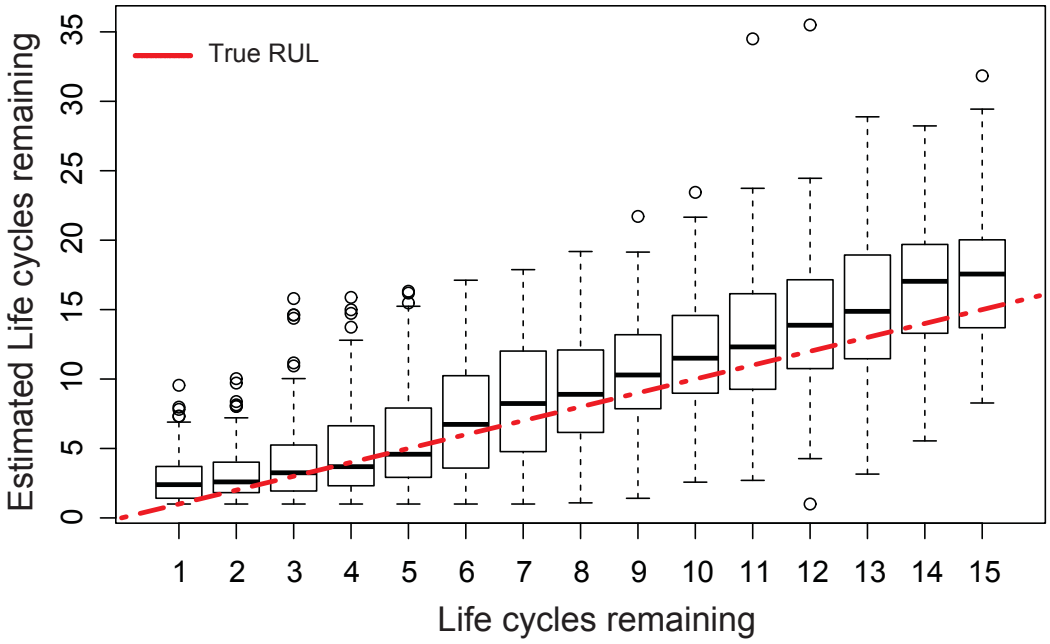


Figure 28: The performance of the model in terms of RUL prediction in the testing set. The dashed line represents the true life cycles remanining, and the boxplots are associated with the estimated RULs for 100 engines in the testing set.

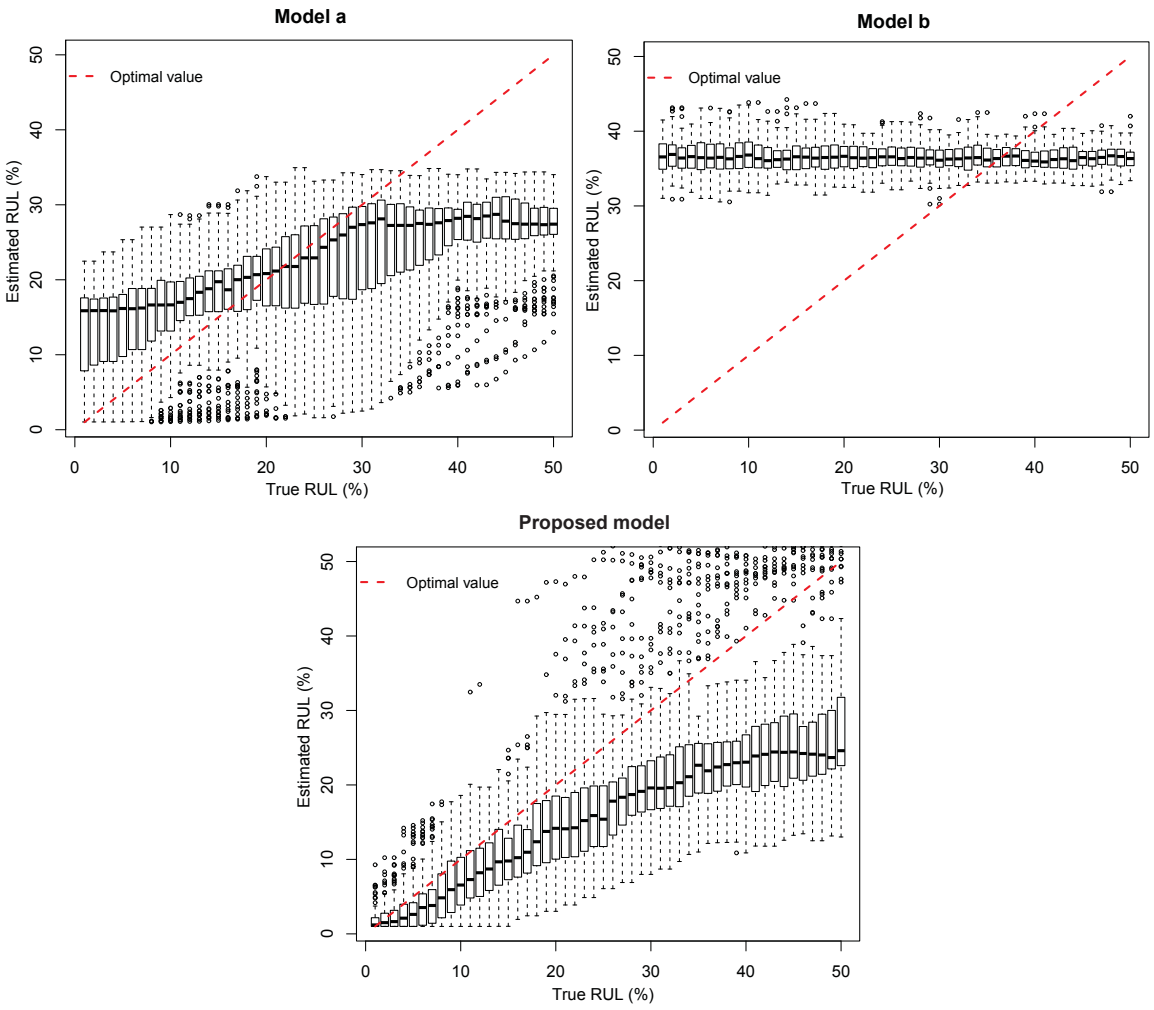


Figure 29: The comparison between true and estimated RUL made at the last 50% of the engines' lifetimes using the proposed model, Model a, and Model b. The dashed line represents the true RUL (%), and the boxplots are associated with the estimated RULs (%) in the test set. The proposed method produces much better results, particularly where the estimation is made closer to the failure point.

CHAPTER 6

Deep Reinforcement Learning for Real-time Decision-Making

Phase II of the proposed Bayesian filtering-based structure describes an original method for real-time decision-making based on recent advancements in deep reinforcement learning (*reinforcement learning + deep neural networks*). As explained in the previous two chapters, the development of real-time control and decision making policies is challenging for complex systems that are monitored by many sensors where only imperfect information describing their latent dynamics can be observed over time. The purpose of the presented framework is to demonstrate the potential of deep reinforcement learning (DRL) for a new generation of decision-making methods for complex systems.

DRL agents can learn independently to establish successful optimal policies for gaining maximum long-term rewards. Although applications that have employed DRL have shown very promising results, sensor-driven maintenance decision making based on DRL has been rarely studied and deserves more research in order to be fully applicable and beneficial for condition-monitored degrading systems. The DRL frameworks presented in this Chapter are able to utilize the outputs from particle filtering described in Chapter 5 and transform them to the maintenance action space.

Two decision-make frameworks based on using Bayesian filtering and DRL in an intelligent and coordinated manner were developed. The first framework uses sensor data to determine when to perform maintenance prior to a system failure based on the relative relationship between the costs of replacement and failure. The second framework shows a prognostics method for remaining useful life estimation that can generate warnings based on the relative relationship between early and late warnings. The original inputs for both frameworks are multi-dimensional sensor data stochastically related to the system's latent degradation state. The frameworks do not depend on the structure of the system's dynamics and have no prior distributional assumptions, making them very generic and thus more applicable to a wide range of degrading systems.

6.1 Bayesian Filtering-based DRL for Maintenance Decision Making

Section 6.1.1 illustrates the procedure that Bayesian filtering can be used for to transform sensor data to the inputs required by DRL, and how to train a DRL agent for a specific task. Section 6.1.2 explains the process of training deep neural networks using limited amount of training data. Section 6.1.3 describes a real-time system control framework in which an agent is trained to provide the best time to replace faulty equipment that minimizes the average maintenance cost. Finally, Section 6.1.4 presents a first-of-its-kind decision making framework for RUL estimation, based on a framework that utilizes particle filtering and DRL while taking into account the relative importance of early and late estimations.

6.1.1 From Bayesian Filtering to DRL

Since the latent states of the system dynamics are not directly observable, it is not possible to derive any decision-making policy that can directly map latent states to the action space. The multivariate sensor observations collected during system operation and the binary system working condition are the only observable inputs for any decision-making task. The decision policy can be defined over the belief space, that is, the probability distribution of latent states. The belief state at any time point can be computed from the vector of observation history using particle filtering. The outcome of particle filtering in Eq. 5.15 is a set of weighted particles that approximates a posterior belief about the latent system dynamics. The set of all particles at the beginning of any decision epoch can estimate the probability distribution of the latent states. Each particle vector represents a point estimate of the latent state $\boldsymbol{z}_t$. Since all particles at time t can represent the probability distribution of the hidden state $\boldsymbol{z}_t$, given observations $Y_{1:t}$ at that time, a discrete probability distribution can be created. First, the particle domain is divided to B intervals with discrete bounds $b_0, b_1, ..., b_B$. Intervals are assumed to be mutually exclusive and of equal length, covering the entire set of values that the particles can take. The values of b_0 and b_B may be selected as the lowest and highest possible values of particles. Then, the dth element of $\boldsymbol{\zeta}_t$ is defined as follows:

$$\zeta_t^d = \frac{\sum_{n=1}^{N_s} \mathbf{1}_{\{z_t^n \in (b_d, b_{d-1}]\}}}{N_s}.$$

Based on the above transformation at every time point t, a discrete particle distribution is obtained with values $\zeta_t^1, ..., \zeta_t^B$ to be used as the system's *belief state*. It should be emphasized here that this quantization occurs separately for the particles of every hidden process in the state-space model. In other words, discrete distributions for

all latent processes are considered. The number of bins should be cross-validated according to the computational complexity and the ability to represent the particle distributions. Multiple bins should be defined for the continuous degradation process and hazard process while only two bins for the operating condition state process are needed. Figure 30 provides a simple example of how particles can generate the empirical probability distributions to be later used as an input for the $\mathcal{Q}$-function approximator neural networks. The multivariate observation process is first analyzed

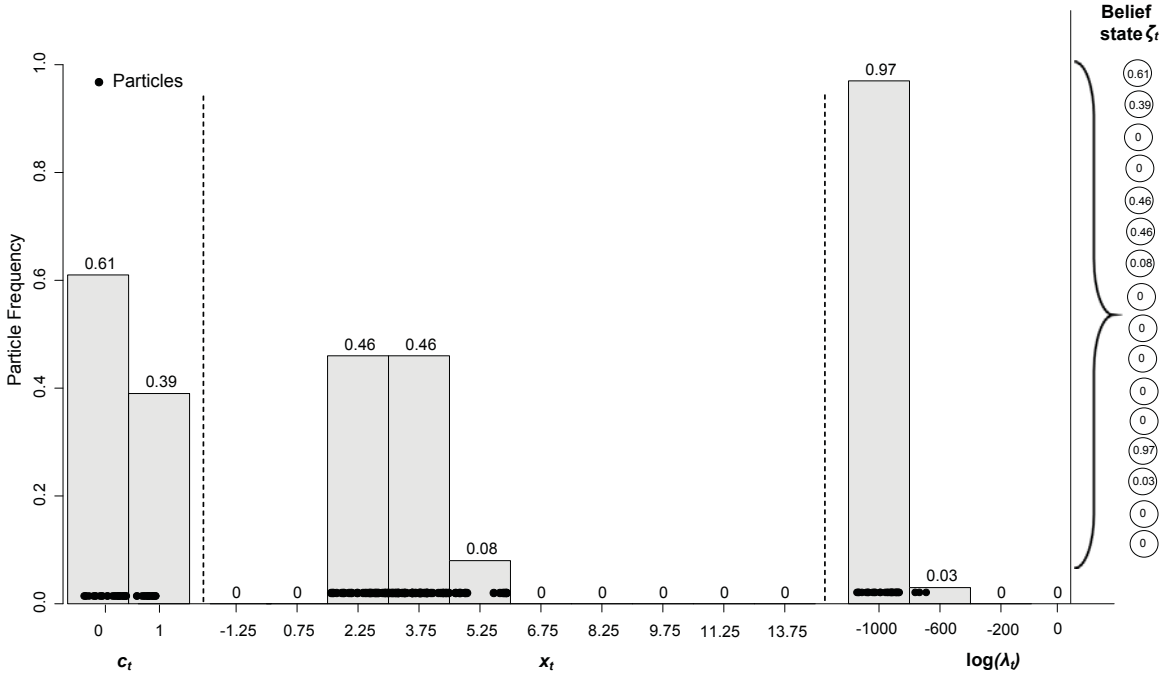


Figure 30: Discretization of particle distribution. The vdots represent the actual particle values. The bars present the empirical probability of the corresponding range. These stochastic values are then used as part of the inputs needed for neural networks (e.g., belief state).

through the hybrid particle filter to estimate the posterior beliefs of the latent system dynamics. These estimates are then discretized to B bins (as discussed above), providing the belief states of the latent system dynamics. The result is a training dataset $\mathcal{I} = \boldsymbol{T} \times B$, with $\boldsymbol{T} = \sum_{i=1}^{N} T_i, i \in \{1, 2, ..., N\}$ denoting the sum of all lifetimes in N. Dataset $\mathcal{I}$ contains the cardinality of all the environment states from all N systems

that are used as inputs to train the DRL agent in the proposed framework. Each state is also associated with a set of possible control actions $\mathcal{A}$, which also act as training inputs. The agent compares the action-value function $\mathcal{Q}$ for every combination of state and possible actions and chooses the one that gives the best cumulative future reward for that state. Algorithm 7 provides the steps on how to train the DRL agent using some historical data recorded in the training set.

The first iteration, or *episode*, starts at $t = 1$. The agent action-value function output $\mathcal{Q}$ is evaluated for all combinations of states and Z actions in $\mathcal{A} = \alpha^1, ..., \alpha^Z$ as inputs for the $\mathcal{Q}$-function approximator. This evaluation repeats for all t. After the final $\mathcal{Q}$-value at time $t = \boldsymbol{T}$ is obtained, the first DRL training iteration concludes. The process repeats for a number of iterations until no significant improvement in the agent's behavior is observed (i.e., $\mathcal{Q}$-values converge). The number of episodes is defined by the user and can reach the several hundreds, depending on the complexity of the problem. During the training process, the agent either randomly selects the action or it selects the action that gives the maximum $\mathcal{Q}$-function value using the most updated information available. The exploration step is critical as it ensures the agent explores states that otherwise may not be selected during the exploitation process. The exploitation/exploration learning rate ϵ determines the trade-off for these two types of selection. The training process can be stopped after a certain number of episodes, or when a pre-defined criterion, such as not improving the loss function beyond a lower limit, is reached.

Algorithm 7 Bayesian Filtering-based Deep Reinforcement Learning

Input: Full Sensor history of N systems in the training set with age $T_j, \forall j \in \{1, 2, ..., N\}$.
for $j = 1$ to N **do**
Apply hybrid PF (Algorithm 4) to obtain particle values for the latent variables. Output $x_{1:T_j}^{1:N_s}, c_{1:T_j}^{1:N_s}, \lambda_{1:T_j}^{1:N_s}$.
end for

- Create the training dataset for DRL, which includes $\boldsymbol{T} \times B$, where $\boldsymbol{T} = \sum_{j=1}^{N} T_j, j \in \{1, 2, ..., N\}$.
- Introduce set of control actions $\mathcal{A} = \{\alpha^1, ..., \alpha^Z\}$.
- Create belief states $\boldsymbol{\zeta}_{1:\boldsymbol{T}}$.
- Initialize random target values $\mathcal{Q}_{1:\boldsymbol{T}}$.
- Set discount factor γ, initial neuron weights $\boldsymbol{\theta}$, the number of episodes E, and exploitation/exploration learning rate ϵ.
for $i = 1$ to E **do**
for $j = 1$ to N **do**
for $t' = 1$ to T_j **do**
- Collect belief state $\boldsymbol{\zeta}_{t'}$.
- With probability ϵ select a random action $\alpha_{t'} \in \mathcal{A}$ (**Exploitation**), otherwise select $\alpha_{t'} = \text{argmax}_{\alpha}\, \mathcal{Q}(\boldsymbol{\zeta}_{t'}, \alpha; \boldsymbol{\theta})$ (**Exploration**).
- Execute action $\alpha_{t'}$, and observe reward $r_{t'}$ and next belief state $\boldsymbol{\zeta}_{t'+1}$.
- Set $\mathcal{Y}_{t'} = \begin{cases} r_{t'}, \text{if } t' = T_j. \\ r_{t'} + \gamma \max_{\alpha'} \mathcal{Q}(\boldsymbol{\zeta}_{t'+1}, \alpha'; \boldsymbol{\theta}), \text{otherwise.} \end{cases}$
end for
end for
- Perform stochastic gradient descent step (Eq. (3.12)) for neural network training.
- New neuron weights $\boldsymbol{\theta}'$ are obtained. Set $\boldsymbol{\theta} = \boldsymbol{\theta}'$.
end for
Output: Trained the neural network and $\mathcal{Q}$-functions.

6.1.2 Training Deep Networks with Limited Data

The main obstacle of using deep neural networks is the large number of hyperparameters (i.e., neuron weights) that need to be tuned during network training. Even medium-sized deep neural networks may have thousands of hyperparameters, in which case a training dataset must be very large to account for the inherent diversity of the data. That makes deep neural networks sample-inefficient, leading to heavy overfitting during network training with small datasets and to very poor results in the testing data. Therefore, a logical assumption would be that deep neural networks are not good choices for action-value function approximator for the RUL estimation framework, since the training datasets have a moderate size ($\sim 10,000$ samples). However, in some special cases, small training datasets can provide good generalization and good results for the testing data, despite overfitting during network training. This can occur when the difference between training and testing distributions is small, which is known as small signal-to-noise ratio.

For the proposed frameworks, both training and testing data occur from the belief states $\boldsymbol{\zeta}_t$, which are essentially noise-free versions of the actual input data (sensor observations) since they were processed through Bayesian filtering and distribution discretization. As a result, the difference between training and testing data distribution is very small, leading to efficient deep neural network training even at the presence of limited data.

6.1.3 Real-time Control and Decision Making

Given a series of N life trajectories (i.e., failed systems) in the training set, where each trajectory $j \in N$ consists of a sequence of continuous observations $\boldsymbol{y}_1^i, ..., \boldsymbol{y}_T^i$,

discrete working status observations $o_1^i, ..., o_T^i$, and lifetime T_j, the objective is to develop a policy for real-time system control by training a DRL agent to minimize the average maintenance cost. One of the most critical parts of the DRL algorithm is the shape of the reward function that provides instant rewards based on the current state of the system and the action taken. For the real-time control framework, the reward function returns higher instant rewards when maintenance control action occurs before system failure. Furthermore, lower rewards are considered when a maintenance control action is chosen while the system is at its relatively early operating stages and when a total failure is unlikely. Similarly to the dynamic cost policy presented in Chapter 4, it is assumed that the cost of replacing a failing system is c_r regardless of its status at the time of replacement, while the cost of failure is c_f. For a system j with lifetime T_j, the instant reward at time $t \leq T_j$ is defined through negative penalty terms depending on the action chosen as follows:

$$r_t = \begin{cases} 0, & \alpha_t = \text{"Do Nothing" \& t} < T_j, \\ -\frac{c_r}{t}, & \alpha_t = \text{"Replace" \& t} < T_j, \\ -\frac{c_r + c_f}{T_j}, & \alpha_t = \text{"Do Nothing" \& t} = T_j, \\ -\frac{c_r + c_f}{T_j}, & \alpha_t = \text{"Replace" \& t} = T_j. \end{cases} \tag{6.1}$$

Two different actions, namely "Do Nothing", in which system operations continue, and "Replacement", in which system's operations are interrupted to perform the replacement, are considered. Both actions can take place either during system operations or after a system failure, in which case they both return a very low reward (or, equivalently, very high penalty). If action "Do Nothing" is chosen by the agent, and the system is still operational ($t < T_j$), then no reward is returned. In this case, the agent is encouraged to let the system continue its operation, as desired. When

action "Replace" is chosen by the agent while $t < T_j$, the reward depends on t. In this situation, the highest reward is given when $t = T_j - 1$. After the system fails, regardless of the chosen action, the system is subject to a negative reward $c_r + c_f$ divided by the age T_j of the system. Algorithm 8 summarizes the framework structure that shows how a trained DRL agent can transform a set of sensor data into an optimal policy, whether to terminate the system operation and perform replacement. Once the $\mathcal{Q}$-function is fully trained at any time point t, the following decision rule can be used at time point t:

$$\alpha_t^* = \underset{\alpha \in \{\text{Do Nothing, Replace}\}}{\operatorname{argmax}} \mathcal{Q}^{\pi^*}(\boldsymbol{\zeta}_t, \alpha). \tag{6.2}$$

Based on this decision rule, the system's operation is terminated when the optimal $\mathcal{Q}$-function suggests replacement or the system fails, whichever occurs first.

Algorithm 8 Real-time Control Framework for the jth System

Input: Trained agent with optimal $\mathcal{Q}$-function $\mathcal{Q}^{\pi^*}(\boldsymbol{\zeta}, \alpha)$.
Input: Initialize replacement cost $C = 0$ and $t = 0$.
Start: System Monitoring
- Set $t = t + 1$.
- Calculate the belief state $\boldsymbol{\zeta}_t$ from available sensor data
- Calculate two possible action-value function values and find the optimal action based on Eq. (6.2).
if $\alpha_t =$ "Replace" **then**
$C = \frac{c_r}{t}$,
if system is failed already **then**
$C = \frac{c_r + c_f}{T_j}$
end if
end if
Termination: If $C > 0$, then terminate the algorithm and output t and C, otherwise go back to the **Start**.
Output: Total average replacement cost C and the effective replacement time $T^* = t$.

6.1.4 RL for Warning Generation and RUL Estimation

The second Bayesian filtering-based DRL framework introduces an original approach to generate warnings and predict RULs for degrading systems. The objective is to train a DRL agent that, given a time threshold d defined by the user, generates an alarm for a working system when the age of the system is as close as possible to d units before failure. The threshold d can take any values from less than D, where D is a number that approximately represents the largest possible system's lifetime ($D \geq T_j$). The DRL agent should be trained to estimate a time window R_d that closely approximates the difference between system failure time T_j and time threshold d at which the alarm should be generated. Since there are training data with known lifetimes $T_{1:N}$, an agent can be trained to take care of such a task. During the training phase, it is possible for the agent to generate early warning or late warning alarms. The algorithm's objective, however, is to train the agent to generate alarms as close as possible to d units before system lifetime T_j. Hence, the reward function is designed to consider instant rewards for the agent depending on the time an alarm (if any) is raised. The action set $\mathcal{A}$ contains two different actions, namely "Continue" and "Warning". The instant reward considers an early warning cost c_e, and a late warning cost c_l, where $c_e < c_l$. The ratio $\frac{c_e}{c_l}$ shows how important early warnings are with respect to late warnings. For critical systems with large failure/downtime costs, c_l should be much larger than c_e. To obtain a policy that minimizes the total warning cost, given an ideal threshold d and denoting the action α_t at time, the reward

function can be defined as follows:

$$r_t = \begin{cases} 0, & \alpha_t^j(d) = \text{"Continue" \& t} \leq T_j - d, \\ -c_e(T_j - d - t), & \alpha_t^j(d) = \text{"Warning" \& t} < T_j - d, \\ 0, & \alpha_t^j(d) = \text{"Warning" \& } t = T_j - d, \\ 0, & \alpha_t^j(d) = \text{"Continue" \& } t > T_j - d \text{ \& } t < T_j, \\ -c_l(d - T_j + t), & \alpha_t^j(d) = \text{"Warning" \& } t > T_j - d \text{ \& } t < T_j, \\ -c_l d, & \text{t} = T_j \text{ (regardless of action } \alpha_t^j(d)). \end{cases} \tag{6.3}$$

The reward function focuses on three scenarios that the agent can be in during training. The first scenario is at $t \leq T_j - d$, where the agent either receives no instant penalty (negative reward) if it chooses action "Continue", or it receives a penalty for early warning if it chooses to generate such a warning. Similarly for $T_j - d < t < T_j$, if action "Continue" is chosen, the agent receives no reward, otherwise a late warning penalty is considered. Both of these occasions incite the agent to learn a policy of generating alarms as close as possible to $T_j - d$. The last scenario occurs at system failure when, regardless of the action taken, the agent receives the highest instant penalty for late warning. Here, more penalty terms can be added for very early/late warning or warning at failure. Algorithm 7 can now be applied for every value of d so that the DRL agent is trained at the end of the training phase. The trained agent returns $R^j(d)$ for system j, where $R^j(d)$ tends to be as close as possible to $|T_j - d|$ depending on the trade-off between early and late warning costs. The warning time $R^j(d)$ can be found as

$$R^j(d) = \inf\{t : \alpha_t^j(d) = \text{Warning}\}. \tag{6.4}$$

The above framework can estimate the remaining useful life only if the process defined

above needs to be repeated for all $d \in \{1, ..., D\}$. At any time point t, the output from each trained agent determines whether to continue (do nothing) or generate a warning. If there is at least one agent that suggests the action of a warning, then its threshold can determine the remaining life of the system at time t. For example, at time t, if the agent with $d = 10$ suggests to generate a warning, then it implies that the system is likely to fail in 10 cycles; that is, the remaining life is estimated to be 10 cycles. If multiple agents suggest the action of a warning, the one with the lowest d determines the estimated remaining useful life. If no agent suggests a warning, then the remaining life at that point cannot be determined. To avoid such a scenario, D is set as a large number so that at least one agent always suggests warning at any time point for a potentially large d. Algorithm 9-a provides an overview of the proposed procedure to train D for warning generation and then Algorithm 9-b presents how the results from Algorithm 9-a can be used to estimate the remaining life at time t for the jth system.

6.1.4.1 An Example of the Application of the Proposed Frameworks

Fig. 31 summarizes the potential outcome of the proposed frameworks by presenting an example on the determination of optimal replacement time, the warning generation process, and remaining useful life estimation. The age of the selected system is $T = 54$ while the effective age (replacement time T^*) determined by the framework is 52. That is, the system's operation is suggested to terminate 2 cycles before the true failure point. The proposed framework for warning generation was repeated for $d \in \{45, 12, 7\}$. It can be observed that for $d = 45$, the warning was issued at time $t = 10$, which is exactly 44 cycles before the failure point. For $d = 12$

Algorithm 9 Steps to Estimate the Remaining Useful Life

Input: Training set, which includes N degrading systems with lifetimes $T_j, j \in \{1, 2, ..., N\}$.
Input: Set the number of threshold values D.
Input: Belief states $\boldsymbol{\zeta}_{1:t}^{j}, j \in \{1, ..., N\}$.
Input: Set costs c_e and c_l.
Input: Initialize warning cost C.

(a) DRL agents training.

for $d = 1$ to D **do**
- Run Algorithm 7 based on the reward function in Eq. 6.3.
- Get the trained agent and its associate optimal $\mathcal{Q}$-function.

end for
(b): Remaining Useful Estimation for System j at time t (l_t^j).

Input: The history of sensor data for system j up to time t
- Calculate $\alpha_t^j(d)$ for all $d \in \{1, ..., D\}$ using the D trained agent from part (a).

Output: Calculate the remaining life l_t^j as

$$l_t^j = \min(d^*), \text{where } d^* = \{d; \alpha_t^j(d) = \text{Warning}\}.$$

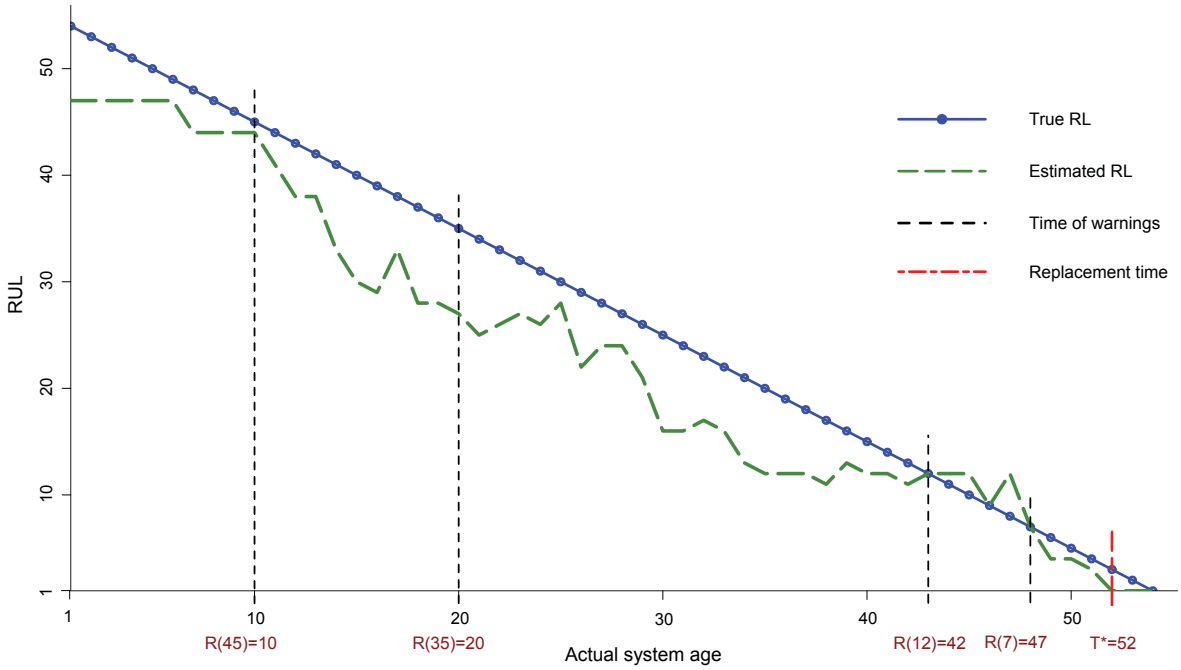


Figure 31: Overview of the proposed frameworks with 3 potential outcomes: warning times (estimated for 3 values for d), replacement time, and remaining life estimates (throughout the lifecycle).

and $d = 7$, the warning was generated at times $t = 42$ and $t = 47$, respectively, which gives $R(12) = 42$ and $R(7) = 47$. It can also be seen that the remaining useful life estimate (dashed line) is relatively close to the actual remaining life particularly near the end of the lifecycle.

6.2 Numerical Experiments

Both frameworks were tested using the same data as in Chapter 5 (simulated and CMAPSS). The process demonstrated in Section 6.1.1 was employed for constructing the final set of belief states that were later used as inputs for the RL frameworks.

6.2.1 Real-Time Decision Making With Reinforcement Learning

The proposed real-time framework for maintenance decision making was compared with a set of benchmark policies in order to establish its effectiveness. These policies are a) an *ideal maintenance policy*, b) a *corrective maintenance policy*, and c) a *time-based maintenance policy* or a *preventive maintenance policy*. The optimal cost associated with each of these benchmark policies can be computed as follows:

a) *Ideal maintenance cost (IMC)*: Based on this hypothetical policy, the system's operation is terminated exactly one cycle before failure so that the system useful life is maximized while the failure is prevented. The cost associated with this policy can be computed as

$$\phi_{IMC} = \frac{N \cdot c_r}{N \cdot (\mathbb{E}(T) - 1)} \approx \frac{N \cdot c_r}{\sum_{j=1}^{N} (T_j - 1)}, \tag{6.5}$$

where $T_j, j \in \{1, 2, ..., N\}$ denotes the lifetimes of the jth system. This cost represents the optimal return value of the reward function, that is replacements always occur at $t = T_j - 1$ for all $j \in \{1, ..., N\}$. In theory, no other policy can provide a better cost than Eq. 6.5, which is why it can be an adequate reference for the comparison. In reality, no policy actually exists that can provide such a low cost. It is only used to evaluate how close the obtained cost from our policy is to this low limit.

b) *Corrective maintenance cost (CMC)*: Based on this policy, the maintenance starts right after the system fails. Thus, each system always operates up to its highest useful life but it is also always subject to the failure cost. The cost of this policy, which can be considered as the upper bound for any policy, can be computed as follows:

$$\phi_{CMC} = \frac{(c_r + c_f)}{\mathbb{E}(T)} \approx \frac{N \cdot (c_r + c_f)}{\sum_{j=1}^{N} T_j}. \tag{6.6}$$

c) *Time-based maintenance cost (TBMC)*: The objective of this policy is to find the time threshold T^*, when the system needs to be replaced regardless of its status. In other words, the system is replaced at time T^* or at failure, whichever occurs first. To numerically find T^*, the empirical average maintenance cost is calculated for a time horizon $t = \{1, 2, ..., T_u\}$, which $t = 1$ denotes the time each system starts functioning and T_u is the maximum possible lifetime in the historical data. Using exhaustive search, T^* is obtained where the average unit cost is minimized. Thus

$$T^* = \underset{t \in \{1,2,...,T_u\}}{\operatorname{argmin}} \frac{1}{N} \sum_{j} \left[\mathbf{1}_{\{t<T_j\}} \frac{c_r}{t} + \mathbf{1}_{\{t \geq T_j\}} \frac{c_r + c_f}{T_j} \right], \tag{6.7}$$

$$\phi_{TBMC} = \sum_{j=1}^{N} \left[\mathbf{1}_{\{T^*<T_j\}} \frac{c_r}{T^*} + \mathbf{1}_{\{T^* \geq T_j\}} \frac{c_r + c_f}{T_j} \right], \tag{6.8}$$

where the first part in the summation at Eq. 6.7 is for the cases where the system's age is larger than t, and the second part is for the systems that ages less than t. The cost of replacement is defined as $c_r = 100$ and, in order to test the agent's sensitivity for timely choosing maintenance actions (or not performing maintenance at all), five different values for the failure cost c_f were chosen, $c_f = (25, 50, 100, 500, 1000)$. Combinations where $c_r \geq c_f$ do not represent real-world applications, but they are considered to test the agent's ability to be trained properly. The proposed framework was tested for the three neural network architectures mentioned in Sections 3.8.1-3.8.2-3.8.3. The number of hidden neurons and hidden layers for the DNN architecture was selected with extensive cross-validation to better represent the interactions between the input data and their target values, given the limitations of each architecture. The final structure of the neural networks used in our numerical experiments is as follows: a) *ANN: 50 neurons*, b) *DNN: 2 hidden layers (1^{st} layer:*

32 *neurons,* 2^{nd} *layer:* 16 *neurons)*, c) *RNN: 2 hidden stacked layers (*1^{st} *layer:* 64 *states,* 2^{nd} *layer:* 32 *states).*

6.2.1.1 Results for the Simulated Dataset

Using the Eqs. 5.22-5.26, 200 systems were simulated, where $N = 150$ were used for training and $\hat{N} = 50$ for testing. At first, the convergence of the DRL framework training after 200 episodes was evaluated. Outcomes from $\mathcal{Q}$-functions using different values of c_f are shown in Fig. 32. It can be seen from this figure that the agent reaches an optimal state after some iterations for all three networks and almost all cost combinations. To better observe the approximation error and convergence, Fig. 33 shows the root mean squared error (RMSE) of the neural network loss over different episodes. This figure shows that the neural networks used in the proposed DRL policy perform reasonably well in terms of estimating Q, and the RMSE values converge to a small value. Among all network structures, the deep neural network performs the best in terms of convergence.

After training each agent, Algorithm 8 was implemented for estimating the replacement time of each system and calculating the average replacement cost of the systems within the test set $\hat{N}$. Tables 7 presents the results compared with the benchmark models discussed in Section 6.2.1.1. The first result is for the average replacement cost for the five aforementioned combinations of (c_r, c_f) and all three neural network architectures (ANN, DNN, RNN). Furthermore, the average cost for the three benchmark policies is reported (IMC, TBMC, CMC). As expected, the ideal replacement policy IMC provides the lowest cost. The proposed model provides the second-best results and performs better than the time-based and corrective policies.

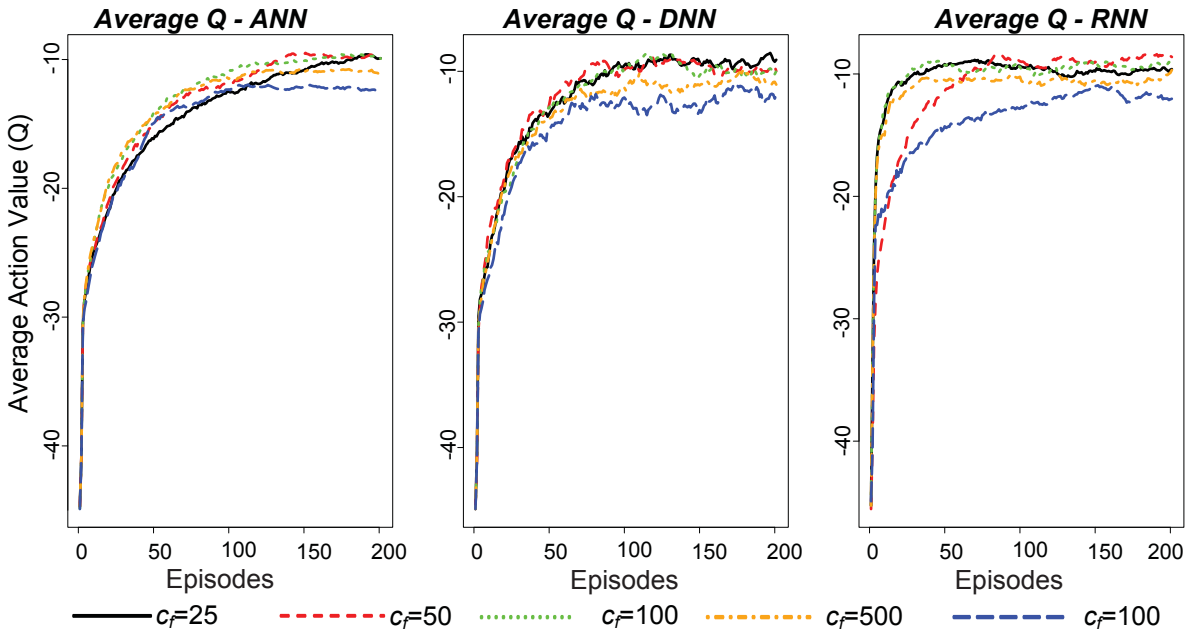


Figure 32: Evolution of the $\mathcal{Q}$-function during the training phase.

It also generally outperforms the time-based policy as the cost of failure increases. Maximum system utilization occurs when the corrective policy CMC is considered, which is expected because all systems are allowed to fail. Additionally, it can be seen that as the cost of failure increases, the agents tend to terminate the operation earlier and prevent more failure replacement. Finally, when the optimal policy IMC is considered, then by default all systems are replaced before failure (0% failure rate). The model yields less failure replacements and reaches 0% failure rate when the cost of failure is too large. The results are very close the IMC, which theoretically gives the lowest possible cost for the systems.

6.2.1.2 CMAPSS Dataset

The numerical experiments presented in Section 6.2.1.1 were also applied on the CMAPPS dataset in order to observe how well the replacement points can be obtained. The structures of the neural networks are the same as the simulation dataset.

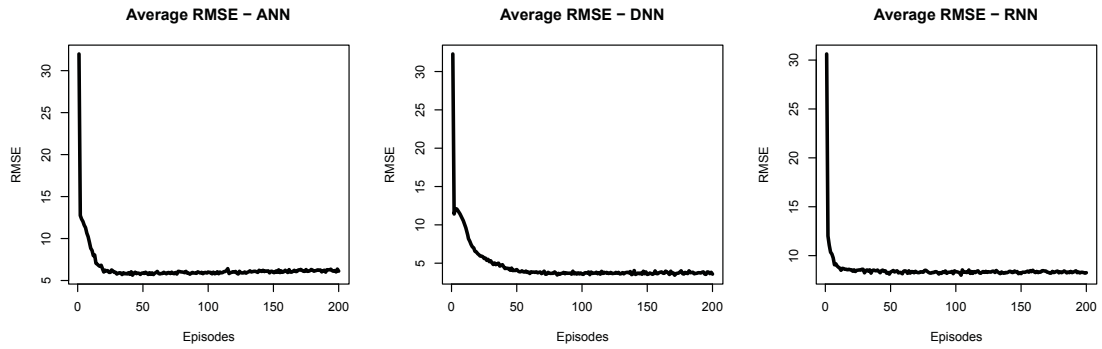


Figure 33: Average neural network loss for three network structures ANN, DNN, RNN during the training phase.

The dataset consists of 100 degradation time-series, out of which $N = 80$ were randomly chosen for training and $\hat{N} = 20$ were used for testing. The trained agent was then evaluated on the testing set $\hat{N}$, and the results obtained are presented in Table 8. As seen in this table, the ideal maintenance cost policy IMC provides the best results. However, the proposed model performs reasonable results in terms of minimizing cost, preventing failure replacement, and maximizing system operation. Additionally, potentially due to their structures that are able to handle more complexity, DNN and RNN perform better than the regular ANN.

Table 9 shows the CPU time for model training for each of the neural network architectures and for both simulated and CMAPSS data. Agent training was conducted for 200 episodes. ANN and DNN require significantly less time for agent training than the RNN, since RNNs have to analyze the input data sequentially. Hence, a trade-off between result accuracy in the testing set and training time exists, and the user should take that fact into consideration before applying the framework. As a proof to this, the scalability of the proposed algorithm is demonstrated in Figure 34, where the same experiments were conducted for different simulated training set sizes, namely

Table 7: Average replacement costs, time, and failure rate for $\hat{N}$ (simulated data)

	c_f	c_r	**ANN**	**DNN**	**RNN**	**IMC**	**TBMC**	**CMC**
Cost	25	100	5.34	5.78	5.29	4.54	5.41	5.43
	50	100	5.48	6.13	6.07	4.54	6.48	6.52
	100	100	5.62	6.38	5.35	4.54	8.09	8.69
	500	100	6.49	6.17	5.81	4.54	10.02	26.06
	1000	100	5.59	6	5.34	4.54	10.93	47.79
Time	25	100	19.68	18.08	19.84	22.02	22.58	23.02
	50	100	19.52	18.58	20.26	22.02	19.28	23.02
	100	100	19.58	16.3	19.68	22.02	13.76	23.02
	500	100	18.5	17.82	15.92	22.02	10.98	23.02
	1000	100	17.9	16.66	17.9	22.02	10.98	23.02
Failure	25	100	20%	28%	46%	0%	94%	100%
	50	100	14%	19%	46%	0%	54%	100%
	100	100	10%	4%	20%	0%	12%	100%
	500	100	4%	2%	2%	0%	2%	100%
	1000	100	0%	0%	0%	0%	2%	100%

for 800, 3,000, and 8,000 samples, and for 200 episodes. As expected, ANN and DNN generally perform well in terms of CPU time, even when the training set becomes very large, whereas the CPU time for RNN training increases significantly. This outcome proves again that, while RNNs behave well as $\mathcal{Q}$-function approximators, the amount of time needed to train them can be a prohibitive factor.

6.2.2 Warning Generation and RUL Estimation

The framework in Section 6.1.4 was validated using the same training and testing datasets in Sections 6.2.1.1-6.2.1.2. The maximum number of threshold values was set to $D = 50$; therefore, 50 RL agents were trained (i.e., $d \in \{1, ..., 50\}$) for the same neural network architectures as in the previous Section using Algorithm 9-a. The first outcome of the dth agent is to monitor the most updated set of sensor data over time to generate warning d units before the actual failure points. After completion of the training process (see Algorithm 9-a), the agents estimated the RULs of the

Table 8: Average replacement costs, time, and failure rate for $\hat{N}$ (CMAPSS data).

	c_f	c_r	**ANN**	**DNN**	**RNN**	**IMC**	**TBMC**	**CMC**
Cost	25	100	2.48	2.13	2.15	1.93	2.23	2.37
	50	100	2.27	2.1	2.1	1.93	2.31	2.84
	100	100	2.32	2.09	2.02	1.93	2.38	3.78
	500	100	2.27	2.21	2.05	1.93	2.38	11.34
	1000	100	2.16	2.6	2.04	1.93	2.38	20.8
Time	25	100	40.4	47	49.9	51.9	46.6	52.9
	50	100	45.25	47.25	47.3	51.9	46.6	52.9
	100	100	43.05	47.8	48.45	51.9	42	52.9
	500	100	44.05	45.35	47.2	51.9	42	52.9
	1000	100	46.35	38.6	48.75	51.9	42	52.9
Failure	25	100	0%	0%	35%	0%	15%	100%
	50	100	5%	0%	5%	0%	15%	100%
	100	100	0%	0%	0%	0%	0%	100%
	500	100	0%	0%	0%	0%	0%	100%
	1000	100	0%	0%	0%	0%	0%	100%

Table 9: CPU time for agent training (minutes).

	Simulated data	**CMAPSS data**
ANN	5	10
DNN	6.67	13.4
RNN	100	233.3

testing sets. Nine cost combinations for early warning (c_e) and late warning (c_l) were considered in the cost function presented in Eq. 6.3, given below:

Cost combinations:

- **C1:**$\{c_e = 100, c_l = 500\}$
- **C2:**$\{c_e = 100, c_l = 100\}$
- **C3:**$\{c_e = 500, c_l = 100\}$
- **C4:**$\{c_e = 100, c_l = 200\}$
- **C5:**$\{c_e = 200, c_l = 100\}$
- **C6:**$\{c_e = 100, c_l = 300\}$

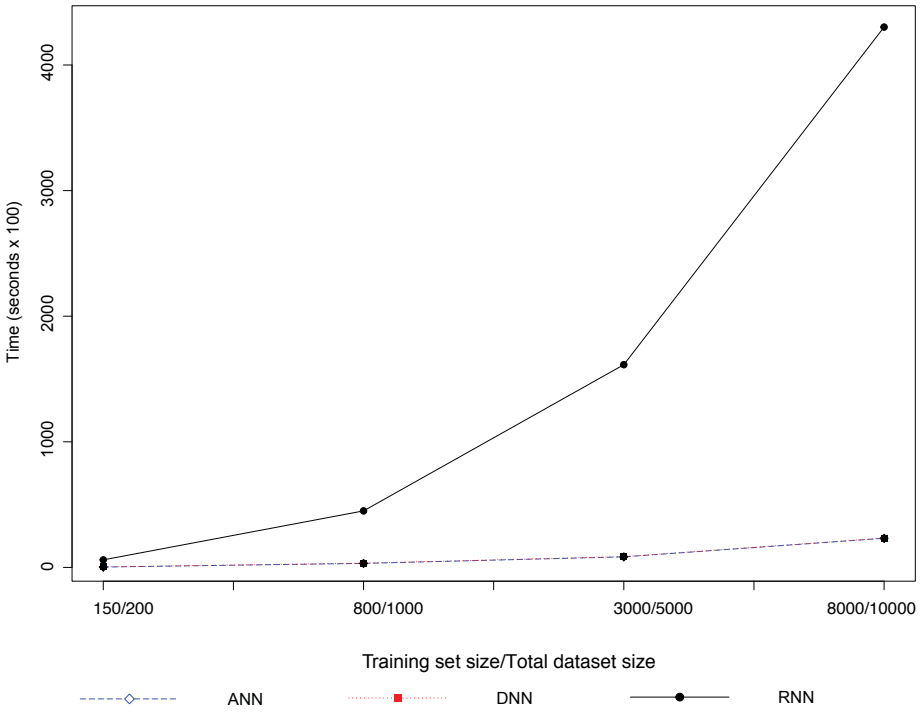


Figure 34: Scalability of the proposed framework.

- **C7:**$\{c_e = 300, c_l = 100\}$
- **C8:**$\{c_e = 100, c_l = 400\}$
- **C9:**$\{c_e = 400, c_l = 100\}$

As mentioned before, for critical systems it is prudent to set $c_e < c_l$. That way, the agent will become risk-averse and will always try to generate warnings before the actual threshold. While that may lead to shorter RUL estimates, in cases that include significant high costs, such as monetary costs and cost of life, it is always better to underestimate the system's RUL. On the other hand, for systems where the actual failure costs are not that significant, the agent can be allowed to be more greedy in its decisions, that is generating more late warnings. For the purpose of the experiments in this section, both cases are considered in order to observe the behavior of the agents during both training and testing phases.

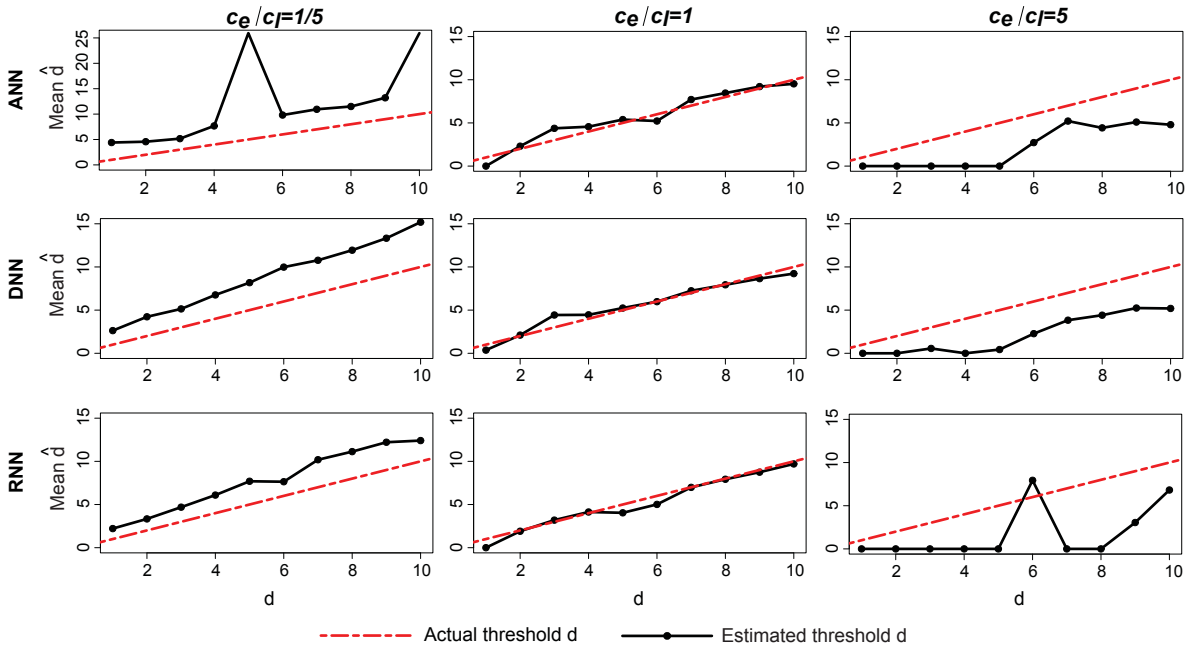


Figure 35: Comparison between the true ds and their estimates for all systems in the training set and for cost combinations **C1**, **C2**, **C3**.

6.2.2.1 Simulation Dataset

Figures 35, 36, and 37 compare true and estimated d values, averaged over all systems in the training set N for $d = \{1, 2, ..., 10\}$ and for all cost combinations. The agents tend to overestimate d, that is, they generate more early warnings, when the cost of early warning is lower than the cost of late warning ($c_e < c_l$). Conversely, the exact opposite occurs when the cost of early warning is significantly higher than its late warning counterpart ($c_e > c_l$). The best and most stable case occurs when both costs are equal (middle column on Figure 35).

To see how well the dth agent can predict system failure, Algorithm 9-b was applied on every system in the testing set $\hat{N}$ to see at what points the warnings are generated, and subsequently estimate their RULs. Figure 38 provides a sample of results obtained when RNN is used as $\mathcal{Q}$-function approximator, between true and estimated RULs in terms of % of a true lifetime, for all 50 systems in $\hat{N}$ along the

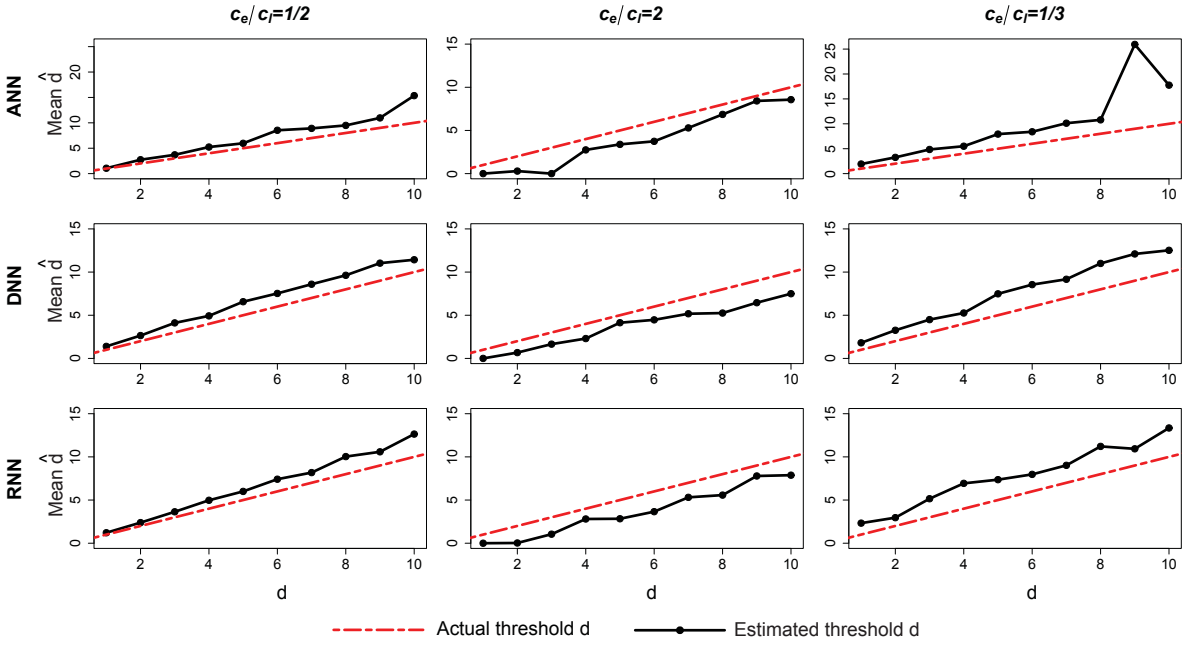


Figure 36: Comparison between the true ds and their estimates for all systems in the training set and for cost combinations **C4**, **C5**, **C6**.

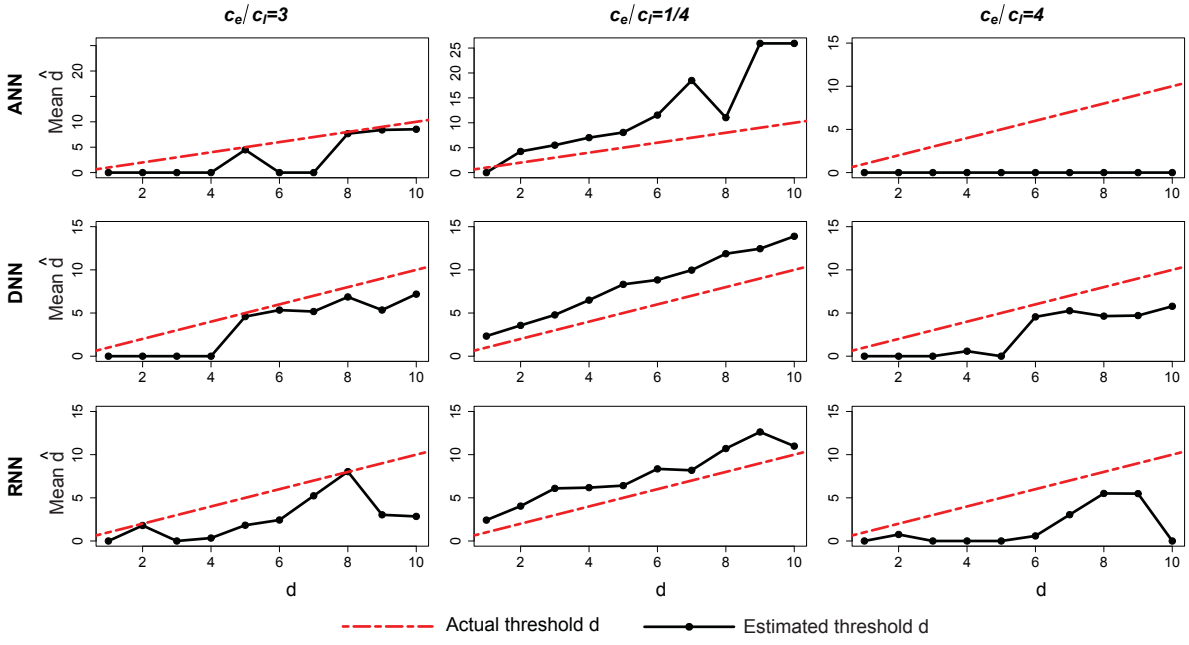


Figure 37: Comparison between the true ds and their estimates for all systems in the training set and for cost combinations **C7**, **C8**, **C9**.

average % values. Furthermore, the results presented are for cases where $c_e \leq c_l$. The framework gives stable results when the cost of late warning is at most equal to the one of early warning. Especially when $c_e < c_l$, the estimated RULs are almost equal to their true values, whereas they diverge when $c_e = c_l$. However, in all cases, the true and estimated RULs converge towards the last 10 cycles of system lifetime (see Tables 10-12).

Table 10: Convergence between true and estimated RUL ($c_e = c_l$).

True RUL	**Estimated RUL (median)**	**Estimated RUL (sd)**
10	15	6.97
9	14	5.48
8	13	4.73
7	9.5	4.78
6	8	5.23
5	7	4.24
4	6	3.66
3	4	3.06
2	3	1.33
1	1	0.99

Table 11: Convergence between true and estimated RUL ($c_e = 3c_l$).

True RUL	**Estimated RUL (median)**	**Estimated RUL (sd)**
10	11.5	7.74
9	10	4.29
8	8	4.57
7	8.5	4.28
6	6	3.54
5	5.5	4.13
4	5	3.31
3	3	2.24
2	2	1.25
1	2	1.1

Table 12: Convergence between true and estimated RUL ($c_e = 4c_l$).

True RUL	Estimated RUL (median)	Estimated RUL (sd)
10	10	3.7
9	10	3.74
8	8	3.24
7	7	3.02
6	6	2.84
5	4.5	2.99
4	3	2.37
3	3	1.68
2	1	0.92
1	1	1.02

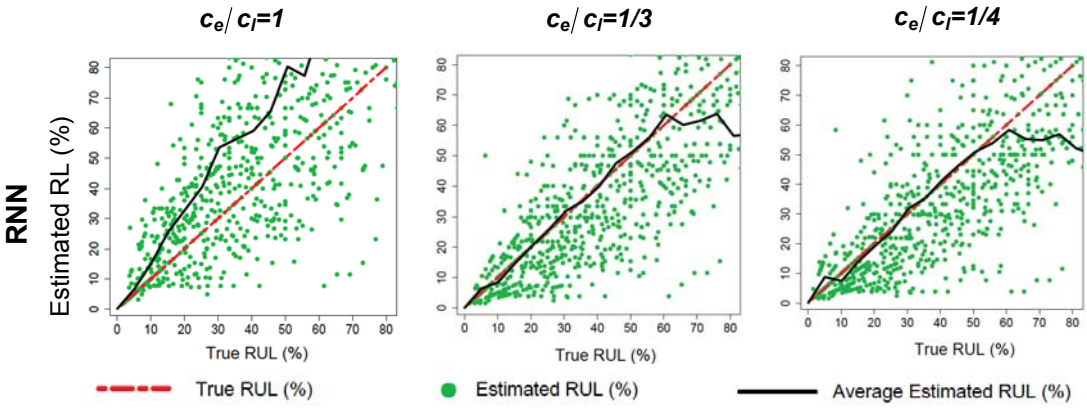


Figure 38: Sample of results for RUL estimation (simulated data).

6.2.2.2 CMAPSS Dataset

The framework was also tested on the CMAPPS dataset. Similar to Section 6.2.2.1, $D = 50$ agents were trained using Algorithm 9-a, and the comparison between true d and estimated $\hat{d}$ values, averaged over all systems on the training set N, are presented in Figures 39-41. Again, the estimated values $\hat{d}$ converge closer to the true values d for cost combination **C2** ($c_e = c_l$). Similar to the simulated data, Algorithm 9-b was applied on every system in the testing set $\hat{N}$ to observe the points at which warnings are generated. A sample of results from comparing the true and estimated

RULs in terms of percentage for all 20 systems in $\hat{N}$, and when RNN is used as $\mathcal{Q}$-function approximator, are shown in Figure 42. The results presented are for cases where $c_e \leq c_l$. Similar to the simulated data, the true and estimated RULs converge to the last 10 cycles of system lifetime (see Tables 13-15).

Table 13: Convergence between true and estimated RUL ($c_e = c_l$).

True RUL	**Estimated RUL (median)**	**Estimated RUL (sd)**
10	11.5	3.83
9	8	4.07
8	8	3.48
7	5	3.35
6	4	2.41
5	4.5	1.89
4	4.5	1.21
3	3	1.4
2	2	1.42
1	2	0.92

Table 14: Convergence between true and estimated RUL ($c_e = 2c_l$).

True RUL	**Estimated RUL (median)**	**Estimated RUL (sd)**
10	9	3.58
9	7	3.34
8	6	2.42
7	6	2.83
6	5	1.72
5	3.5	1.6
4	3	1.13
3	3	1.14
2	2	0.99
1	2	1.27

Table 16 shows the total CPU time for model training for each of the neural network architectures and for both simulated and CMAPSS data. The results are summarized over D. Agent training was conducted for 50 episodes. Similar to the maintenance cost minimization framework, ANN and DNN require significantly less

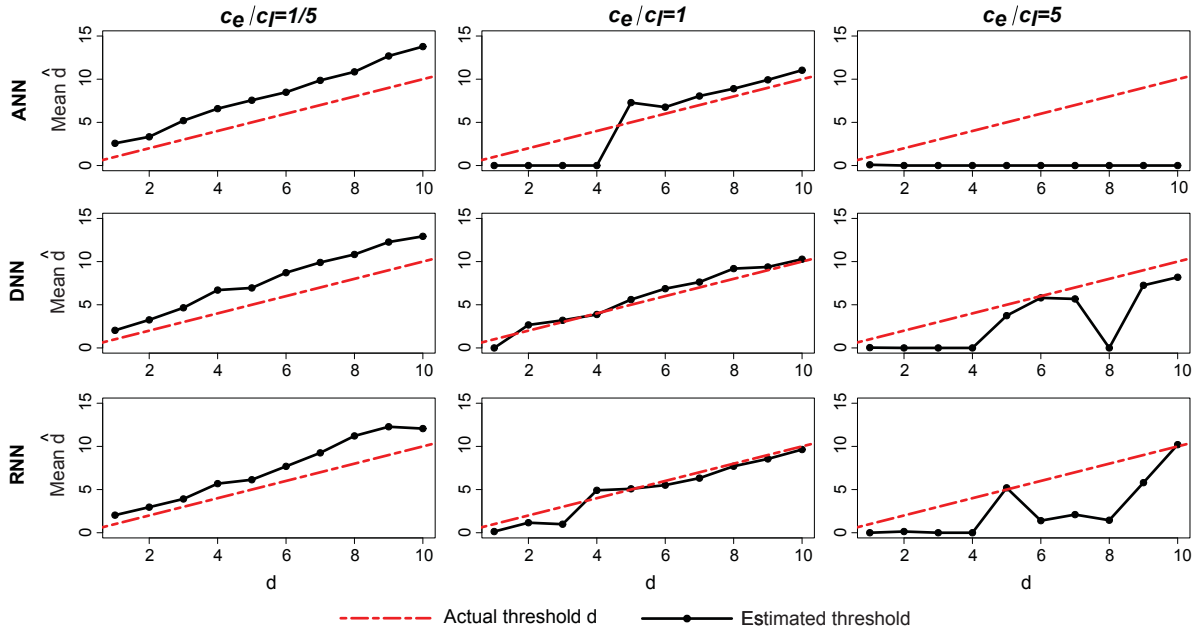


Figure 39: Comparison between true and average warning thresholds for all systems in the testing set $\hat{N}$ and for cost combinations **C1**, **C2**, **C3**.

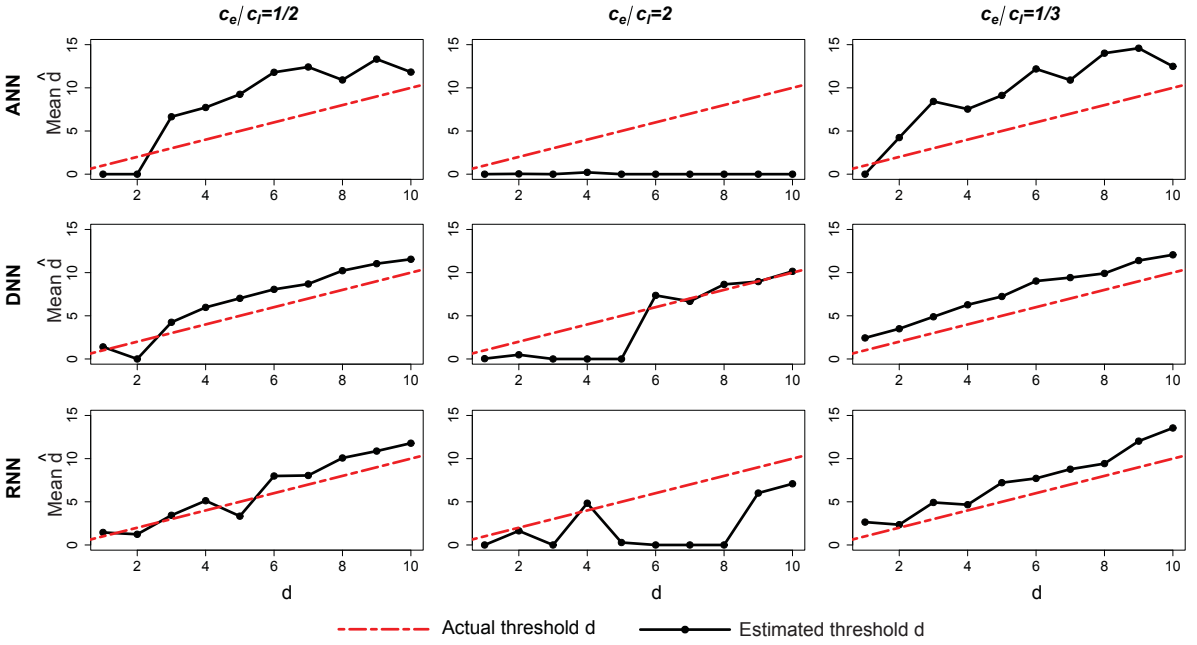


Figure 40: Comparison between true and average warning thresholds for all systems in the testing set $\hat{N}$ and for cost combinations **C4**, **C5**, **C6**.

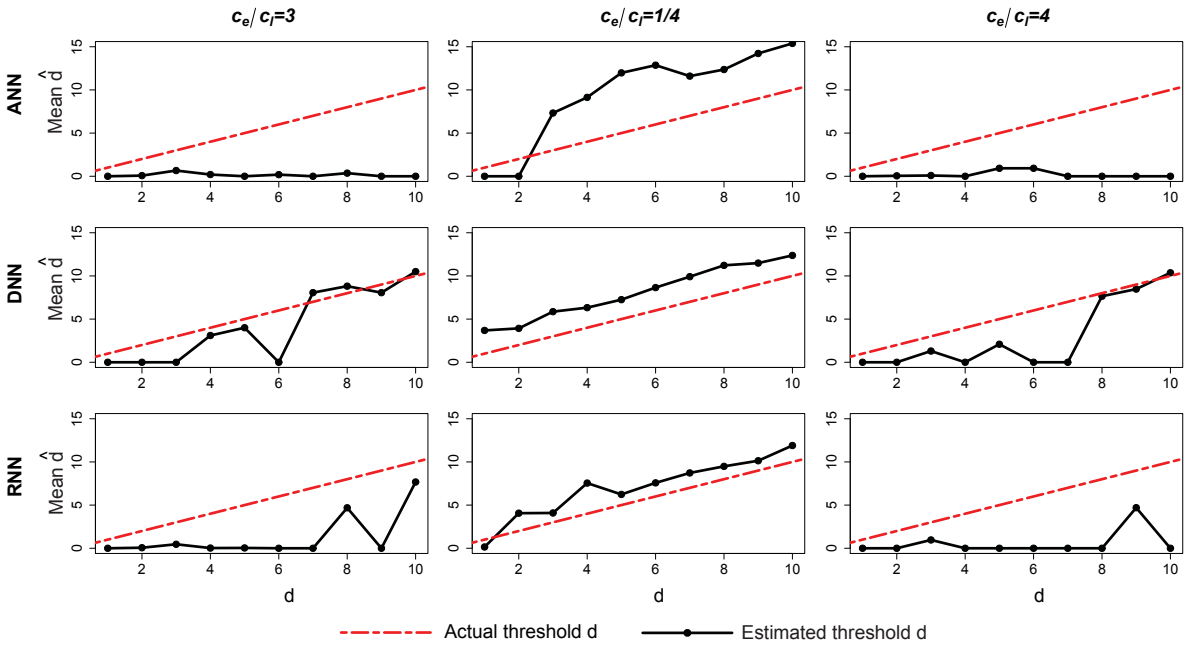


Figure 41: Comparison between true and average warning thresholds for all systems in the testing set $\hat{N}$ and for cost combinations **C7**, **C8**, **C9**.

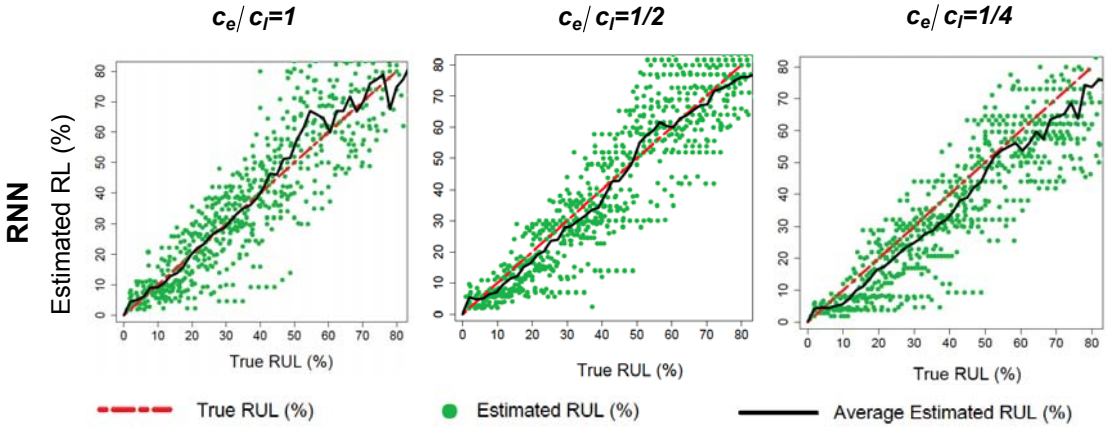


Figure 42: Sample of results for RUL estimation (CMAPSS data).

Table 15: Convergence between true and estimated RUL ($c_e = 4c_l$).

True RUL	Estimated RUL (median)	Estimated RUL (sd)
10	10	4.23
9	7	3.69
8	5	3.15
7	4	3.24
6	4	1.69
5	2	1.34
4	2	0.88
3	2	0.55
2	2	0.48
1	2	0.36

Table 16: Total CPU time for D agents training (hours).

	Simulated data	CMAPSS data
ANN	1.32	1.7
DNN	1.43	1.76
RNN	32.2	38.5

time for agent training than the RNN. This is more obvious in Figure 43, where the scalability of the RUL estimation framework is demonstrated, for the same training sets as in Figure 34. The values represent the total amount of training time for a single agent d and for 50 episodes. Similar to the maintenance cost framework, ANN and DNN require much less training time, regardless of the size of the training data. On the other hand, the CPU time for RNN increases significantly. Although RNN gives the most accurate RUL estimation results, the user has to carefully measure the trade-off between training time and model accuracy.

From summarizing over all the results obtained for both simulated and CMAPSS datasets, it is proved that the proposed RUL estimation framework provides a reliable way to generate warnings and estimate RUL at different time points while the system is operating. Comparing the results for all nine different cost combinations, it is

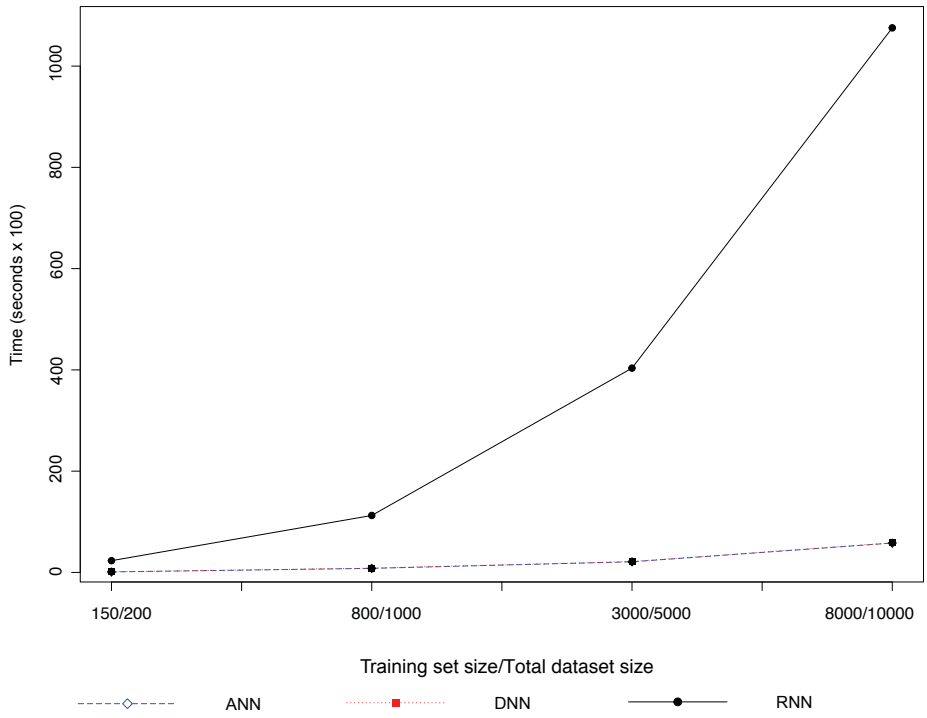


Figure 43: Scalability of the RUL estimation framework.

observed that the most objective RUL estimates are calculated when the costs of early and late warnings are equal. The users have the opportunity to change the cost of early and late warnings and to adjust the estimation results. However, they must be careful because the RUL estimates for unequal costs of early and late warnings diverge from the true RUL values. With regards to the neural network structures used as function approximators, the RNN performs better than ANN/DNN in most cases. The stronger performance of RNN can be explained from the fact that it can store the entire sequence of inputs, making it more adequate to accommodate complex environment dynamics. Nevertheless, due to the significant amount of time required for RNN training, the users must perform exhaustive cross validation regarding user-specific parameters, such as i) the number of hidden neurons and/or layers, ii) the number of training episodes, or iii) neural network regularization techniques (e.g., dropout). In conclusion, the RUL estimation framework provides the best results;

that is, DRL agents develop optimal policies for RUL estimation when both costs for early and late warnings are equal. The only drawback of this model is that it is computationally time-intensive, especially when RNNs are used as $\mathcal{Q}$-function approximator. Futher research is required to reduce the CPU time for model training.

6.3 Summary

This chapter describes novel dynamic decision frameworks using DRL that transform real-time sensor data collected from monitoring sensors into decision-making intelligence and actionable insights. It proposes two original frameworks for degrading systems using DRL with stochastic environments: i) a real-time control and decision-making framework for maintenance replacement, and ii) a RUL estimation framework that can also be used to generate warnings at any point while the system is operating. Bayesian filtering was utilized to infer system latent dynamics and transform sensor data to RL inputs.

CHAPTER 7

Conclusion, Limitations, and Future Work

The main objectove of this thesis was to propose a fully automated, real-time control and decision-making structure that can manage to combine Bayesian filtering, machine learning, and deep reinforcement learning methods into one comprehensive monitoring, control, and decision-making structure. This structure requires only the multidimensional sensor data and the systems working condition as inputs at any given time. A multitude of outputs can be extracted at every step, including hidden state and RUL estimates and optimal maintenance costs, depending on the users needs. Three monitoring and control frameworks were presented in this thesis. The first framework, described in Chapter 4, was based on a combination of Kalman filtering and machine learning for state-parameter inference, aiming to analyze systems with linear and Gaussian dynamics. While not part of the proposed structure, it introduced closed-form solution methods for an array of important diagnostics and prognostics measures that significantly improved accuracy and computational times compared to other numerical methods, such as Monte-Carlo. Despite its potential, the first framework can only work when system dynamics are linear. However, the majority of complex systems rarely follow a linear dynamics pattern. For this reason, the first part of the proposed structure was based on the particle filtering-based frame-

work described in Chapter 5. This framework was designed to accommodate systems with both linear and non-linear dynamics, as well as multiple levels of hidden and observable processes. Furthermore, this framework introduced a number of innovations that manage to address certain limitations, such as predefined failure thresholds and unrealistic parametric processes, compared with similar works in the current literature. For the former limitation, the framework used a similar approach as the first proposed structure, where the logistic regression was utilized as a stochastic process directly dependent on the latent system degradation. In addition to this, the Extreme Learning Machine neural network was used to approximate the observation process in a non-parametric manner, thus avoiding the utilization of any parametric formula. The results from both diagnostic and prognostic measures proved the frameworks potential for system monitoring and control. Finally, the second part of the proposed structure introduced a decision-making mechanism that was based on the most recent advancements in deep reinforcement learning. In this framework, software agents were trained to determine policies for maintenance cost minimization and RUL estimation, using the concepts of deep reinforcement learning. The novelty of this model is the purely stochastic environment with which the agents interacted in order to update its policy. The environment consisted of the latent system dynamics that were estimated from particle filtering. The results proved that deep reinforcement learning can benefit sensor-driven control and decision-making approaches where minimal human intervention is desirable.

There are still a number of assumptions and limitations that were considered during the development of the proposed frameworks. One of them concerns the purely linear degradation process of the first framework, which does not consider any sudden

changes. A more realistic approach that will be able to capture these changes, such as one using ensemble Kalman filters, could be investigated in the future. The most important limitation is the CPU time during model training. The proposed models are robust, but, because of their complexity, they require a significant amount of time for model training. This can be prohibitive in some real-world applications. The RNN training time is a good example of this limitation. Training RNNs is the biggest obstacle for these deep neural networks, due to their design in analyzing data sequentially and their demand for hardware resources (not parallelizable). Another limitation is described in Chapter 5, where the proposed HSSM utilizes an exponential process to describe the latent degradation. Exponential formulas are popular choices for describing the evolution of the latent degradation process in data-driven approaches. However, as was proved through the sensor observation process in this model, a parametric approach cannot always capture the true complexity of the process evolution. Yet another limitation is that all frameworks presented in this thesis assume failures of single-unit systems (1-D latent degradation). This approach is useful for simplifying the structure of the proposed mathematical models, but it cannot fully represent modern systems that are composed of multiple subsystems. Finally, although it is a contribution of this thesis, it will not always be easy to obtain such noiseless environments for agent training as described in Chapter 6. Therefore, different approaches and techniques in model design and neural network architecture should be considered. Furthermore, model designs that involve processes to run using GPUs (e.g. CUDA) for faster analysis are also necessary and will be considered for future work.

In future work, the model training approach for the framework described in Chapter 5 would need to be reconsidered in order to reduce the CPU training time into more acceptable levels for real-time applications. More stable training options in the Bayesian domain (e.g. particle marginal Metropolis algorithm) will also be investigated. Regarding Phase I of the proposed structure, fully non-parametric processes for both latent and observable levels need to be considered. Such models will not require any parametric or distributional assumptions, and they will be able to fully describe the system dynamics. Recent advancements in deep learning, such as deep belief networks and temporal convolutional networks, are potential model candidates. However, this procedure must be carefully conducted, as these methods usually require long training times and very large training datasets. Therefore, specialized data preprocessing steps, similar to the ones described in Chapter 6, need to be considered. Finally, all frameworks will need to be generalized in order to be used for multi-unit systems with competing and non-competing failure modes.

APPENDIX

Cardinality of RUL Results from Framework 3

This appendix provides the results from RUL estimation for all early/late warning cost combinations in Section 6.2.2. The images show the results for simulated and CMAPSS data, and for every neural network architecture.

Simulated Data

Figures 44-46 provide scatterplots comparing true and estimated RULs in terms of % of a true lifetime for all 50 systems in $\hat{N}$ along the average % values. In most cases the agents slightly underestimate RUL when $c_e < c_l$, however there are cases where the average values are close to the true RUL %. On the other hand, when $c_e > c_l$, the RUL estimates for all different $\mathcal{Q}$-function approximators are overestimated the closer the estimates are to the actual system end-of-life, which is reasonable since the agents here are more likely to generate late warnings.

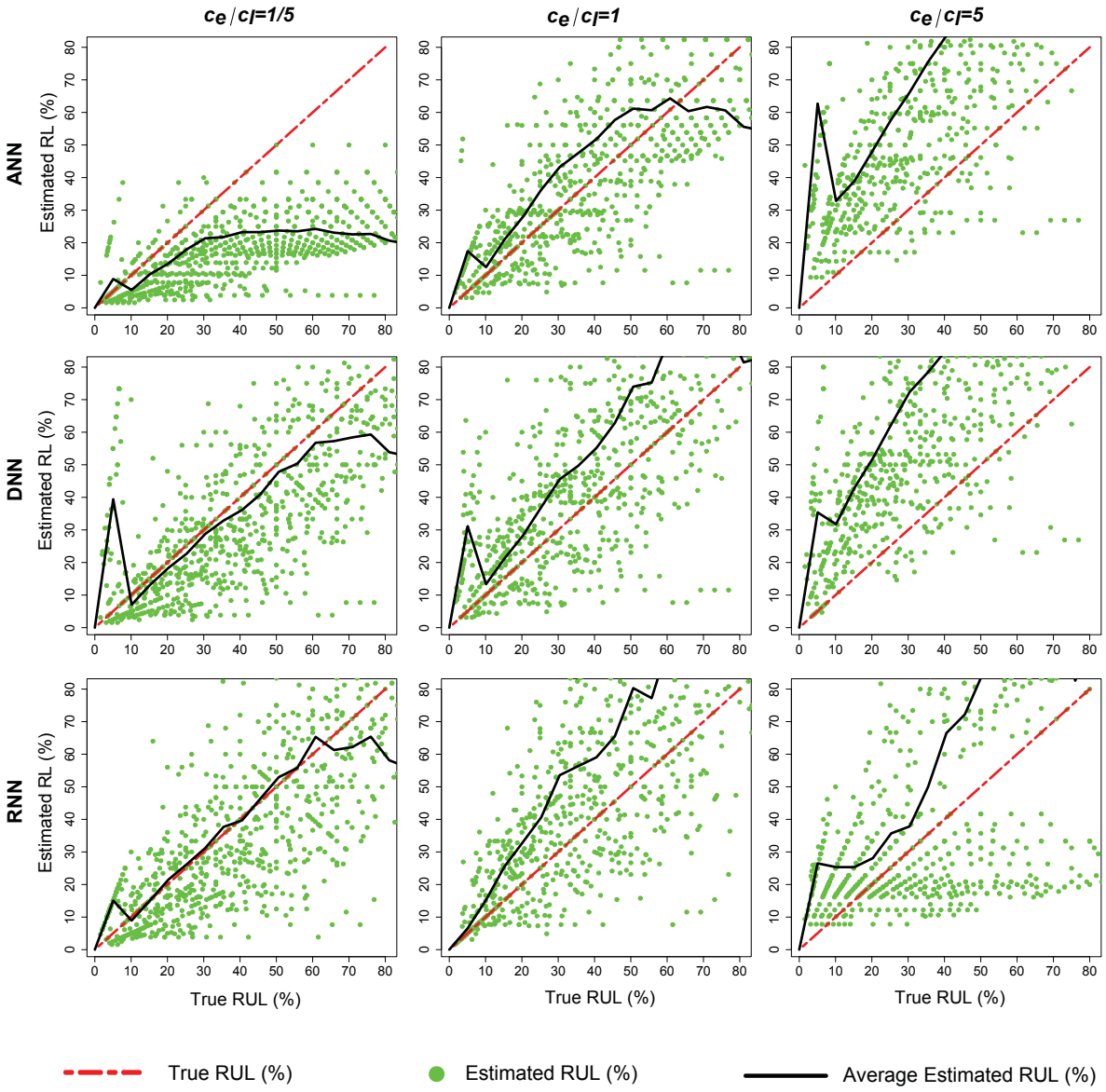


Figure 44: Distribution of RUL estimates as percentages of true lifetimes for cost combinations **C1**, **C2**, **C3**. The dots present the estimate for each system.

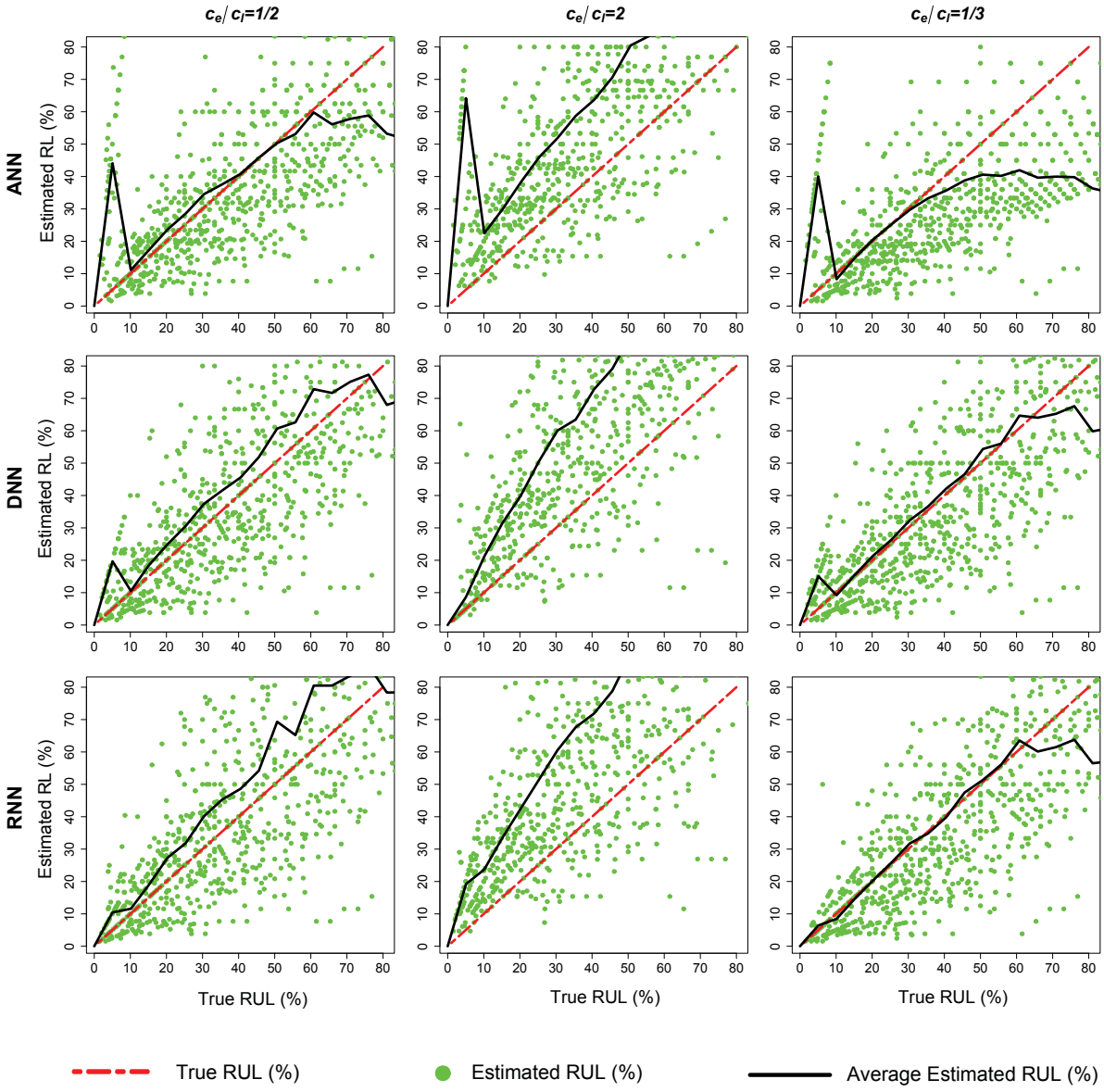


Figure 45: Distribution of RUL estimates as percentages of true lifetimes for cost combinations **C4**, **C5**, **C6**. The dots present the estimate for each system.

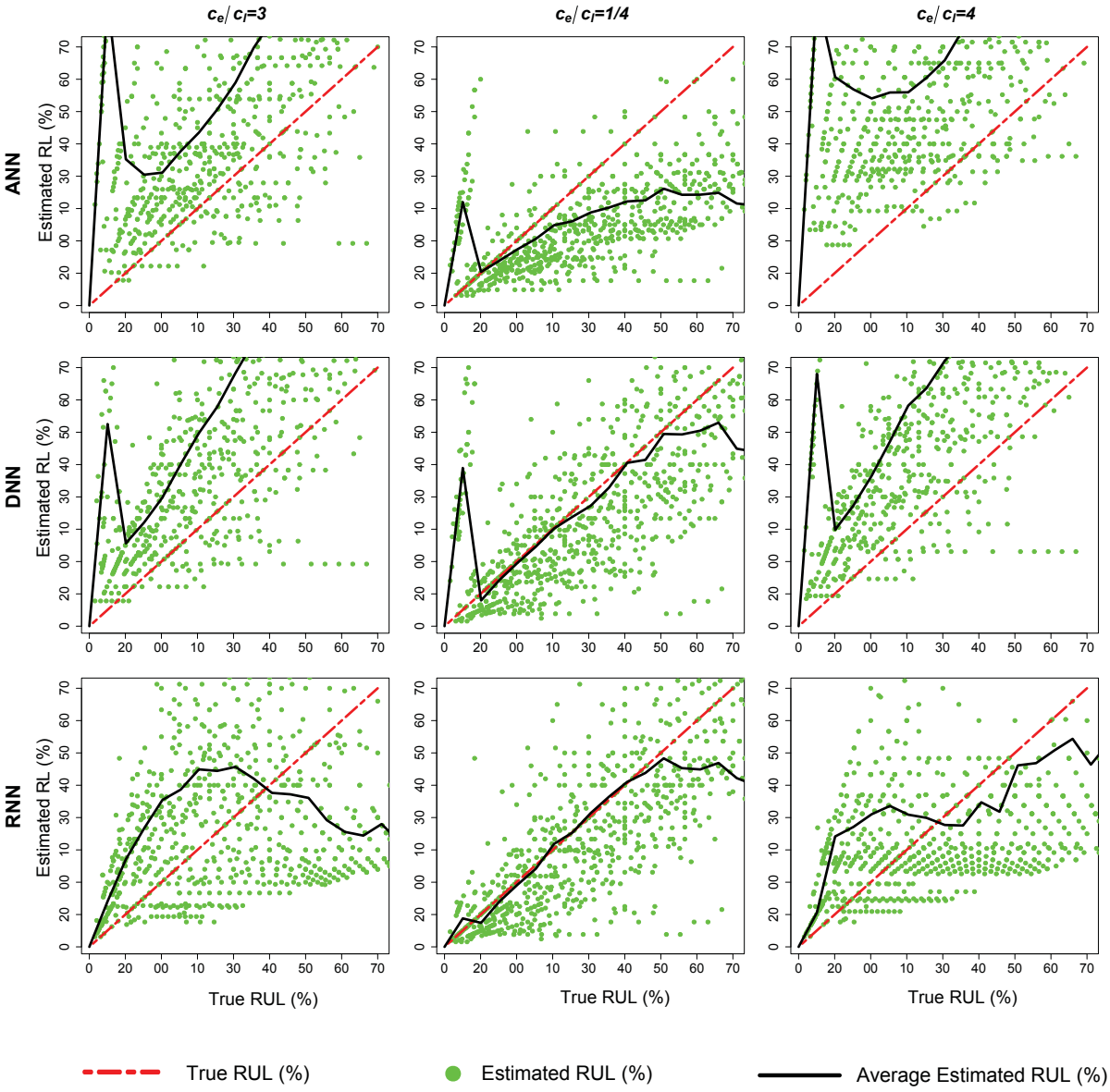


Figure 46: Distribution of RUL estimates as percentages of true lifetimes for cost combinations **C7**, **C8**, **C9**. The dots present the estimate for each system.

CMAPSS Data

The same results as above, for the CMAPSS data, are given in Figures 47,48,49. The plots shown verify the validity of the proposed RUL estimation framework. In the case in which $c_e = c_l$, the average estimated RUL almost equals the true RUL, especially closer to the actual system failure. Furthermore, when $c_e < c_l$, the agents suggest earlier replacement times, whereas the opposite occurs when $c_e > c_l$.

Summary

In both cases, the RUL estimation results are more accurate when the cost of late warning is greater or equal to cost of early warning. That suggests that the agents become risk-averse, since they are motivated to give warning signals almost always before the actual value.

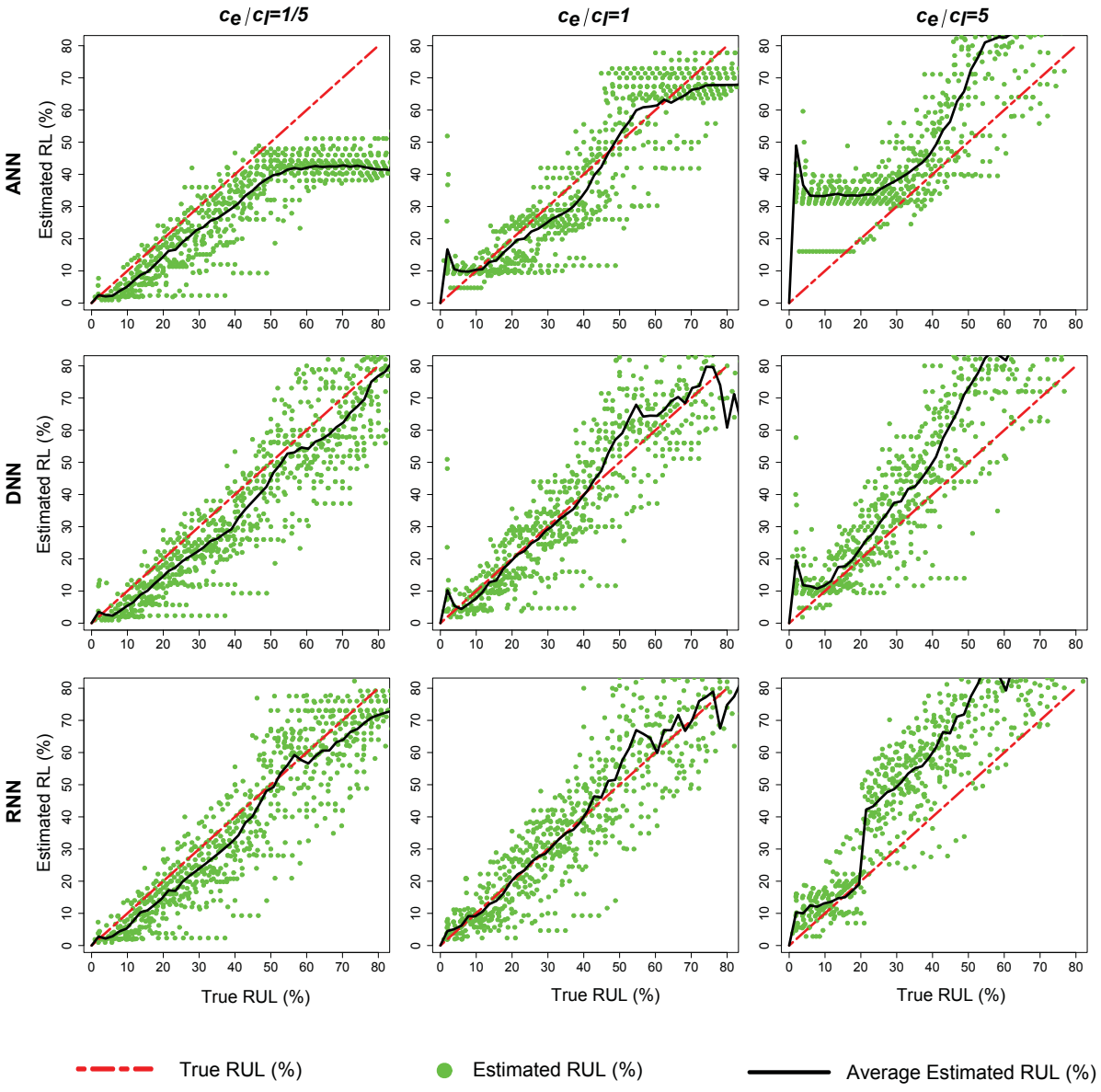


Figure 47: Distribution of RUL estimates as % of true lifetimes for cost combinations **C1**, **C2**, **C3**.

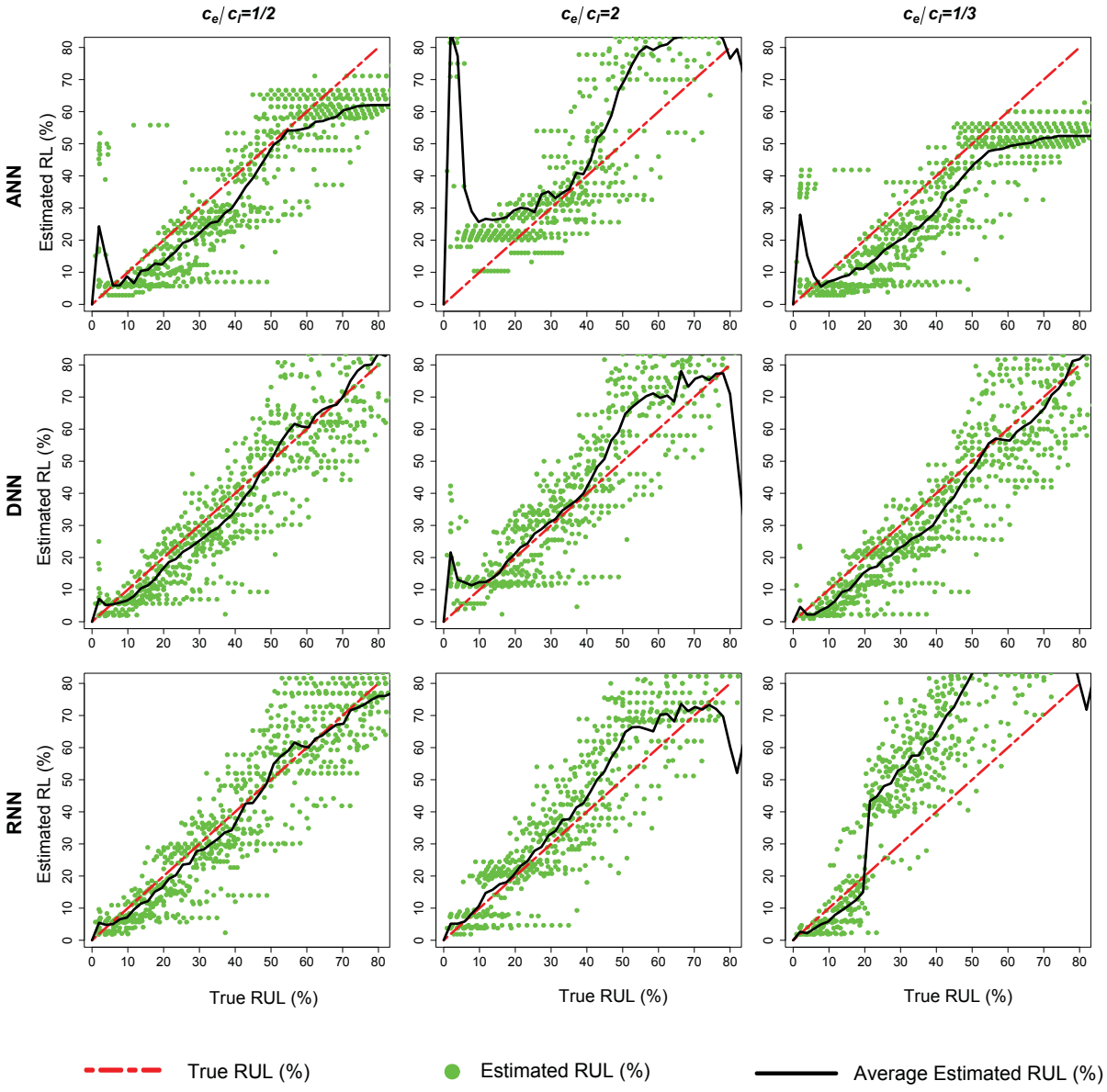


Figure 48: Distribution of RUL estimates as % of true lifetimes for cost combinations **C4**, **C5**, **C6**.

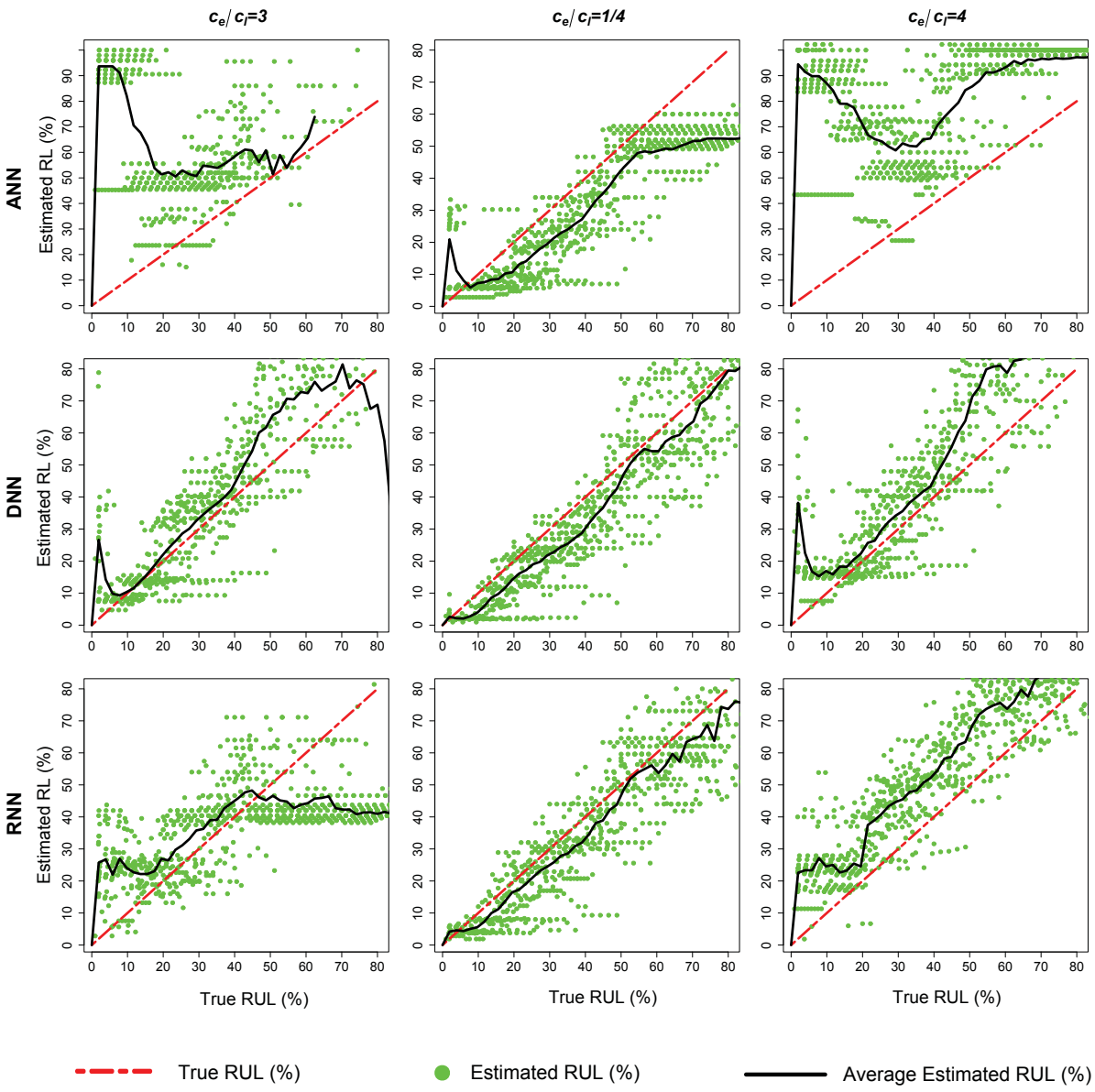


Figure 49: Distribution of RUL estimates as % of true lifetimes for cost combinations **C7**, **C8**, **C9**.

Bibliography

[1] L. Xu and M.-Y. Chow, "A classification approach for power distribution systems fault cause identification," *IEEE Transactions on Power Systems*, vol. 21, no. 1, pp. 53–60, 2006.

[2] B. Chen, X. Chen, B. Li, Z. He, H. Cao, and G. Cai, "Reliability estimation for cutting tools based on logistic regression model using vibration signals," *Mechanical Systems and Signal Processing*, vol. 25, no. 7, pp. 2526–2537, 2011.

[3] H. Li, Y. Wang, P. Zhao, X. Zhang, and P. Zhou, "Cutting tool operational reliability prediction based on acoustic emission and logistic regression model," *Journal of Intelligent Manufacturing*, vol. 26, no. 5, pp. 923–931, 2015.

[4] A. Kumar, R. Shankar, A. Choudhary, and L. S. Thakur, "A big data mapreduce framework for fault diagnosis in cloud-based manufacturing," *International Journal of Production Research*, vol. 54, no. 23, pp. 7060–7073, 2016.

[5] W. Bukhari and S. Hong, "Real-time prediction and gating of respiratory motion in 3d space using extended kalman filters and gaussian process regression network," *Physics in Medicine & Biology*, vol. 61, no. 5, p. 1947, 2016.

[6] L. Bai, Z. Li, and P. Guo, "Classification of stellar spectral data based on kalman filter and rbf neural networks," in *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*, vol. 1. IEEE, 2003, pp. 274–279.

[7] Y. Wen, J. Wu, and Y. Yuan, "Multiple-phase modeling of degradation signal for condition monitoring and remaining useful life prediction," *IEEE Transactions on Reliability*, vol. 66, no. 3, pp. 924–938, 2017.

[8] Y. Wen, J. Wu, Q. Zhou, and T.-L. Tseng, "Multiple-change-point modeling and exact bayesian inference of degradation signal for prognostic improvement," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 2, pp. 613–628, 2019.

[9] A. P. Ompusunggu, J.-M. Papy, and S. Vandenplas, "Kalman-filtering-based prognostics for automatic transmission clutches," *IEEE/ASME Transactions on Mechatronics*, vol. 21, no. 1, pp. 419–430, 2015.

[10] J. Chen, C. Ma, D. Song, and B. Xu, "Failure prognosis of multiple uncertainty system based on kalman filter and its application to aircraft fuel system," *Advances in Mechanical Engineering*, vol. 8, no. 10, p. 1687814016671445, 2016.

[11] L. C. K. Reuben and D. Mba, "Diagnostics and prognostics using switching kalman filters," *Structural Health Monitoring*, vol. 13, no. 3, pp. 296–306, 2014.

[12] X. Tang, M. Xiao, and B. Hu, "Application of kalman filter to model-based prognostics for solenoid valve," *Soft Computing*, pp. 1–13, 2019.

[13] G. Kim, H. Kim, E. Zio, and G. Heo, "Application of particle filtering for prognostics with measurement uncertainty in nuclear power plants," *Nuclear Engineering and Technology*, vol. 50, no. 8, pp. 1314–1323, 2018.

[14] B. Saha, K. Goebel, S. Poll, and J. Christophersen, "Prognostics methods for battery health monitoring using a bayesian framework," *IEEE Transactions on instrumentation and measurement*, vol. 58, no. 2, pp. 291–296, 2008.

[15] L. Saidi, J. B. Ali, E. Bechhoefer, and M. Benbouzid, "Particle filter-based prognostic approach for high-speed shaft bearing wind turbine progressive degradations," in *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2017, pp. 8099–8104.

[16] C. Yang, Q. Lou, J. Liu, Y. Yang, and Y. Bai, "Particle filter-based method for prognostics with application to auxiliary power unit," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2014, pp. 198–207.

[17] Y. Wang, S. Geng, and H. Gao, "A proactive decision support method based on deep reinforcement learning and state partition," *Knowledge-Based Systems*, vol. 143, pp. 248–258, 2018.

[18] S. Spielberg, A. Tulsyan, N. P. Lawrence, P. D. Loewen, and R. B. Gopaluni, "Towards self-driving processes: A deep reinforcement learning approach to control," *AIChE Journal*, 2019.

[19] Y.-T. Chen, W.-N. Lai, and E. W. Sun, "Jump detection and noise separation by a singular wavelet method for predictive analytics of high-frequency data," *Computational Economics*, pp. 1–36, 2019.

[20] J. Zhang, G. Welch, N. Ramakrishnan, and S. Rahman, "Kalman filters for dynamic and secure smart grid state estimation," *Intelligent Industrial Systems*, vol. 1, no. 1, pp. 29–36, 2015.

[21] M. M. Rana, L. Li, and S. W. Su, "Microgrid state estimation and control using kalman filter and semidefinite programming technique," *International Energy Journal*, vol. 16, no. 2, 2016.

[22] J. Zhu, Z. Shi, H. Liang, R. Lu, and X. Shen, "Particle filter based grid synchronization with voltage unbalance and frequency variation in smart grid," in *2013 International Conference on Wireless Communications and Signal Processing*. IEEE, 2013, pp. 1–6.

[23] X. Liu, L. Li, Z. Li, X. Chen, T. Fernando, H. H.-C. Iu, and G. He, "Event-trigger particle filter for smart grids with limited communication bandwidth infrastructure," *IEEE Transactions on Smart Grid*, vol. 9, no. 6, pp. 6918–6928, 2017.

[24] M. Glavic, R. Fonteneau, and D. Ernst, "Reinforcement learning for electric power system decision and control: Past considerations and perspectives," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6918–6927, 2017.

[25] M. Peters, W. Ketter, M. Saar-Tsechansky, and J. Collins, "A reinforcement learning approach to autonomous decision-making in smart electricity markets," *Machine Learning*, vol. 92, no. 1, pp. 5–39, 2013.

[26] J.-W. Lee, "Design of kalman filter to estimate heart rate variability from ppg signal for mobile healthcare," *Journal of Information and Communication Convergence Engineering*, vol. 8, no. 2, pp. 201–204, 2010.

[27] K. Gordon, "The multi-state kalman filter in medical monitoring," *Computer Methods and Programs in Biomedicine*, vol. 23, no. 2, pp. 147–154, 1986.

[28] A. Krengel, J. Hauth, M.-R. Taskinen, M. Adiels, and M. Jirstrand, "A continuous-time adaptive particle filter for estimations under measurement time uncertainties with an application to a plasma-leucine mixed effects model," *BMC Systems Biology*, vol. 7, no. 1, p. 8, 2013.

[29] L. Fischer, "Using shape particle filters for robust medical image segmentation," *Tech. Report-Vienna University of Technology, Tech. Rep.*, 2009.

[30] C. Yu, J. Liu, and S. Nemati, "Reinforcement learning in healthcare: A survey," *arXiv preprint arXiv:1908.08796*, 2019.

[31] O. Gottesman, F. Johansson, M. Komorowski, A. Faisal, D. Sontag, F. Doshi-Velez, and L. A. Celi, "Guidelines for reinforcement learning in healthcare," *Nat Med*, vol. 25, no. 1, pp. 16–18, 2019.

[32] E. Ruschel, E. A. P. Santos, and E. d. F. R. Loures, "Industrial maintenance decision-making: A systematic literature review," *Journal of Manufacturing Systems*, vol. 45, pp. 180–194, 2017.

[33] T. Xia, X. Jin, L. Xi, Y. Zhang, and J. Ni, "Operating load based real-time rolling grey forecasting for machine health prognosis in dynamic maintenance schedule," *Journal of Intelligent Manufacturing*, vol. 26, no. 2, pp. 269–280, 2015.

[34] A. Ghasemi, S. Yacout, and M. Ouali, "Optimal condition based maintenance with imperfect information and the proportional hazards model," *International Lournal of Production Research*, vol. 45, no. 4, pp. 989–1012, 2007.

[35] T. Xia, X. Jin, L. Xi, and J. Ni, "Production-driven opportunistic maintenance for batch production based on mam–apb scheduling," *European Journal of Operational Research*, vol. 240, no. 3, pp. 781–790, 2015.

[36] A. Chen and G. Wu, "Real-time health prognosis and dynamic preventive maintenance policy for equipment under aging markovian deterioration," *International Journal of Production Research*, vol. 45, no. 15, pp. 3351–3379, 2007.

[37] A. K. Jardine, D. Lin, and D. Banjevic, "A review on machinery diagnostics and prognostics implementing condition-based maintenance," *Mechanical Systems and Signal Processing*, vol. 20, no. 7, pp. 1483–1510, 2006.

[38] R. Ahmad and S. Kamaruddin, "A review of condition-based maintenance decision-making," *European Journal of Industrial Engineering*, vol. 6, no. 5, pp. 519–541, 2012.

[39] A. Bousdekis, B. Magoutas, D. Apostolou, and G. Mentzas, "Review, analysis and synthesis of prognostic-based decision support methods for condition based maintenance," *Journal of Intelligent Manufacturing*, vol. 29, no. 6, pp. 1303–1316, 2018.

[40] Y. Lei, N. Li, L. Guo, N. Li, T. Yan, and J. Lin, "Machinery health prognostics: A systematic review from data acquisition to rul prediction," *Mechanical Systems and Signal Processing*, vol. 104, pp. 799–834, 2018.

[41] F. X. Diebold, "State space modeling of time series: a review essay," *Journal of Economic Dynamics and Control*, vol. 13, no. 4, pp. 597–612, 1989.

[42] D. An, N. H. Kim, and J.-H. Choi, "Practical options for selecting data-driven or physics-based prognostics algorithms with reviews," *Reliability Engineering & System Safety*, vol. 133, pp. 223–236, 2015.

[43] A. Ray and S. Tangirala, "Stochastic modeling of fatigue crack dynamics for on-line failure prognostics," *IEEE Transactions on Control Systems Technology*, vol. 4, no. 4, pp. 443–451, 1996.

[44] R. K. Singleton, E. G. Strangas, and S. Aviyente, "Extended kalman filtering for remaining-useful-life estimation of bearings," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 3, pp. 1781–1790, 2014.

[45] C. K. R. Lim and D. Mba, "Switching kalman filter for failure prognostic," *Mechanical Systems and Signal Processing*, vol. 52, pp. 426–435, 2015.

[46] X. Zhang and J. Zeng, "Deterioration state space partitioning method for opportunistic maintenance modelling of identical multi-unit systems," *International Journal of Production Research*, vol. 53, no. 7, pp. 2100–2118, 2015.

[47] D. Liu, X. Yin, Y. Song, W. Liu, and Y. Peng, "An on-line state of health estimation of lithium-ion battery using unscented particle filter," *IEEE Access*, vol. 6, pp. 40 990–41 001, 2018.

[48] J. Sun, H. Zuo, W. Wang, and M. G. Pecht, "Application of a state space modeling technique to system prognostics based on a health index for condition-based maintenance," *Mechanical Systems and Signal Processing*, vol. 28, pp. 585–596, 2012.

[49] X. Xu and N. Chen, "A state-space-based prognostics model for lithium-ion battery degradation," *Reliability Engineering & System Safety*, vol. 159, pp. 47–57, 2017.

[50] Z. Ghahramani, "An introduction to hidden markov models and bayesian networks," in *Hidden Markov models: applications in computer vision*. World Scientific, 2001, pp. 9–41.

[51] D. Tobon-Mejia, K. Medjaher, and N. Zerhouni, "Cnc machine tool's wear diagnostic and prognostic by using dynamic bayesian networks," *Mechanical Systems and Signal Processing*, vol. 28, pp. 167–182, 2012.

[52] C. Andrieu, M. Davy, and A. Doucet, "Efficient particle filtering for jump markov systems. application to time-varying autoregressions," *IEEE Transactions on Signal Processing*, vol. 51, no. 7, pp. 1762–1770, 2003.

[53] K. Salahshoor and M. F. Samadi, "A novel fuzzy approach for state estimation of nonlinear hybrid systems using particle filtering method," *Asian Journal of Control*, vol. 14, no. 4, pp. 974–990, 2012.

[54] T. Wei, Y. Huang, and C. P. Chen, "Adaptive sensor fault detection and identification using particle filter algorithms," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 39, no. 2, pp. 201–213, 2009.

[55] M. E. Orchard and G. J. Vachtsevanos, "A particle-filtering approach for on-line fault diagnosis and failure prognosis," *Transactions of the Institute of Measurement and Control*, vol. 31, no. 3-4, pp. 221–246, 2009.

[56] V. Climente-Alarcon, J. A. Antonino-Daviu, E. G. Strangas, and M. Riera-Guasp, "Rotor-bar breakage mechanism and prognosis in an induction motor," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 3, pp. 1814–1825, 2014.

[57] S. Butler, "Prognostic algorithms for condition monitoring and remaining useful life estimation," Ph.D. dissertation, National University of Ireland Maynooth, 2012.

[58] D. Wang and K.-L. Tsui, "Brownian motion with adaptive drift for remaining useful life prediction: Revisited," *Mechanical Systems and Signal Processing*, vol. 99, pp. 691–701, 2018.

[59] E. Skordilis and R. Moghaddass, "A condition monitoring approach for real-time monitoring of degrading systems using kalman filter and logistic regression," *International Journal of Production Research*, vol. 55, no. 19, pp. 5579–5596, 2017.

[60] Y. Song, D. Liu, C. Yang, and Y. Peng, "Data-driven hybrid remaining useful life estimation approach for spacecraft lithium-ion battery," *Microelectronics Reliability*, vol. 75, pp. 142–153, 2017.

[61] Y. Jiang, Y. Wang, Y. Wu, and Q. Sun, "Fault prognostic of electronics based on optimal multi-order particle filter," *Microelectronics Reliability*, vol. 62, pp. 167–177, 2016.

[62] J. M. Ko and C. O. Kim, "A multivariate parameter trace analysis for online fault detection in a semiconductor etch tool," *International Journal of Production Research*, vol. 50, no. 23, pp. 6639–6654, 2012.

[63] S. Yang, "A condition-based failure-prediction and processing-scheme for preventive maintenance," *IEEE Transactions on Reliability*, vol. 52, no. 3, pp. 373–383, 2003.

[64] S. Yang and T. Liu, "State estimation for predictive maintenance using kalman filter," *Reliability Engineering & System Safety*, vol. 66, no. 1, pp. 29–39, 1999.

[65] P. Lall, R. Lowe, and K. Goebel, "Prognostics health management of electronic systems under mechanical shock and vibration using kalman filter models and metrics," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 11, pp. 4301–4314, 2012.

[66] D. C. Swanson, J. M. Spencer, and S. H. Arzoumanian, "Prognostic modelling of crack growth in a tensioned steel band," *Mechanical Systems and Signal Processing*, vol. 14, no. 5, pp. 789–803, 2000.

[67] M. Gašperin, D. Juričić, P. Boškoski, and J. Vižintin, "Model-based prognostics of gear health using stochastic dynamical models," *Mechanical Systems and Signal Processing*, vol. 25, no. 2, pp. 537–548, 2011.

[68] A. Christer, W. Wang, and J. Sharp, "A state space condition monitoring model for furnace erosion prediction and replacement," *European Journal of Operational Research*, vol. 101, no. 1, pp. 1–14, 1997.

[69] M. Bressel, M. Hilairet, D. Hissel, and B. O. Bouamama, "Extended kalman filter for prognostic of proton exchange membrane fuel cell," *Applied Energy*, vol. 164, pp. 220–227, 2016.

[70] X. Zhang and P. Pisu, "Prognostic-oriented fuel cell catalyst aging modeling and its application to health-monitoring and prognostics of a pem fuel cell," *International Journal of Prognostics and Health Management*, vol. 5, no. 1-003, pp. 1–16, 2014.

[71] X. Zheng and H. Fang, "An integrated unscented kalman filter and relevance vector regression approach for lithium-ion battery remaining useful life and short-term capacity prediction," *Reliability Engineering & System Safety*, vol. 144, pp. 74–82, 2015.

[72] Z. Chen *et al.*, "Bayesian filtering: From kalman filters to particle filters, and beyond," *Statistics*, vol. 182, no. 1, pp. 1–69, 2003.

[73] A. Doucet, S. Godsill, and C. Andrieu, "On sequential monte carlo sampling methods for bayesian filtering," *Statistics and Computing*, vol. 10, no. 3, pp. 197–208, 2000.

[74] H. Tanizaki and R. S. Mariano, "Nonlinear and non-gaussian state-space modeling with monte carlo simulations," *Journal of Econometrics*, vol. 83, no. 1-2, pp. 263–290, 1998.

[75] B. Zhao, R. Skjetne, M. Blanke, and F. Dukan, "Particle filter for fault diagnosis and robust navigation of underwater robot," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2399–2407, 2014.

[76] V. Verma, G. Gordon, R. Simmons, and S. Thrun, "Real-time fault diagnosis [robot fault diagnosis]," *IEEE Robotics & Automation Magazine*, vol. 11, no. 2, pp. 56–66, 2004.

[77] Z. Duan, Z. Cai, and H. Min, "Robust dead reckoning system for mobile robots based on particle filter and raw range scan," *Sensors*, vol. 14, no. 9, pp. 16 532–16 562, 2014.

[78] N. De Freitas, R. Dearden, F. Hutter, R. Morales-Menendez, J. Mutch, and D. Poole, "Diagnosis by a waiter and a mars explorer," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 455–468, 2004.

[79] J. M. Pak, C. K. Ahn, Y. S. Shmaliy, and M. T. Lim, "Improving reliability of particle filter-based localization in wireless sensor networks via hybrid particle/fir filtering," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 5, pp. 1089–1098, 2015.

[80] F. Caron, M. Davy, E. Duflos, and P. Vanheeghe, "Particle filtering for multisensor data fusion with switching observation models: Application to land vehicle positioning," *IEEE Transactions on Signal Processing*, vol. 55, no. 6, pp. 2703–2719, 2007.

[81] Y. Boers and J. Driessen, "Interacting multiple model particle filter," *IEE Proceedings-Radar, Sonar and Navigation*, vol. 150, no. 5, pp. 344–349, 2003.

[82] ——, "Multitarget particle filter track before detect application," *IEE Proceedings-Radar, Sonar and Navigation*, vol. 151, no. 6, pp. 351–357, 2004.

[83] S. Park, J. P. Hwang, E. Kim, and H.-J. Kang, "Vehicle tracking using a microwave radar for situation awareness," *Control Engineering Practice*, vol. 18, no. 4, pp. 383–395, 2010.

[84] S. Tafazoli and X. Sun, "Hybrid system state tracking and fault detection using particle filters," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 6, pp. 1078–1087, 2006.

[85] D. Wang, S. Sun, and W. T. Peter, "A general sequential monte carlo method based optimal wavelet filter: A bayesian approach for extracting bearing fault features," *Mechanical Systems and Signal Processing*, vol. 52, pp. 293–308, 2015.

[86] H. A. Blom and E. A. Bloem, "Exact bayesian and particle filtering of stochastic hybrid systems," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 43, no. 1, pp. 55–70, 2007.

[87] W. J. Godinez, M. Lampe, P. Koch, R. Eils, B. Muller, and K. Rohr, "Identifying virus-cell fusion in two-channel fluorescence microscopy image sequences based on a layered probabilistic approach," *IEEE Transactions on Medical Imaging*, vol. 31, no. 9, pp. 1786–1808, 2012.

[88] M. Jouin, R. Gouriveau, D. Hissel, M.-C. Péra, and N. Zerhouni, "Particle filter-based prognostics: Review, discussion and perspectives," *Mechanical Systems and Signal Processing*, vol. 72, pp. 2–31, 2016.

[89] F. Camci and R. B. Chinnam, "Health-state estimation and prognostics in machining processes," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 3, pp. 581–597, 2010.

[90] P. Wang and R. X. Gao, "Automated performance tracking for heat exchangers in hvac," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 634–645, 2017.

[91] L. Feng, H. Wang, X. Si, and H. Zou, "A state-space-based prognostic model for hidden and age-dependent nonlinear degradation process," *IEEE Transactions on Automation Science and Engineering*, vol. 10, no. 4, pp. 1072–1086, 2013.

[92] M. Orchard, L. Tang, B. Saha, K. Goebel, and G. Vachtsevanos, "Risk-sensitive particle-filtering-based prognosis framework for estimation of remaining useful life in energy storage devices," *Studies in Informatics and Control*, vol. 19, no. 3, pp. 209–218, 2010.

[93] B. Zhang, C. Sconyers, C. Byington, R. Patrick, M. E. Orchard, and G. Vachtsevanos, "A probabilistic fault detection approach: Application to bearing fault detection," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 5, pp. 2011–2018, 2010.

[94] F. Lu, W. Zheng, J. Huang, and M. Feng, "Life cycle performance estimation and in-flight health monitoring for gas turbine engine," *Journal of Dynamic Systems, Measurement, and Control*, vol. 138, no. 9, p. 091009, 2016.

[95] M. S. Jha, M. Bressel, B. Ould-Bouamama, and G. Dauphin-Tanguy, "Particle filter based hybrid prognostics of proton exchange membrane fuel cell in bond graph framework," *Computers & Chemical Engineering*, vol. 95, pp. 216–230, 2016.

[96] B. E. Olivares, M. A. C. Munoz, M. E. Orchard, and J. F. Silva, "Particle-filtering-based prognosis framework for energy storage devices with a statistical characterization of state-of-health regeneration phenomena," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 2, pp. 364–376, 2012.

[97] W. Yan, B. Zhang, X. Wang, W. Dou, and J. Wang, "Lebesgue-sampling-based diagnosis and prognosis for lithium-ion batteries," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 3, pp. 1804–1812, 2015.

[98] D. Wang, F. Yang, K.-L. Tsui, Q. Zhou, and S. J. Bae, "Remaining useful life prediction of lithium-ion batteries based on spherical cubature particle filter," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 6, pp. 1282–1291, 2016.

[99] C. Andrieu, A. Doucet, and R. Holenstein, "Particle markov chain monte carlo methods," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 72, no. 3, pp. 269–342, 2010.

[100] N. Chopin, P. E. Jacob, and O. Papaspiliopoulos, "Smc2: an efficient algorithm for sequential analysis of state space models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 75, no. 3, pp. 397–426, 2013.

[101] E. Tsoutsanis and N. Meskin, "Derivative-driven window-based regression method for gas turbine performance prognostics," *Energy*, vol. 128, pp. 302–311, 2017.

[102] C. Duan, C. Deng, and B. Wang, "Optimal multi-level condition-based maintenance policy for multi-unit systems under economic dependence," *The International Journal of Advanced Manufacturing Technology*, vol. 91, no. 9-12, pp. 4299–4312, 2017.

[103] F. Cheng, L. Qu, and W. Qiao, "Fault prognosis and remaining useful life prediction of wind turbine gearboxes using current signal analysis," *IEEE Transactions on Sustainable Energy*, vol. 9, no. 1, pp. 157–167, 2017.

[104] R. Kontar, J. Son, S. Zhou, C. Sankavaram, Y. Zhang, and X. Du, "Remaining useful life prediction based on the mixed effects model with mixture prior distribution," *IISE Transactions*, vol. 49, no. 7, pp. 682–697, 2017.

[105] P. Baraldi, G. Bonfanti, and E. Zio, "Differential evolution-based multi-objective optimization for the definition of a health indicator for fault diagnostics and prognostics," *Mechanical Systems and Signal Processing*, vol. 102, pp. 382–400, 2018.

[106] D. Barraza-Barraza, V. G. Tercero-Gómez, M. G. Beruvides, and J. Limón-Robles, "An adaptive arx model to estimate the rul of aluminum plates based on its crack growth," *Mechanical Systems and Signal Processing*, vol. 82, pp. 519–536, 2017.

[107] Y. Qian and R. Yan, "Remaining useful life prediction of rolling bearings using an enhanced particle filter," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 10, pp. 2696–2707, 2015.

[108] M. Ibrahim, N. Y. Steiner, S. Jemei, and D. Hissel, "Wavelet-based approach for online fuel cell remaining useful lifetime prediction," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 8, pp. 5057–5068, 2016.

[109] X. Jin, Y. Sun, Z. Que, Y. Wang, and T. W. Chow, "Anomaly detection and fault prognosis for bearings," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 9, pp. 2046–2054, 2016.

[110] Z. Huang, Z. Xu, X. Ke, W. Wang, and Y. Sun, "Remaining useful life prediction for an adaptive skew-wiener process model," *Mechanical Systems and Signal Processing*, vol. 87, pp. 294–306, 2017.

[111] N. Li, Y. Lei, L. Guo, T. Yan, and J. Lin, "Remaining useful life prediction based on a general expression of stochastic process models," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 7, pp. 5709–5718, 2017.

[112] H.-K. Wang, H.-Z. Huang, Y.-F. Li, and Y.-J. Yang, "Condition-based maintenance with scheduling threshold and maintenance threshold," *IEEE Transactions on Reliability*, vol. 65, no. 2, pp. 513–524, 2015.

[113] W. Peng, Y.-F. Li, Y.-J. Yang, S.-P. Zhu, and H.-Z. Huang, "Bivariate analysis of incomplete degradation observations based on inverse gaussian processes and copulas," *IEEE Transactions on Reliability*, vol. 65, no. 2, pp. 624–639, 2016.

[114] A. Heng, S. Zhang, A. C. Tan, and J. Mathew, "Rotating machinery prognostics: State of the art, challenges and opportunities," *Mechanical Systems and Signal Processing*, vol. 23, no. 3, pp. 724–739, 2009.

[115] D. Banjevic, A. Jardine, V. Makis, and M. Ennis, "A control-limit policy and software for condition-based maintenance optimization," *INFOR: Information Systems and Operational Research*, vol. 39, no. 1, pp. 32–50, 2001.

[116] F. Yang, M. S. Habibullah, T. Zhang, Z. Xu, P. Lim, and S. Nadarajan, "Health index-based prognostics for remaining useful life predictions in electrical machines," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 4, pp. 2633–2644, 2016.

[117] J. Liu and E. Zio, "An adaptive online learning approach for support vector regression: Online-svr-fid," *Mechanical Systems and Signal Processing*, vol. 76, pp. 796–809, 2016.

[118] R. Khelif, B. Chebel-Morello, S. Malinowski, E. Laajili, F. Fnaiech, and N. Zerhouni, "Direct remaining useful life estimation based on support vector regression," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 3, pp. 2276–2285, 2016.

[119] A. Zhang, H. Wang, S. Li, Y. Cui, Z. Liu, G. Yang, and J. Hu, "Transfer learning with deep recurrent neural networks for remaining useful life estimation," *Applied Sciences*, vol. 8, no. 12, p. 2416, 2018.

[120] J. Deutsch, M. He, and D. He, "Remaining useful life prediction of hybrid ceramic bearings using an integrated deep learning and particle filter approach," *Applied Sciences*, vol. 7, no. 7, p. 649, 2017.

[121] M. C. O. Keizer, R. H. Teunter, and J. Veldman, "Clustering condition-based maintenance for systems with redundancy and economic dependencies," *European Journal of Operational Research*, vol. 251, no. 2, pp. 531–540, 2016.

[122] E. Byon and Y. Ding, "Season-dependent condition-based maintenance for a wind turbine using a partially observed markov decision process," *IEEE Transactions on Power Systems*, vol. 25, no. 4, pp. 1823–1834, 2010.

[123] X. Zhou, L. Xi, and J. Lee, "Opportunistic preventive maintenance scheduling for a multi-unit series system based on dynamic programming," *International Journal of Production Economics*, vol. 118, no. 2, pp. 361–366, 2009.

[124] M. J. Kim, "Robust control of partially observable failing systems," *Operations Research*, vol. 64, no. 4, pp. 999–1014, 2016.

[125] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*. Athena Scientific Belmont, MA, 1996, vol. 5.

[126] A. Xanthopoulos, A. Kiatipis, D. E. Koulouriotis, and S. Stieger, "Reinforcement learning-based and parametric production-maintenance control policies for a deteriorating manufacturing system," *IEEE Access*, vol. 6, pp. 576–588, 2017.

[127] X. Wang, H. Wang, and C. Qi, "Multi-agent reinforcement learning based maintenance policy for a resource constrained flow line system," *Journal of Intelligent Manufacturing*, vol. 27, no. 2, pp. 325–333, 2016.

[128] G. Vachtsevanos, B. Lee, S. Oh, and M. Balchanos, "Resilient design and operation of cyber physical systems with emphasis on unmanned autonomous systems," *Journal of Intelligent & Robotic Systems*, vol. 91, no. 1, pp. 59–83, 2018.

[129] T. Dohi and H. Okamura, "Dynamic software availability model with rejuvenation," *Journal of the Operations Research Society of Japan*, vol. 59, no. 4, pp. 270–290, 2016.

[130] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[131] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[132] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[133] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[134] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[135] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[136] "MIT Technology Review," www.technologyreview.com/lists/technologies/2013.

[137] "MIT Technology Review," www.technologyreview.com/lists/technologies/2017.

[138] G. Welch, G. Bishop *et al.*, "An introduction to the kalman filter," 1995.

[139] D. Van Ravenzwaaij, P. Cassey, and S. D. Brown, "A simple introduction to markov chain monte–carlo sampling," *Psychonomic Bulletin & Review*, vol. 25, no. 1, pp. 143–154, 2018.

[140] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[141] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Oxford University Press*, 1970.

[142] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[143] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence properties of the nelder–mead simplex method in low dimensions," *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 112–147, 1998.

[144] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive Modeling*, vol. 5, no. 3, p. 1, 1988.

[145] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[146] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[147] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring," *Mechanical Systems and Signal Processing*, vol. 115, pp. 213–237, 2019.

[148] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, PhD thesis, University of Cambridge, 2016.

[149] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[150] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[151] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[152] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT Press Cambridge, 1998, vol. 135.

[153] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[154] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[155] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning*. The MIT Press, Cambridge, 2006.

[156] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT Press, 2012.

[157] J.-G. Wang, "Test statistics in kalman filtering," *Positioning*, vol. 1, no. 13, p. 0, 2008.

[158] P. Wang and R. X. Gao, "Markov nonlinear system estimation for engine performance tracking," *Journal of Engineering for Gas Turbines and Power*, vol. 138, no. 9, p. 091201, 2016.

[159] M. Rigamonti, P. Baraldi, E. Zio, D. Astigarraga, and A. Galarza, "Particle filter-based prognostics for an electrolytic capacitor working in variable operating conditions," *IEEE Transactions on Power Electronics*, vol. 31, no. 2, pp. 1567–1575, 2015.

[160] A. Akusok, K.-M. Björk, Y. Miche, and A. Lendasse, "High-performance extreme learning machines: a complete toolbox for big data applications," *IEEE Access*, vol. 3, pp. 1011–1025, 2015.

[161] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[162] "NASA Prognostics Data Repository Datasets," https://ti.arc.nasa.gov/tech/dash/pcoe/prognostic-datarepository.

[163] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *2008 IEEE International Conference on Prognostics and Health Management*, 2008, pp. 1–9.