A SUPERVISED FAULT TOLERANT CONTROL ARCHITECTURE FOR NONLINEAR SYSTEMS

By

PEDRO GERBASE DE LIMA

Batchelor of Science

Universidade de São Paulo

São Paulo, Brazil

2000

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
December, 2005

A SUPERVISED FAULT TOLERANT CONTROL ARCHITECTURE FOR NONLINEAR SYSTEMS

Dissertation approved:

Dr. Gary Yen

Dissertation Advisor

Dr. Martin Hagan

Dr. Rafael Fierro

Dr. R. Russell Rhinehart

Dr. A. Gordon Emslie

Dean of the Graduate College

PREFACE

This report summarizes the research conducted with the goal developing a Fault Tolerant Control (FTC) architecture capable of increasing the availability of complex nonlinear systems potentially subject to a wide range of fault scenarios. Motivated by an encompassing literature survey in the areas of fault information extraction and FTC itself, the proposed hierarchical architecture is composed of three levels.

The lowest level is composed of a baseline nonlinear reconfigurable controller that generates identification models and new control solutions for previously unknown faults. To implement such a controller as well as an identifier for fault modeling, an adaptive critic design known as Globalized Dual Heuristic Programming (GDHP) manages a set of three recurrent neural networks. The use of GDHP grants the architecture the power to preserve system stability and as much performance as possible in the presence of faults that may extend the order or add crucial nonlinearities to the dynamics of the system.

Operating on a middle level, a novel supervisor increases the reconfiguration speed of the GDHP controller for abrupt faults known at design time as well as faults autonomously modeled and addressed online during a previous occurrence. Moreover, the supervisor also increases the stability of the online GDHP reconfigurable controller by preventing malfunctions within its training algorithm (that would lead to divergence or local minima convergence) from building up to the point of degrading the tracking

performance of the plant. At the core of the supervisor, two innovative decision logics based on three quality indexes perform fault detection and diagnosis as well as controller malfunction detection. Modifying parameters of such quality indexes then adjusts the response of the supervisor to faults and controller malfunctions. The presented architecture counts with a procedure for initializing and tuning twelve design parameters to shape the supervisor's response to comply with key practical FTC specifications, ranging from maximum acceptable reconfiguration delay for abrupt known faults, to the expected maximum measurement noise level during fault scenarios.

Overviewing the entire architecture, a fault development rule extraction algorithm is positioned at the highest level. Through information gathered from the GDHP controller, identifier and from the supervisor, this final component's goal is to use all historical data from the system to build linguistic rules that inform the human mission planner (e.g., user, operator or pilot) of the probability that different fault scenarios have of taking place at particular future time frames. Once implemented, the fault development rule set will present crucial information to the mission planner when deciding if the desired trajectory of a mission should be altered after the occurrence of a minor fault to reduce the chance of a major, possibly irremediable, fault occurring. Although this final component is not yet fully developed, critical work that sets the foundation for it is presented in this report, along with the encouraging simulation results.

To substantiate the presented architecture, extensive simulation results are presented, covering both the workings of specific components and the integration of the overall architecture. The power of the algorithm can be observed in the series of proof-of-the-concept simulated systems, ranging from SISO linear systems to MIMO nonlinear

systems with unobserved states. Stability concerns involving the proposed architecture are discussed, substantiating the design choices.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1 – Introduction

1.1 Motivation

The growing complexity of physical plants and control missions inevitably leads to increasing occurrence, diversity and severity of faults. *Availability*, defined as the probability that a system or equipment will operate satisfactory and effectively at any point of time [1], becomes a factor of increasing importance. For automated production processes for example, availability is now considered to be the single factor with the highest impact on profitability [2].

The concept of local safety has been applied in practice over certain components or sub-systems of a plant to prevent continued operation or start-up if sensors (such as fuses and limit switches) indicate that conditions are met to enter a local shut down mode. Local safety though, does not necessarily lead to global fail-safe operation for the whole plant. In ship's propulsion systems where local safety is widely implemented, for example, the lack of a global treatment of a fault has resulted in many events where consequences vary from irregularity to major economic loss and casualties [3].

Fault Tolerant Control (FTC) is a field of research that aims to increase availability and reduce the risk of safety hazards and other undesirable consequences by specifically designing control algorithms capable of maintaining stability and/or performance despite the occurrence of faults [4].

In the presented work, faults are modeled as an agent of change in the plant dynamic structure. In some particular cases, information on certain fault scenarios (e.g., gear crack propagation and bearing corrosion spalling in gearbox [5]) is available beforehand, allowing the generation of specific control solutions during design time. However, the ever-growing fault diversity in complex systems makes it unrealistic to possess prior information on all possible cases. Fully solving the FTC problem during design time becomes truly intractable as we consider the fact that fault information must be in the form of models that represent the dynamics of fault scenarios precisely enough to allow the development of corresponding controllers. The general FTC problem demands online structural adaptation capability that goes beyond the adjustments of parameters in a fixed model, requiring a highly flexible reconfigurable control architecture. In addition, while an approximate linear model can often be derived for a plant operating close to its nominal point, nonlinearities introduced or augmented by a fault after its occurrence can become of paramount importance to achieve a successful new control solution [6]. Therefore, a complete FTC architecture must contain a reconfigurable controller with adaptive capabilities for the online generation of new nonlinear control solutions in response to unknown fault scenarios.

When a change on the plant dynamics occurs due to a fault, it is necessary that sufficient time be given to a reconfigurable controller (independent of its particular implementation) to experiment with the input-output response of the new dynamics before it can be expected to generate a suitable control solution. Such a time during which performance is degraded is known as reconfiguration time and we are interested in minimizing it in order to increase availability.

Although allowing some reconfiguration time is inevitable when dealing with unknown faults, faster approaches can be taken for known faults. Known faults permit that control solutions designed beforehand be implemented directly whenever they re-occur in the plant. However, in order to deal with known faults differently, it is necessary for a FTC architecture to autonomously determine when and which known fault occurs. Therefore, such an approach calls for a supervisor operating at a higher hierarchical level than the reconfigurable controller capable of performing complete Fault Detection and Diagnosis (FDD).

Another critical FTC issue arises from the fact that state-of-the-art reconfigurable controllers flexible enough to reach the widest range of fault solutions do not possess online training with guaranteed stability, especially when it is taken into consideration the fact that abrupt faults inherently cause discontinuous changes in the plant dynamics. To account for such deficiency, it is possible to observe the evolution of the training under a similar paradigm used to perform FDD in the plant. In fact, performing Controller Malfunction Detection (CMD) allows the detection of structural faults within the training procedure. If such internal detection can be made before any measurable consequences reach the plant input, the supervisor can also modify the training procedure to avoid paths that lead to instability.

Different plants require different responses to faults. For instance, plants with very restrictive safety limits require high performance even during fault scenarios, while in others have strict limits on the amount of time a known fault is allowed to act during operation before a suitable solution is implemented. Although the need to adjust the FTC response to the requirements of each plant is well accepted, no available FTC approach

provides the means to do so. For the proposed architecture, an initialization and tuning procedure is introduced to adjust twelve design parameters involved in the calculation of the quality index and in this way shape the FTC response of the supervisor in order to fulfill key FTC specification, including maximum fault detection delay, maximum performance recovery delay for known faults, and maximum acceptable tracking error under a fault scenario.

Besides the functions of the reconfigurable controller and the supervisor, one final function must be performed by an entity at an even higher hierarchical level in order to complete the proposed FTC architecture. Once a fault occurs, even if the nominal level of performance is recovered to the point of making the effects of it imperceptible to the mission planner (e.g., process operator, production manager or pilot), the fact remains that one or more of the plant's subsystems is no longer operational or with its operational range severely reduced. In such a scenario, due to the change in the controller structure, it is safe to assume that the probability of occurrence of other fault scenarios will be changed (generally increased) while a particular fault is already active. Such a change in the probability of occurrence of different fault scenarios depends not only on the current faults active in the system at a given time, but it also depends on states of the plant and, most importantly, the desired trajectory set by the mission planner.

Therefore, it is crucial for any complete FTC architecture to possess the means to inform the mission planner of the probability that a fault scenario may develop after a given time if the current desired trajectory is maintained. Furthermore, the FTC architecture must be capable of determining the probability of fault occurrence under

different trajectory alternatives before they are implemented in the plant. The following is an example of type of fault development rules we propose to extract:

IF Fault 15 (valve seal compromised) is active AND input #5 remains very low AND reference #2 remains high, **THEN** Fault 23 (loss of valve actuator) will have a 85% chance of occurrence after a delay of 30 to 45 minutes.

With such information available after the occurrence of a fault, a pilot may decide to execute a safe premature landing or an industrial process operator may decide when best to interrupt production for maintenance to minimize the overall cost impact.

1.2 Implementation

The complete FTC architecture proposed and develop in this report to attain such goals is shown in Figure 1.1. Adaptive Critic Designs (ACD) managing three Neural Networks (NN) are chosen as the reconfigurable controller due to its known effectiveness to work in noisy, nonlinear environments while making minimal assumptions regarding the nature of that environment [7]. In this report, the results of proof-of-the-concept FTC simulations show the power of ACDs when dealing with MIMO nonlinear systems subject to a multitude of challenging faults.

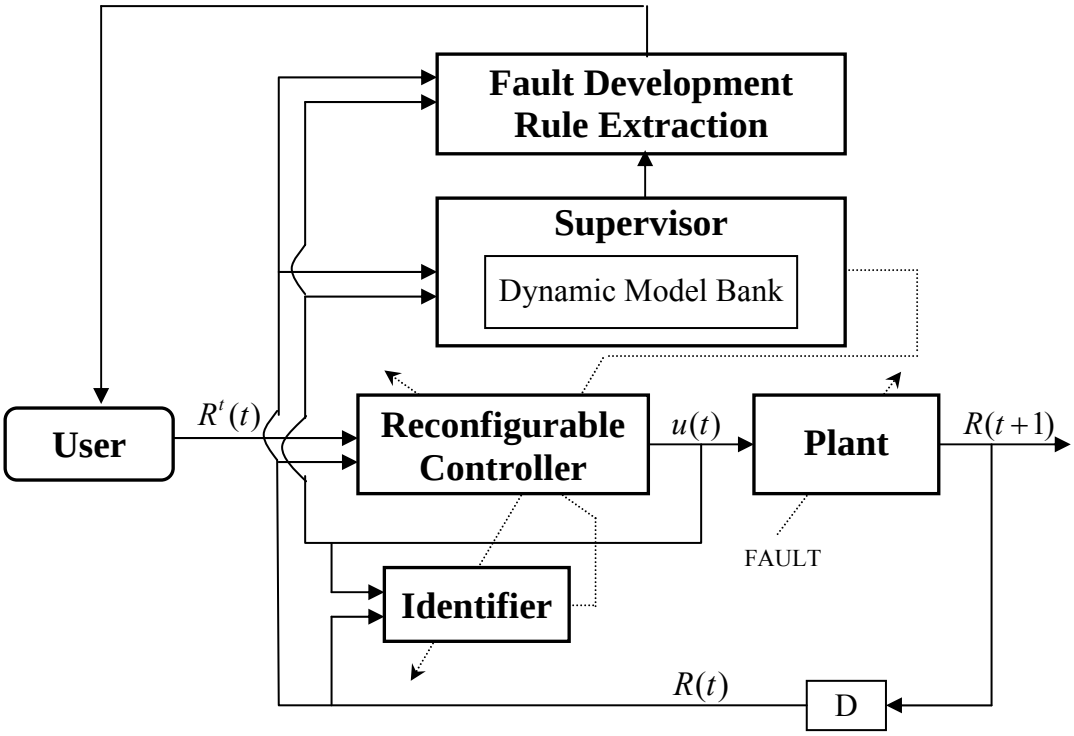


Figure 1.1. General diagram of the proposed FTC architecture.

As mentioned previously, the capability of the Supervisor to improve the response of the reconfigurable controller rests on its competence in performing complete FDD. In practice, faults can be classified in many ways. If its time profile is analyzed, faults can be categorized into incipient (causes slow changes) or abrupt (causes immediate changes). Another classification approach is to divide the faults according to the amount of knowledge the FTC controller possesses regarding each fault scenario. Known faults can make use of specific counteractions built inside the FTC scheme and therefore have a greater potential to present a faster and surest recovery than unknown ones. Unknown faults require higher levels of robustness and adaptability of the controller, but may be made known after the first occurrence, if detected, identified and in the presence of appropriate online learning capabilities. If a certain fault is expected, it may be

predictable through a stochastic analysis [8], while an unexpected fault may occur at any point in time.

In the proposed architecture, a continuously adapting identifier and a FTC supervisor work at a higher hierarchical level to track the performance of the plant online and compare its responses to a Dynamics Model Bank (DMB) within the supervisor in order to perform FDD. With the architecture introduced in this report, it is possible to detect the time of occurrence of faults, classify them into abrupt or incipient, determine if they are unknown or known, and, if known, identify them accordingly. In particular, FDD is achieved in the proposed architecture by a FDD Decision Logic nested within the supervisor using the information supplied by two quality indexes. A controller quality index measures its degree of success in tracking the desired trajectory, while an identification quality index measure the deviations between the models in the DMB and the current dynamics of the plant.

The same FDD Decision Logic is also used to determine when to add new fault identification models and control solutions to the DMB as soon as a previously unknown fault is properly dealt with. Furthermore, the FDD Decision Logic determines when to make use of the knowledge previously stored in the DMB for a particular fault scenario by performing a switching operation in order to reduce the reconfiguration time required by the online adaptive controller. Speed of reconfiguration is a crucial issue since the occurrence of a fault will move the system away from the nominal operating regime, increasing the probability of additional faults to occur until properly adjusted for. By adjusting how information on the status of the plant is gathered by the quality indexes and how it is interpreted by the supervisor's decision logics, it is possible to set the

supervisor to comply with the reconfiguration time requirements of each application. The proposed FTC architecture is presented with an offline initialization and an online tuning procedure for design parameters of the quality indexes that allow the FTC response to be adjusted to only to reconfiguration time specifications, but also to twenty four other identified FTC specifications. Simulation results highlighting the ability of the supervisor to learn the solutions to new fault scenarios online and to achieve specified reconfiguration times for abrupt known faults are also documented in this report.

Although ACDs represent the best choice of a nonlinear adaptive controller due to its capability to deal with a large scope of changes in the dynamic structure of the plant, as with any other reconfigurable controller in its class, such power comes with the tradeoff of a lack of overall stability guarantee over all possible fault scenarios. However, the development of the proposed supervisor for FDD positioned at a higher hierarchical level than the baseline controller creates a structure capable of supporting Controller Malfunction Detection (CMD). In particular, the introduction of a third quality index, the weight quality index, combined with the information generated by the controller quality index allows for a CMD Decision Logic to be generated. In this manner, the supervisor then becomes capable of detecting malfunctions within the online adaptive controller before such internal malfunctions have time to build up enough to significantly deteriorate the operation of the plant. With such information, through a similar approach used for fault switching, alternative initial conditions can be supplied to the baseline controller to prevent that such controller malfunctions lead the adaptation process to either local minima trapping or to online training divergence. The process of preventing

controller malfunctions to develop into faults that may compromise the plant is termed Controller Malfunction Recovery (CMR)

The final component of the proposed FTC architecture shown in Figure 1.1 has the goal of extracting fault development linguistic rules to aid in the mission planning of the user, operator or pilot of the system in question. To take steps toward implementing it, we present a novel autonomous rule extraction approach capable of obtaining temporal linguistic rules directly from continuous process data using Fuzzy Logic representation and a Multiple Objective Evolutionary Algorithm.

This report is organized as follows. Chapter 2 provides a literature survey on the most relevant topics of FDD and FTC, covering a series of different existing approaches that go beyond the application of an adaptive controller alone. In Chapter 3 a series of ACDs implemented during the course of this research as the baseline nonlinear adaptive controller are introduced, culminating with a NN implementation of GDHP. Chapter 4 provides a detailed explanation of the proposed supervisor and the novel FDD Decision Logic used in conjunction to greatly reduce the reconfiguration time of the baseline controller.

Chapter 5 covers the fundamentals of the proposed CMD and CMR additions to the supervisor, as well as provides proof-of-the-concept results that make clear the benefits brought by the implementation of the proposed supervisor. Chapter 6 then introduces the novel autonomous linguistic rule extraction approach that will serve as the basis for the development of the fault development rule extraction feature. Chapter 7 introduces extended versions of all three quality indexes and presents the methodologies for their offline initialization and online tuning in order to adjust the supervisor's

response to match key FTC specifications and restrictions. With the proposed FTC architecture fully drawn, Chapter 8 offers a discussion on the overall stability of the approach, indicating points of concern and justifying design choices in the light of their application in the whole architecture. Finally, conclusions concerning the entire architecture and future work directions are drawn in Chapter 9.

CHAPTER 2 – Literature Survey

2.1. Extracting fault information

The performance of a controller crafted to deal with faulty systems can be largely affected by the amount of information it is capable of gathering about the actual status of the system. Techniques for extracting fault information from systems now compose an active field of research, and some definitions are required to be introduced before further discussions on the available methods and its limitations.

As suggested by the Safeprocess Technical Committee of IFAC [7], a fault is defined as an unpermitted deviation of at least one characteristic property or parameter of the system from acceptable/usual/standard condition. Failures, on the other hand, are defined as the permanent interruption of a system to perform a required function under specific operating conditions. Under this point of view, it can be stated that the FTC's goal is to, in the event of a fault, reconfigure the system dynamics to prevent the build up of a system failure.

Faults can be classified in many ways. If its time profile is analyzed, faults can be separated into incipient (generating slow changes) and abrupt (generating fast changes) classes. Another classification approach is to divide the faults according to the amount of knowledge the FTC controller possesses regarding each fault scenario. Expected faults may have specific counteractions built inside the FTC scheme during the design phase and therefore have a greater potentiality to present a faster and surest recovery than

unexpected ones. Unexpected faults require higher levels of robustness and adaptability of the controller, but may be made expected after the first occurrence, if detected and identified, in the presence of automated learning capabilities. If a certain fault is expected, it may be predictable through a stochastic analysis [8].

By analyzing its sources, faults can be classified as sensor, actuator or component faults. This preliminary study focuses on actuator and components faults, leaving sensor faults to be identified and recovered in parallel by any of the currently available methods that have been developed exclusively to deal with this kind of faults, such as sensor fusion [3] and specialized filters [9].

2.1.1. Basic definitions

Fault Detection (FD) determines if a fault is present in the system and the time of occurrence with appreciable significance. Once a fault is detected, the next step is to perform fault isolation. It is the goal of fault isolation to establish the type or location of the fault. If a detailed model of the plant is available, fault isolation may point out in which component, actuator or sensor the detected fault was originated. The combination of both leads to a Fault Detection and Isolation (FDI) scheme.

The next step, performed by fault identification, ascertains the evolution in size and time of the fault [10]. The information generated by fault isolation and identification is referred to as fault diagnosis. If all three concepts are applied, Fault Detection and Diagnosis (FDD) is achieved [11]. Figure 2.1 illustrates the classification of the different concepts regarding the fault information extracted.

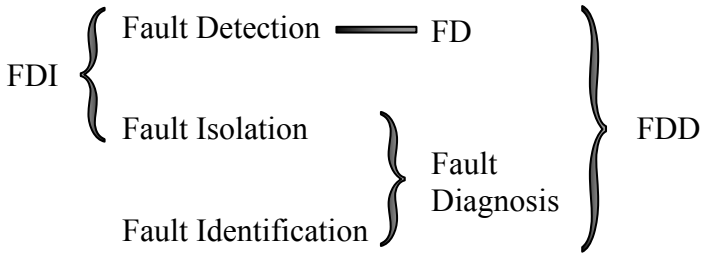


Figure 2.1. Terminology diagram of fault information extraction.

2.1.2. Available methods

The definition that a fault will alter the dynamic behavior of a system lead to model based approaches to implement FD [12]. Following this concept, banks of Kalman filters, Leunberger observers and parity space approaches [13] are tools used when models of the plant in the nominal conditions and under the faults of interest are known and well defined. The amount of prior information required can be relaxed by the use of system identification and parameter estimation to perform on-line adaptation. In all three approaches, residuals are generated by comparing state or output variables of the plant with those derived from the models. Signal analysis can also be used to generate fault indication by means of band pass filters or spectral analysis.

Whether residuals or frequency magnitudes are being monitored, thresholds must be set to distinguish the effects of faults from those of signal noise and model uncertainties. If set too low, the thresholds will increase the chance of false alarms, while setting it too high decreases sensitivity, resulting in a greater misdetection rate. Although usually adjusted off-line, thresholds can also be adapted on-line though statistical analysis of the signals.

The fault isolation problem can be regarded as a classification problem, in which the goal is to correctly identify the source of a fault based on the set of available

residuals. The residuals are in this way viewed as signatures of faults. Detailed system information in the form of a component level model and extensive data collected from fault occurrences are two possible sources of symptom/fault pairs. This data can then be used to design fault isolation schemes based on geometrical distance or probabilistic distributions, or to train an artificial neural network, with the goal of classifying symptoms into different fault scenarios [14]. If human specialists already possess some fault isolation information through experience, fuzzy logic also becomes a useful tool.

In order to generate information according the dynamic evolution of an on-going fault, and in this way reach a complete fault diagnosis, a greater understanding of the system and its faults is mandatory. Diagnostic models, such as symptom-fault causal trees [15], can be used in decision making algorithms in an attempt to gain insight over the behavior and possible outcomes of a fault scenario. Reasoning methods based on probabilistic considerations, fuzzy logic and neural networks are some of the tools available for such a task [7].

A complete fault propagation analysis may generate results that surpass the scope of FDD, helping to identify redundancy requirements early in the designing phase and also leading to a structural analysis that supplies information on which avenues of reconfiguration are still open after a certain fault takes place [3].

2.1.3. Fault information extraction limitations

As methods for FD to FDD and beyond are applied to a plant, the amount of fault information collected grows at the expenses of a greater need of knowledge of the plant and its faults. Although it is possible to perform FDI without a detailed analytical model

of the plant under nominal and faulty conditions, the quality of the information tends to be poorer than otherwise.

On the other hand, methods that rely exclusively on models defined at the designing phase cannot cope with the unpredictable faults that are bound to happen in real world applications. Complex systems pose a problem to this approach even when only the characterization of known faults is considered due to the high level of precision required in mathematical models involving a large number of variables and system parameters. In practice, fault information extraction approaches based solely on fixed models are destined to fail due to their inability to cope with unexpected or uncertain parameter(s) [12].

One alternative way to extract knowledge over the system and its faults is to derive it in terms of facts and rules from the description of their structure and behavior. FDD can then be achieved without the need of precise analytical models, though the information fed to the FDD scheme may be incomplete and uncertain [16].

It is also important to note that most of the fault diagnosis methods are built on top of fault detection modules. Therefore, if a fault is misdetected, it holds no chance of being diagnosed. False alarms may also reduce the effectiveness of the diagnostic procedure, especially in algorithms designed with learning capabilities.

As commented before, the adjustment of FD thresholds directly affects the detection rate and quality. If a FTC that does not make use of fault information (e.g., robust control) is applied over a plant, additional complications may arise. Since FTC's goal is to minimize the effects of faults, it becomes possible that the residuals and the alterations on the signal spectrum will also be reduced, increasing the rate of

misdetections [17]. Although the FTC algorithm correctly addresses to the misdetected fault, the fact that a fault occurred may reduce the redundancy of the system. Without the knowledge of its occurrence, the necessary maintenance intervention may not occur before repeated faults deplete the available redundancy leaving the FTC scheme with no avenues of reconfiguration.

2.2. Fault Tolerant Control

Having established the relevant notions of fault information extraction, the literature review now focuses on the actual Fault Tolerant Control. Following an introduction to the fundamentals of FTC, passive and active approaches are presented and contrasted. Due to some properties of particular interest to this research main goal, further attention is given to the available methods of active FTC with special focus on multiple model architectures.

2.2.1. Introduction to FTC

Failure prevention is not a new concept in theory or practice of engineering. The components or machinery that form a system are often built with safety protections such as fuses or limit switches. Continued operation or start-up is prevented if sensors like those inform that conditions are met to enter a local shutdown mode. This local safety approach though, does not guarantee global fail-safe operation for the complete system. A ship propulsion system depicts an example where the application of local safety with

the lack of analysis of the global implications resulted in many events where consequences vary from irregularity to major economic loss and casualties [3].

Another failure prevention approach derives from the use of direct hardware redundancy. If three or more independent sensors are used to directly measure the same variable, a majority voting can be used not only to detect a fault, but also to isolate the faulty sensor. When only two redundant sensors are available, isolation is not necessarily achievable, but fault detection is still guaranteed. The remedial action to be taken is then simply ignoring the isolated sensor or generating an alarm when no trustable signal is available.

The same principle is applied to components and actuators, though it is possible in those cases that more than one output from different elements operating at only a fraction of its total capability is used at the same time. After a fault is isolated in one of the elements, the failure prevention approach then becomes one of energy redistribution among the healthy set.

Fault Tolerant Control's goal is to prevent failures at system level through proper actions in the programmable parts of a control loop. In this approach, analytical redundancy can be used in place of its hardware counterpart. Analytical redundancy helps not only to reduce the cost involved in using extra elements, but also delivers greater design freedom to avoid the loss of performance that may result from direct hardware redundancy implementation. When sensors are considered, the use of analytical relations united with the actual measurements also increases the degree of confidence of the considered variable. Since FTC focus on the overall mission goal and aims for continuous system availability, different from the other failure prevention approaches mentioned

earlier, a loss of performance is allowed after a fault occurs. As a matter of fact, given the specific redundancies available in a given system, a reconfiguration to a state of inferior performance might be an optimal solution when the mission objective, such as stability, is preferred.

2.2.2. Passive versus active approaches

One possible way to implement fault tolerance is to design static control laws capable to compensate for some plant uncertainties such as disturbances and noise [18]. If the effects of a fault are small enough to be in the range covered by the robustness of the controller, no specific reconfiguration is required. Since no information about the faults is typically utilized by the control system, this type approach is often referred to as "passive fault tolerant control".

By utilizing fault information extracted from the system, it becomes possible to design a reconfigurable controller that modifies the control function (parameters or structure) in response to faults, characterizing an "active fault tolerant control". This approach is preferable over the passive one when tolerance to a wider range of faults is intended since the required increase in robustness has a negative effect on the performance, even under nominal operation. As depicted in the generic active FTC diagram in Figure 2.2, it is common to separate the control algorithm into two distinct blocks: a baseline controller and a supervisor system. While the baseline controller focuses on the maintenance of the immediate control objectives, the supervisor extracts fault information, determines remedial action and executes them by modifying the baseline controller [19].

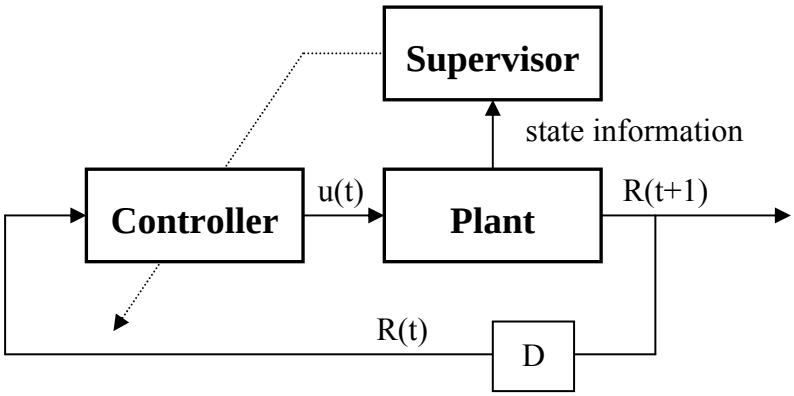


Figure 2.2. A generic active fault tolerant architecture depicting the base line controller and the supervisory system. In the diagram: D represents a delay block, u(t) is the controlled input and R(t) is output of the plant.

2.2.3. Active FTC methods

Active FTC systems compensate for the effects of a fault either by selecting a new precomputed control law (projection-based methods) or by synthesizing a new control law on-line (on-line automatic controller redesign methods) [20].

Gain scheduling [21], fuzzy decision logic [22] and structural analysis [3] are some of the possible ways to implement projection-based active FTC. Models and pre-computed controllers for the system under nominal conditions and under the effect of the faults of interest are used during the design phase to grant the controller quick and correct responses to the envisioned scenarios. However, since fault information at least to the level of isolation is essential, it is necessary for the models of the faulty scenarios to be accurate enough to be distinguishable under the effect of noise and disturbances. Even when precision is not taken into account, the mere task of off-line design of characteristic models for fault scenarios with a strong stochastic nature is by itself a challenging one, especially if complex nonlinear plants are considered.

On-line automatic redesign methods are of particular interest in light of the goal of the proposed work due to its capability of providing specific control actions even to fault scenarios that had not been necessarily anticipated during the design phase. Reconfigurable control can be used to implement on-line redesign requiring only the residuals generated by fault detection. Nevertheless, the flexibility gained by this approach comes at the expense of slower response since the controller must be allowed time to learn the new dynamics and modify itself. Since the reconfigurable controller does not require knowledge of the dynamics of the system under the effect of each specific fault, it is inherently immune to modeling errors and possesses a greater potential to deal with unmeasured disturbances and noise-corrupted data.

A reconfiguration approach in which the eigenstructure can be directly assigned to the close-loop system to achieve the desired system stability and dynamic performance is known as Eigenstructure Assignment (EA) [23]. The conditions for exact assignment are the existence of a sufficient number of actuators and measurements available and that the desired eigenvectors reside in the achievable subspaces. The limitations of EA are that the system performance may not be optimal in any sense, and that the system requirements are often not easily specified in terms of the eigenstructure [24].

The Pseudo-Inverse Method (PIM), on the other hand, is a reconfiguration method that is optimal in the sense that it minimizes the Frobenius norm of the difference matrix between the original and the impaired closed-loop system transition matrices. Since in its initial formulation stability cannot be guaranteed, a Modified Pseudo-Inverse Method (MPIM) was proposed [25]. In its initial formulation however, MPIM required full state feedback and relied on stability bounds that could give very conservative

results. Those limitations were the focus of [26], where the problem was reevaluated from an optimization point of view while focusing on FTC application. Although the state feedback constraint was relaxed to output feedback, the method still requires residuals for each parameter of the model to be generated (comparison between transition matrices), limiting in the reconfigurable fault scenarios to those with the same dynamic structure than the nominal mode.

However, both EA and PIM-based controllers are restricted to implementation on linear models. When a dynamical nonlinear structure is given and only the parameters are unknown, adaptive control can be used. Even to this restricted case, the assumptions that have to be made concerning the unknown plant to develop a stable adaptive controller were established only in the 1980's [27]. The problem becomes truly formidable when the plant is nonlinear and the input-output characteristics are unknown and time-varying.

From a system theoretic point of view, artificial Neural Networks (NN) can be considered as practically implementable parametrizations of nonlinear maps from one finite dimension space to another. Theoretical works by several researchers have proven that, even with one hidden layer, neural networks can uniformly approximate at any degree of precision any piecewise continuous function over a compact domain, provided the network has a sufficient number of units, or neurons. Therefore, NN can, by their very nature, cope with complexity, uncertainty and nonlinearity, and NN have been used successfully to identify and control nonlinear dynamic systems [28].

Multilayer Neural Networks (MNN) and Radial Basis Functions Networks (RBFN) have proven extremely successful in pattern recognition problems, while Recurrent Neural Networks (RNN) have been used in associative memories as well as for

the solution of optimization problems [29]. From the theoretic point of view MNN and RBFN represent static nonlinear maps while RNN are represented by nonlinear dynamic feedback systems [30].

In [31] a Recurrent High Order Neural Network (RHONN) was developed with the goal of identification of dynamical systems displaying similar convergence properties of classical adaptive and robust adaptive schemes. A Lyapunov-based approach is used to prove the convergence property of the learning algorithm that ensures that the identification error converges to zero exponentially and that, if it is initially zero, it remains in zero during the whole identification process. Later in [30] the identifications capabilities of the RHONN were used to provide state information to a sliding mode controller to solve a tracking problem. However, the RHONN displays serious restrictions to its applicability to complex systems due to a lack of scalability in its heavily connected architecture.

In [32] a simplified RNN is used to identify the system and its parameters used as input to a controller based on feedback linearization and pole placement. Stability though, is only assured if the controller system remains stable, a limitation that greatly decreases the applicability of the method to the FTC problem.

A RNN based adaptive controller specially developed to deal with nonlinear systems with unknown dynamics is presented in [33]. In the proposed configuration, the output from the RNN adaptive controller was applied to the system summed with the output of a linearizing controller designed off-line to deal with the nonlinearities in the nominal model. The proposed learning algorithm was stable in the Lyapunov sense, but

the restrictions applied to achieve such proof make this approach capable only to deal with incipient faults.

In order to achieve semi-global boundedness of all signals in a control loop of a MIMO system, a backstepping approach is used in [34] to divide the MIMO nonlinear model into a series of SISO nonlinear models and design controllers separately using RBFN's. However, in order to achieve such degree of decoupability, it must be possible to describe the system in block-triangular form. Even if true for the nominal model, a fault may increase relationships between states that could previously be ignored, making it impossible for the system to fit in a block-triangular form again.

Taking inspiration in a PID controller, a modified RNN architecture is applied in a model reference adaptive control framework to control an automotive engine in [35]. Although identification and control are performed by RNNs, the identification is performed off-line while only the controller is trained on-line. Therefore direct application of this method to systems which dynamics may be affected by faults in unexpected ways is not possible.

2.3.4. Multiple models as a framework for active FTC

Even though a reconfigurable adaptive controller is a key element without which solutions for unknown faults cannot be designed online, if used as a FTC architecture alone, it displays two major limitations. The first involves the fact that a reconfigurable controller makes it impossible for any available fault knowledge to be incorporated during design time. Although an ideal reconfigurable controller will always reach a solution (given its existence) for a given fault scenario, the amount of time it must be

allowed to learn the new dynamics and modify itself accordingly could be greatly reduced by the direct application of a known solution. The second major limitation is caused by the known tradeoff between adaptability and long-term memory. As a reconfigurable controller is optimized to deal with a broader scope of faults with minimum reconfiguration time, previously configured controllers are forgotten and the reconfiguration process has to be repeated even when returning to the healthy condition from an intermittent fault scenario.

Multiple Models Architecture (MMA) [18,22,28] presents a framework in which projection-based methods and online redesign can be synergistically integrated to provide the fast and specific response of the first combined with the flexibility and robustness of the second. More specifically, in [36] and [37] it was shown that implementing a reconfigurable controller in a MMA has the potential to overcome the cited limitations for the tracking of complex nonlinear plants. Since then, MMA has been applied to FTC by combining fault scenarios and their respective control solutions in model banks coordinated by a supervisor. However, most publications so far are based on fixed model banks built offline and therefore are incapable of improving the controller response in the reoccurrence of faults that were unexpected during design time. In [38] a Dynamic Model Bank (DMB) is used to allow the insertion of new plant dynamics as they were identified online, but the use of a linear controller and the lack of a complete Fault Detection and Diagnosis (FDD) scheme significantly limit its applicability.

To better understand the MMA approach, its simplest implementation, Gain Scheduling (GS), will first be introduced and discussed. GS is a technique that aims to provide control over nonlinear systems without requiring the design of nonlinear

controllers. The first step in GS is to linearize the model about one or more operating points. Then linear design methods are applied to the linearized model at each operating point in order to arrive at a set of linear feedback control laws that perform satisfactorily when the close-loop system is operating near the respective operating points. The zone of the state space where a controller still performs satisfactory is denoted operating region. The final step is the actual gain scheduling, which is intended to handle the nonlinear aspects of the design problem. The basic idea involves interpolating in some way the linear control law designs at intermediate operating conditions. It is usual in GS applications to choose a particular structure for the linear controllers (e.g., PID) and therefore its parameters (gains) are modified (scheduled) according to the states of the closed-loop system.

In addition to the evident simplicity brought by the design of the controllers for linear approximations instead of the global non-linear models, GS also provides the potential to respond rapidly to changing operating conditions and its real-time computational burden is light [21]. However, since the design process of GS in its original formulation is based only in local information of a limited set of operation points, no global characteristic (stability, performance, robustness, etc…) can be guaranteed. In the same way that a well-designed set of linear controllers does not necessarily result in even a globally stable control law for the nonlinear system, reachable nonlinear systems may provide uncontrollable linearized models, preventing GS to be applied at all.

Advanced MMAs make use of local nonlinear models to design its controllers, resulting into the operating region in which each controller remains valid potentially

bigger. Given enough information of the system, this property allows the main components of a system dynamics to be represented in a finite set of nonlinear models, making it possible to incorporate global stability, performance and robustness requirements in the design phase of multiple models. Model predictive control, feedback linearization and sliding mode [39] are examples of such methods. Another benefit from the use of nonlinear models and controllers is that it provides the possibility to dramatically reduce the total number of models, making it feasible to apply the MMA concept to systems with widely diversified complex dynamics.

Nevertheless, independent from the linearity of the models used to generate the set of controllers, the quality of the end result of the application of a MMA approach is still largely affected by a wide range of design choices concerning how many to create, where to position and how to interpolate the controllers designed at each operating point or region.

For a better understanding and comparison between different approaches, the parameter space representation presented in [36] will be used. The parameter space (S) is an augmented version of the state space representation that includes "states" of the environment that contain information of sensors present in the plant used solely to extract fault information. Temperature, for example, can be considered an environmental state if the model of the plant does not take it into account directly, but as the temperature deviates from the nominal condition the dynamics of the plant are altered. In the examples that follow, the parametric space is a bounded region that encompasses the physically achievable values of each state.

For the sake of visualization, the following discussion will be held with examples using two dimensional parameter spaces. The conclusions however are not limited to this particular case, being possible to apply all the discussed methods in higher dimensional spaces directly.

Figure 2.3 brings a basic MMA setting where a set of controllers is devised for some specific operating regions sparsely distributed on the parameter space. Each of the operating regions (O_1, O_2 and O_3) is generated around an operating point and limited by the range of the state space of the plant in which the corresponding controller performs with a minimum performance. In the dimensions of the parameter space that do not represent states of the plant, the operating regions represent the robustness of the controller.

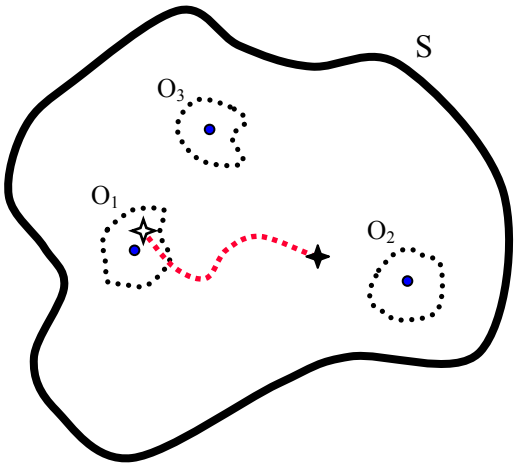


Figure 2.3. Performing Multiple Model control with sparsely distributed operating regions. O_1 to O_3 are operating regions around each operating point. The system is originally in the position of the parameter spaced marked by the white star and follows the depicted trajectory.

If the plant is in a position in the parameter space that is close enough to an operating point to be inside its operating region, it is reasonable to apply the respective pre-computed control law. This is the case of the original position (white star) of the

trajectory shown in Figure 2.3. However, variations in the set-point or the occurrence of faults may take the system to a point away from all operating points that were considered off-line (black star) and the question of what control law to use is raised. As a matter of fact, since precise description of the operating regions is not often available in practice, such question may arise even while the plant is still inside the true operating region.

Perhaps the most intuitive approach, one of the ways to generate control laws for in-between operating points is to assign a mean of the parameters of the controllers at each operating point to the parameters of the active controller, weighting it by their geometrical distance with respect to the present position in the parameter space. The main critic to this method is that it does not take into account the nonlinear characteristic of the system that creates a heterogeneous parameter space. In Figure 2.3 for example, since the plant finds itself closer to O_2, weighting the sum by the geometrical distance alone would result in a control law more similar to the one devised for that operating point. However, if a strong nonlinearity existed between O_2 and the present system position, the ideal control law may be more similar to those created for O_1 and O_3.

Among the techniques that have been researched aiming to overcome this limitation, some are of special interest to this study as they were specifically designed for FTC applications. In [22] a set of IF-THEN rules was used in a fuzzy logic framework to compare the present position in the parameter space with the symptoms of known faults. The degree of similarity with each fault scenario was then used to weight the mean that adjusts the parameters of the controller. Assuming that knowledge is available regarding the status of the system that make it prone to develop each expected fault, in [18] this

approach was improved by applying the fuzzy algorithm only to the set of possible faults at a given position in the parameter space.

A different approach was taken in [24] where the probability of occurrence of each expected fault was modeled in a finite-state Markov chain with known transition probabilities. With this information at hand, the mean of control parameters was weighted favorably to the most probable fault scenarios.

Regardless of the weighting scheme chosen, it is still an approximation of the behavior of the system outside the considered regions of operation and as such it is inevitably susceptible to nonlinearities active outside those regions. One way to solve this deficiency is to generate closely connected models by dividing the whole parameter space into even operating regions as shown in Figure 2.4(a). The natural tradeoff of this method is that increased control performance tends to require controllers designed for smaller, and therefore less complex, regions. This in turn causes the final number of models needed to cover the whole space to grow, requiring extensive design work since a control law has to be designed for each model. This relationship can be clearly seen in a series of simulation results performed in [37]. If made small enough, each region can be represented by a linear model given by the linearization of the nonlinear plant on the center of the operating region, making it possible to apply GS.

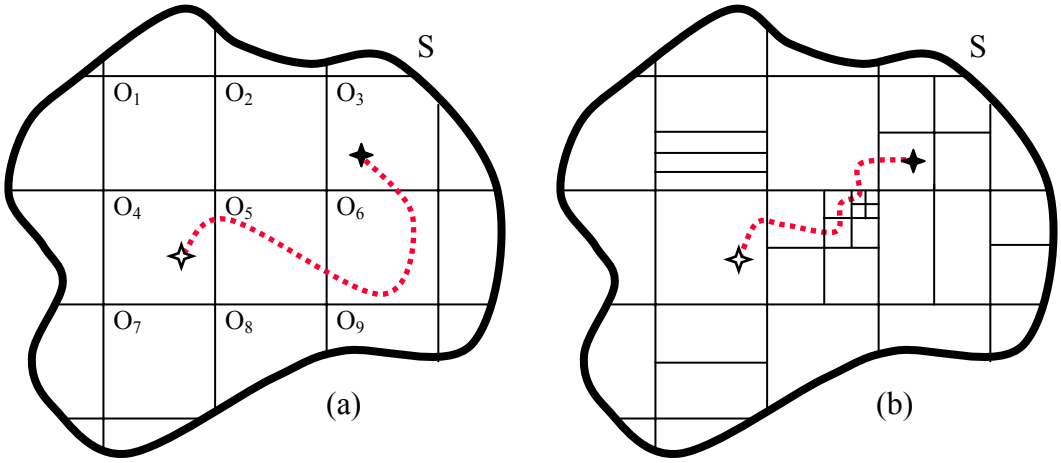


Figure 2.4. Closely connected multiple model implementations: fixed size operating regions (a) and plant dynamics dependant operating regions (b). In the diagrams each rectangular section represents an operating region. The system is originally in the position of the parameter spaced marked by the white star and follows the depicted trajectories.

Since all parameter space is covered by previously designed controllers, a basic closely connected MMA algorithm would be composed only of two steps: determine the present position of the plant in the parameter space and apply the corresponding controller. However, even though each controller is designed to result in a desirable behavior for the plant while inside its respective operating region, if no special procedure is performed, switching directly from one control law to another may cause all kinds of unwanted responses as the plant navigates from one operating region to another. In [36], a minimum time (or number of iterations) was set for permanence inside an operating region before switching takes place, creating in this way a time based hysteresis in an effort to prevent oscillations between adjacent operating regions. Another approach, requiring all controllers to possess the same structure, is to create an area on the border of adjacent operating regions in which the parameters of both controllers are combined causing one to gradually change to another. However, both methods are solely heuristic solutions and no proof of their efficiency, let alone deterministic way to configure their

design parameters, is available. A method that guarantees stability of systems when perform control switching has been presented in [40]. The referenced paper describes a way to compute a pre-transition sub-region inside an operating region from which stability is assured when switching to another specific operating regions. In [37], an adaptive controller that operates in parallel with the MMA is used to assure stability of the system during the transient behavior generated by switching controllers.

If complete information on the dynamics of the nonlinear system is available beforehand, it is possible to divide the parameter space taking into account the sensibility of different areas (as shown in Figure 2.4(b)) and produce a combination of controllers with good performance based on a compact set of operating regions. It is important to notice that, independent of the number and uniformity of the regions, because no interpolation is fundamentally necessary, different control structures or even strategies can be used for each region.

From the point of view of FTC applications that consider the occurrence of unexpected faults, model weighting is not an attractive technique since there is no reason to assume that a new fault dynamic will hold any relationship with those previously known. When closely connected multiple models are considered, the quality of the response depends on the robustness of the design of each controller and the way the control laws are switched from one to another. Although in this formulation an active FTC is being performed for expected faults, no direct action can be defined for unexpected dynamics. At the same time that the requirement for robustness increases, since the areas of sensitivity are no longer available at design time, a large number of

evenly spaced operating regions have to be created making the memory requirement and design effort increase greatly.

It is therefore interesting to explore yet another way to apply the MMA concept in which controllers are designed on-line as new operating regions are reached [37]. Since no information about the parameter space is supposed to be available at design time, nonlinear online identification is required in order to learn new operating regions (models) and recognize the ones to which a controller has already been designed. In this way, different from the previously discussed methods that adjust the controller based on the position of the plant in the parameter space, the on-line building of models achieves the same in an indirect manner by the identification error of the models designed so far. Therefore, if at a given moment the identification error of every known model (contained in a dynamic database) is high the plant is considered to be in an unknown region of the parameter space, while if the error of one of the models is low it indicates that the plant is inside the respective operating region. What is considered to be "high" or "low" depends on an identification threshold selected by the user. By reducing this threshold the operating region of each model shrinks, causing a greater number of models to be generated. In this sense, a parallel can be traced between the setting of the identification threshold and the choice of how many fixed models to have in the closely connected operating regions approach.

It is interesting to notice that, due to the indirect measuring through the dynamics of the plant, the operating regions now span in the space of the identifier, not in the parameter space. If a neural network is used to approximate the plant dynamics for example, the operating regions span in the dimension defined by its weights. Operating

regions described in the parameter space possesses all its dimensions with direct physical meaning since they came from sensor readings. Although this property can be highly desirable in certain operations such as translating expert knowledge to the model database, changes in the dynamics are not always directly linked to the position on the environmental space. For example, a high temperature in a certain part of a system may not instantly incur in a fault, but may increase the probability of its occurrence. Since the identifier focus on the change in the dynamics and not on the secondary symptoms of faults, it does not suffer from such drawback. On the other hand, in order to extract information from expert sources it is necessary to duplicate the described conditions in simulation so that the identifier is able to produce a model in its own space.

As with the identification models, the control laws must also be devised on-line. A single control strategy that modifies itself based on the identified models, such as approximate feedback linearization [41], is a valid approach for plants which dynamics do not present extreme non-linearities. When it is not the case, highly flexible nonlinear adaptive controllers [28] may be applicable.

If a new model is added to the database every time the identification threshold is exceeded, the area of the parameter space to which the system is exposed will be filled with closely connected models and therefore there is no need to use the same control structure for every operating region. Particular solutions previously known to exist to particular regions of the parameter space can then be directly introduced. For example, fuzzy logic can be used to extract expert knowledge on the solution of a particular fault, while neural networks is used to generate novel control laws to cope with unexpected fault scenarios.

Sparsely connected model distribution can also be attained by the on-line MMA approach if a second threshold to measure model dissimilarity is created. The dissimilarity threshold, always greater than the identification one, indicates the regions in which the present dynamics of the plant are considered to be different enough from all the models in the database to justify the addition of a new model. Such scheme was implemented in [38] where the parameters of the controllers for regions not covered by the models in the database were adjusted by a mean of the known controllers weighted by the inverse of the identification error of their respective models. In this way, the control laws for regions between models hold more similarity to the ones devised for similar plant dynamics.

Apart from the above mentioned concerns involving the transient behavior of the system when switching is performed, the application of MMA to FTC harbors two other points that require careful consideration. The first of them is the fact that the task to link either the present location on the parameter space or the prediction errors of identification models to the occurrence of a particular fault represents a FDI process and as such is vulnerable in all issues outlined in Section 2.1. The second point focus on FTC applications that require new models to be designed on-line in a continuously growing database. In such a scenario the nonlinear adaptive controller is required to be at the same time: quick to converge, highly flexible and possess guaranteed stability, often conflicting characteristics in practice.

CHAPTER 3 – Baseline Nonlinear Adaptive Controller

3.1. Introduction

Increased performance specifications are often achieved at the cost of amplified plant and control complexity. As overall complexity rises, so does the chance of occurrence, diversity and severity of faults. When complex systems suffer from faults, the original model parameters or even its own dynamic structure may change in a multitude of unpredictable ways. Even if the system has a satisfactory linearization around the nominal operation point, nonlinearities may become of paramount importance after a fault occurs [4]. When the stochastic nature of faults is taken into consideration and to predict all fault scenarios is made impossible, it becomes clear that the problem of interest of FTC cannot be dealt with without an on-line nonlinear adaptive control strategy.

It is important to state here that, for the benefit of the discussion in this chapter, the required redundancy is assumed to exist in the system. Hardware redundancy requires two or more independent instruments that perform the same function, while analytical redundancy uses two components based on different principles to measure a variable, where at least one of them uses a mathematical model in analytical form. In either case, from the theoretical point of view, this assumption matches the requirement for sustained

observability and controllability (or global reachability for nonlinear systems) through fault scenarios. This condition is later relaxed in Chapter 6.

In this section, we explore control architectures that take advantage of the universal approximation capability of the nonlinear maps generated by Neural Networks (NN). In particular, five different NN control architectures are presented in order of growing sophistication, starting from two basic NN architectures and leading to three adaptive critic approaches. The merits of each are discussed and their shortcomings exposed, which in turn becomes the motivation for the next. The first architecture is an application of a single NN with a classical training algorithm and the requirement of full knowledge of the plant's dynamics at all times. The controller is then improved by the addition of a second NN capable of generating online a map of the plant's dynamics, however the training algorithm remains fundamentally the same.

The addition of a third NN and a change in the training paradigm leads to the adaptive critic architectures: Heuristic Dynamic Programming (HDP), followed by Dual Heuristic Programming (DHP) and finally Globalized Dual Heuristic Programming (GDHP). Made clear through their description, adaptive critic architectures are preferred for FTC implementations due to its great flexibility and known effectiveness to work in noisy, nonlinear environments while making minimal assumptions regarding the nature of such environment [7].

3.2. A single NN control architecture

The goal of this approach is to use a NN to generate a nonlinear map connecting the states of the plant $x(t)$, previous inputs $u(t-1)$ and current target $x^t(t)$ to an input $u(t)$ that will minimize the utility function $U(t)$ defined by (3.1).

$$U(t) = \frac{1}{2}\left(x(t) - x^t(t)\right)^T Q\left(x(t) - x^t(t)\right) + \frac{1}{2}\rho u(t)^T R\, u(t) \tag{3.1}$$

where, Q is a diagonal square matrix that can be used to assign different degrees of importance to each state, R is the equivalent matrix that penalizes the amount of control action used and ρ is a scalar used to balance the minimization of the tracking error and the energy use during the process.

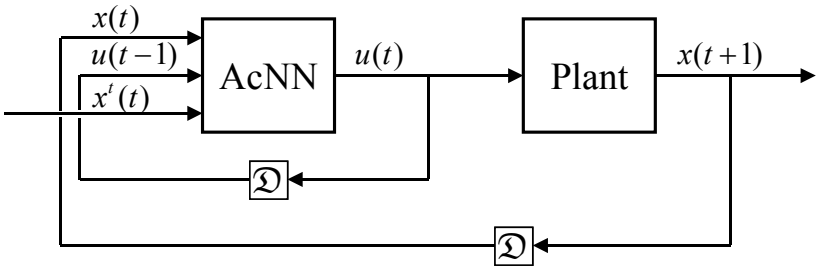


Figure 3.1. A single NN control architecture

In order to differentiate it from the other NNs that will be introduced in later architectures, this NN is named Action Neural Network (AcNN). Figure 3.1 depicts such architecture. When performing the training of the AcNN, the information of how its weights affect the states of the plant is required. However, backpropagation through the AcNN only provides information on how the inputs $u(t)$ are affected by its weights. Therefore, this approach requires the availability of a differential model of the dynamics

of the plant from which the information on how the states $x(t)$ are affected by the inputs $u(t)$ can be extracted. As a result, this architecture is not suitable for FTC application, since faults are assumed to modify the dynamics of the plant in unpredictable ways, making it impossible to design models beforehand.

3.3. A direct adaptive control architecture using two NNs

Since it is not possible to offline design models of the plant dynamics for all fault scenarios, in this architecture a second neural network is introduced with the goal of performing online plant identification. Once this network has converged to represent a map of the dynamics of the plant, the derivative of the states with respect to the inputs can be extracted through standard backpropagation. Such network will be referred to as the Identification Neural Network (IdNN). Figure 3.2 displays this second approach.

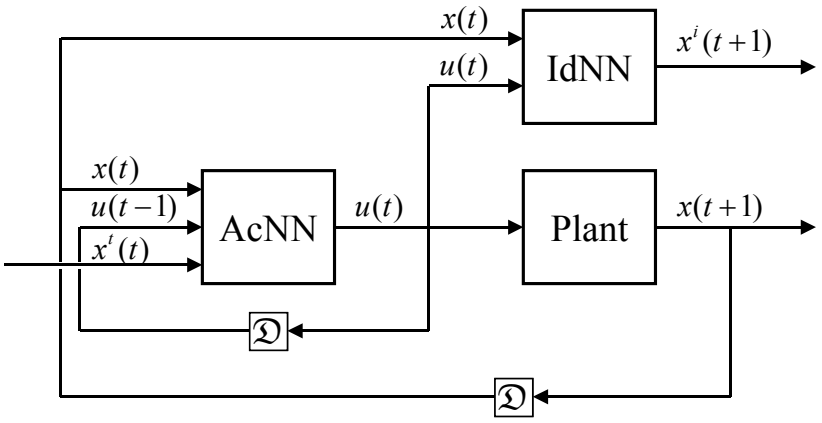


Figure 3.2. A direct adaptive control architecture using two NNs

Although no critical restrictions prevent this architecture to be used as a solution to the FTC problem, its performance can still be largely improved if the training

algorithm for the AcNN is reevaluated. In these first two architectures, the AcNN is trained at each iteration with the goal of reducing the current value of the utility function $U(t)$. This is performed under the assumption that this process will ultimately lead to a set of weights that minimize the utility function for all times. However, this training approach provides no mechanisms to minimize the values that $U(t)$ assumes *during* training (or the time it takes). Clearly it is of the interest of FTC to provide a new control solution to a fault scenario as quick as possible and with minimum performance impact.

3.4. Heuristic Dynamic Programming

Seeking to overcome the limitations of the previous approaches, the first adaptive critic controller is introduced. Adaptive critic architectures have a much greater potential to achieve the required degrees of reconfiguration and stability because more than the simple instantaneous difference between desired and actual states is available to be used as performance index. Due to the continuous interaction between the controller and the plant, the quality of a certain control strategy can only be fully measured after analyzing all future effects it has on the control mission, in our case trajectory tracking.

Therefore, HDP trains the AcNN to minimize not the present utility function alone, but also the sum of all future values of $U(t)$ with a decaying factor γ $(0<\gamma<1)$. Such quantity is referred to as the cost-to-go $J(t)$, as defined by the Hamilton-Jacobi-Bellman equation (3.2), and represents the core of dynamic programming [42].

$$J(t)=\sum_{k=0}^{\infty}\gamma^{k}U(t+k) \tag{3.2}$$

Problems formulated in this form are the main focus of dynamic programming, that solves it through a backward search from the final step [43]. To make the problem tractable to an on-line learning approach, adaptive critic designs require an estimate of the actual cost-to-go to be constantly determined. Although ACD's can be implemented with any differentiable structure [44], neural networks have been widely used [45] due to their generalization and nonlinear mapping capabilities as well as having suitable methods for on-line learning. Given the complexity of FTC systems, dynamic or recurrent neural networks were chosen due to their more efficient handling of dynamic nonlinear mapping [30]. It is in this context that we introduce a third NN, denominated the Critic Neural Network (CrNN), responsible for approximating $J(t)$. The resulting block diagram is shown in Figure 3.3.

In other words, in adaptive critic designs, the training of the AcNN is done in the direction of the minimization of the cost-to-go approximation. In HDP this is accomplished by starting the training path of the AcNN with the information of how the inputs and states will affect the current cost-to-go $J(t)$. Since the CrNN is trained to estimate it, such information can be easily extracted from the NN via backpropagation though time [46,47].

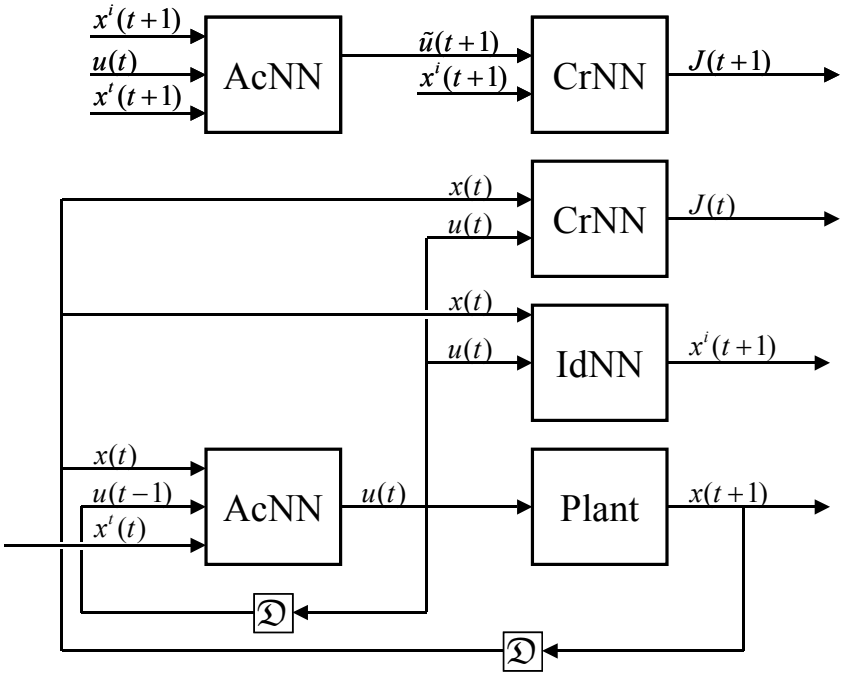


Figure 3.3. Heuristic Dynamic Programming

3.5. Dual Heuristic Programming

Dual Heuristic Programming reevaluates the purpose of the CrNN and redesigns it. Although in HDP the CrNN is trained to estimate $J(t)$, its true purpose is to provide the AcNN with the partial derivatives of $J(t)$ with respect to the states and inputs (usually referred to as $\lambda^x(t)$ and $\lambda^u(t)$ respectively). In DHP architecture, as shown in Figure 3.4, the CrNN is trained to output such derivatives directly. Using this direct approach, DHP is capable of generating smoother derivatives and has shown improved performance when compared to HDP. Those results were presented in [44], where both methods were applied to a turbogenerator, characterized as a highly complex, nonlinear, fast-acting, multivariable system with dynamic characteristics that vary as operating

conditions change. These benefits, however, come with the tradeoff of a more complex training algorithm for the CrNN as shown in [48].

3.6. Globalized Dual Heuristic Programming

3.6.1. Introduction

The adaptive critic GDHP algorithm combines the HDP and DHP approaches to generate the most complete and powerful adaptive critic design [7]. In GDHP, λ^x and λ^u are determined with the precision and smoothness of DHP, while improving the CrNN training by also estimating $J(t)$ as in HDP. Figure 3.5 depicts the block diagram of this approach.

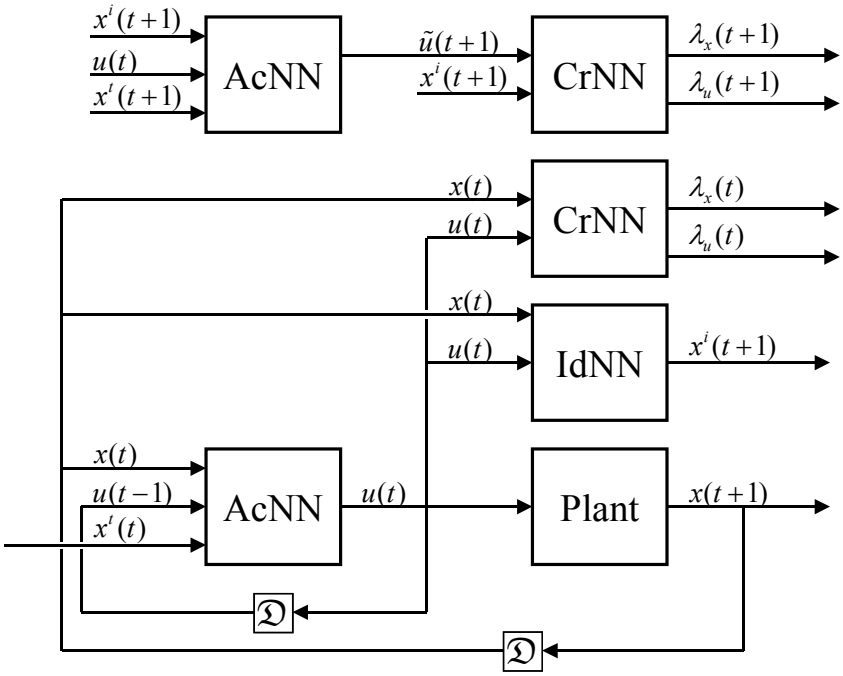


Figure 3.4. Dual Heuristic Programming

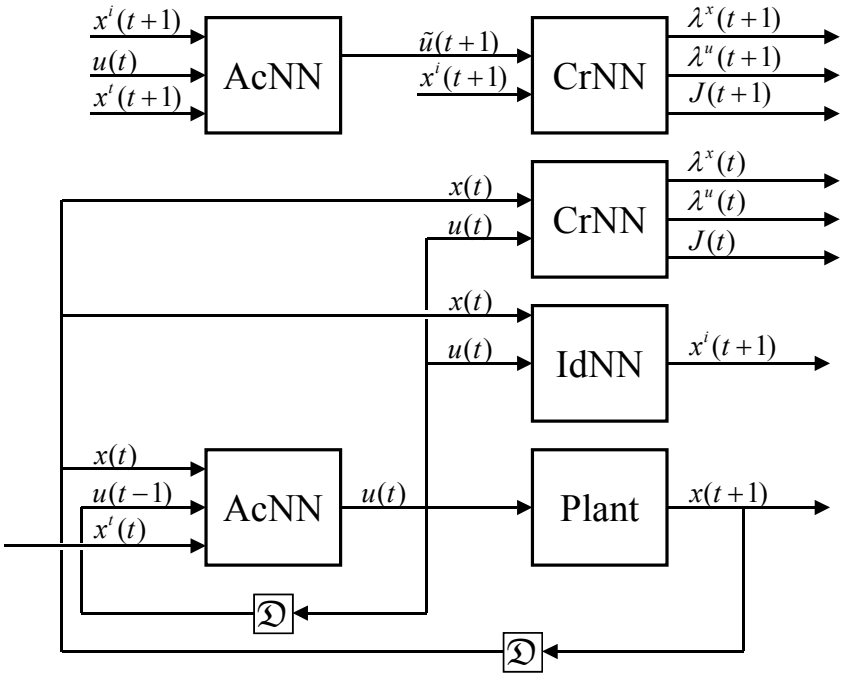


Figure 3.5. Globalized Dual Heuristic Programming

In this section, the adaptive critic architecture of Globalized Dual Heuristic Programming is presented in detail. Following this introduction, the adaptive control problem of interest to FTC is stated mathematically and the adopted notation introduced. The next three subsections are focused each on one of the neural networks that compose the GDHP architecture: identifier, action and critic. Each neural network has its structure presented, followed by a discussion on its training algorithm and the ways through which information required by other networks is extracted. Finally, all information contained in this section is summarized in the complete GDHP algorithm presented in a manner that can be readily applied.

3.6.2. Preliminaries

The first step is to define $x(t)$ (3.3) and $u(t)$ (3.4), column vectors of the $n\,x$ states and $n\,u$ inputs at time t, and the Tap Delay Line (TDL) vectors $\overline{x}(t)$ (3.5) and $\overline{u}(t)$ (3.6) that combine information of TDL_x and TDL_u sampling times respectively.

$$x(t) = \begin{bmatrix} x_1(t) & x_2(t) & \cdots & x_{n\,x}(t) \end{bmatrix}^T \tag{3.3}$$

$$u(t) = \begin{bmatrix} u_1(t) & u_2(t) & \cdots & u_{n\,u}(t) \end{bmatrix}^T \tag{3.4}$$

$$\overline{x}(t) = \begin{bmatrix} x(t)^T & x(t-1)^T & \cdots & x(t-T\,D\,L_x+1)^T \end{bmatrix}^T \tag{3.5}$$

$$\overline{u}(t) = \begin{bmatrix} u(t)^T & u(t-1)^T & \cdots & u(t-T\,D\,L_u+1)^T \end{bmatrix}^T \tag{3.6}$$

Given the causal plant (3.7) with nonlinear $f(.)$ subject to abrupt faults characterized by discontinuous changes in its parameters or structure, the primary goal of the controller (3.8) is to make the states track the desired trajectory $x^t(t)$. Since particular fault scenarios may render regions of the state space unreachable to the plant, the controller is not required to reduce the tracking error to zero, but rather minimize it under the constrains of each particular fault. In the controller, $g(.)$ is a nonlinear continuously differentiable approximator composed of three neural networks: identification, action and critic. The way each neural network is trained online and how they interact in the GDHP architecture is explained in detail in the following sections.

$$x(t) = f\left(\overline{x}(t-1), \overline{u}(t-1)\right) \tag{3.7}$$

$$u(t) = g\left(\overline{x}(t), \overline{u}(t-1), x^t(t)\right) \tag{3.8}$$

3.6.3. Identification Neural Network (IdNN)

The IdNN (shown in Figure 3.6) is responsible for generating a differentiable map that matches the dynamics of the plant. Note that, in the used notation, all variables related specifically to the IdNN receive the superscript *i*. Designed as a two-layered recurrent neural network [49,50] with input $p^i(t)$ (3.9), *nhi* neurons in the hidden layer and a tangent sigmoid transfer function (3.10), the IdNN outputs a vector of the estimated states $x^i(t)$ (3.11).

$$p^i(t) = \begin{bmatrix} \overline{x}(t-1) \\ \overline{u}(t-1) \\ a^i(t-1) \\ 1 \end{bmatrix} \tag{3.9}$$

$$a^i(t) = \mathrm{tansig}\left(W^{i1}(t)p^i(t)\right) \tag{3.10}$$

$$x^i(t) = W^{i2}(t)\begin{bmatrix} a^i(t) \\ 1 \end{bmatrix} \tag{3.11}$$

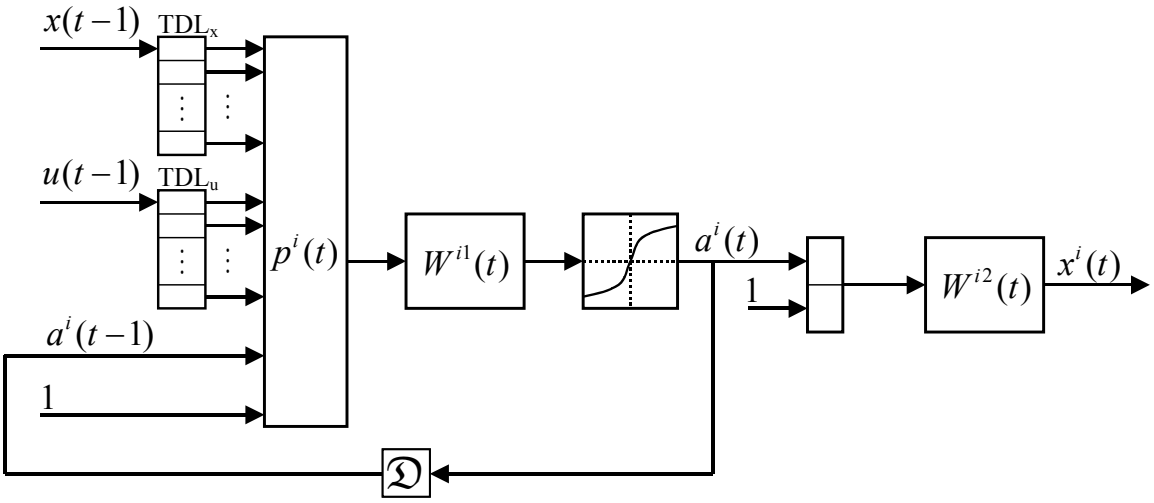


Figure 3.6. IdNN recurrent neural network architecture

The network is trained online with the goal of minimizing the identification error $E^i(t)$ subject to the relative importance matrix S (3.12). Generally the matrix S is set as the identity, however, by adjusting the magnitude of the diagonal elements, the IdNN can be made to focus more on the reduction of the identification error of certain states. By applying the steepest descent training algorithm, the weight update equation (valid for both layers) is given by (3.13).

$$E^i(t) = \frac{1}{2}\left(x^i(t) - x(t)\right)^T S\left(x^i(t) - x(t)\right) \tag{3.12}$$

$$w^i(t+1) = w^i(t) - \beta^i \left(\frac{d\,x^i(t)}{d\,w^i}\right)^T S\left(x^i(t) - x(t)\right) \tag{3.13}$$

where w^i is a column vector of the elements of the corresponding weight matrix W^i, and β^i is the learning rate. Equations (3.14) to (3.16) show how the required derivatives are calculated.

$$\frac{d\,a^i(t)}{d\,w^{i1}} = \left(I - \mathrm{diag}\,(a^i(t))^2\right)\left(\begin{bmatrix} p^i(t)^T & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & p^i(t)^T \end{bmatrix} + W^{i1}_{(:,end-nhi:end-1)}(t)\frac{d\,a^i(t-1)}{d\,w^{i1}}\right) \quad (3.14)$$

$$\frac{d\,x^i(t)}{d\,w^{i1}} = W^{i2}_{(:,1:nhi)}(t)\frac{d\,a^i(t)}{d\,w^{i1}} \quad (3.15)$$

$$\frac{d\,x^i(t)}{d\,w^{i2}} = \begin{bmatrix} \begin{bmatrix} a^i(t)^T & 1 \end{bmatrix} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \begin{bmatrix} a^i(t)^T & 1 \end{bmatrix} \end{bmatrix} \quad (3.16)$$

where w^{i1} corresponds to the weights of the first layer and w^{i2} to those of the second, and the standard MATLAB notation for the indication of rows and columns within a matrix is used. In such notation, when used by itself, the colon indicates all entities in a particular dimension (e.g., all rows or all columns), while when used between two numbers or variables it indicates the range between and containing such values in the corresponding dimension (e.g., all rows from 5 to *nhi*).

In order to train both the AcNN and the CrNN, information on the plant dynamics is required. Once the IdNN has converged to an estimator of the plant, the derivative of the output with respect to the input calculated by Equations (3.17) to (3.19) can be used as an approximation to part of the plant dynamics. Equations (3.20) and (3.21) show how the previous derivative is used to build the complete dynamic description when TDL_x is greater than 1.

$$\frac{da^i(t)}{d\bar{u}(t)} = \left[0_{(nhi,nu)} \;\vdots\; \frac{da^i(t)}{d\bar{u}(t-1)_{(:,1:end-nu)}} \right] \tag{3.17}$$

$$\frac{d\,a^i(t+1)}{d\,\bar{u}(t)} = \left(I - \mathrm{diag}\,(a^i(t+1))^2 \right) W^{i1}_{(:,1:end-1)}(t+1) \begin{bmatrix} \dfrac{d\,\bar{x}(t)}{d\,\bar{u}(t)} \\ \dfrac{d\,\bar{u}(t)}{d\,\bar{u}(t)} \\ \dfrac{d\,a^i(t)}{d\,\bar{u}(t)} \end{bmatrix} \tag{3.18}$$

$$\frac{d\,x(t+1)}{d\,\bar{u}(t)} \approx \frac{d\,x^i(t+1)}{d\,\bar{u}(t)} = W^{i2}_{(:,1:nhi)}(t+1) \frac{d\,a^i(t+1)}{d\,\bar{u}(t)} \tag{3.19}$$

$$\frac{d\,\bar{x}(t)}{d\,\bar{u}(t)} = \left[0_{(nx*TDLx,nu)} \;\vdots\; \frac{d\,\bar{x}(t)}{d\,\bar{u}(t-1)_{(:,1:end-nu)}} \right] \tag{3.20}$$

$$\frac{d\,\bar{x}(t+1)}{d\,\bar{u}(t)} = \begin{bmatrix} \dfrac{d\,x(t+1)}{d\,\bar{u}(t)} \\ \cdots\cdots \\ \dfrac{d\,\bar{x}(t)}{d\,\bar{u}(t)_{(1:end-nx,:)}} \end{bmatrix} \tag{3.21}$$

The information on the plant dynamics is completed with the knowledge of how the current and past states affect the state on the next step. Therefore, there is also need to use the differential map of the IdNN to calculate the derivative of $x(t+1)$ with respect to $\bar{x}(t)$. The process through which such derivative is obtained, detailed in Equations (3.22) to (3.26), is analogous to the one performed in (3.17) to (3.21). Note that in the process of calculating both derivatives, the causality of the plant is taken into consideration.

However, while causality restricts $\frac{d\bar{x}(t+1)}{d\bar{u}(t)}$ to a block upper triangular matrix, $\frac{d\bar{x}(t+1)}{d\bar{x}(t)}$ is an upper triangular matrix with ones in the diagonal.

$$\frac{da^i(t)}{d\bar{x}(t)} = \left[0_{(nhi,nx)} \;\vdots\; \frac{da^i(t)}{d\bar{x}(t-1)_{(:,1:end-nx)}} \right] \tag{3.22}$$

$$\frac{d\,a^i(t+1)}{d\,\bar{x}(t)} = \left(I - \mathrm{diag}\,(a^i(t+1))^2 \right) W^{i1}_{(:,1:end-1)}(t+1) \begin{bmatrix} \frac{d\,\bar{x}(t)}{d\,\bar{x}(t)} \\ \frac{d\,\bar{u}(t)}{d\,\bar{x}(t)} \\ \frac{d\,a^i(t)}{d\,\bar{x}(t)} \end{bmatrix} \tag{3.23}$$

$$\frac{d\,x(t+1)}{d\,\bar{x}(t)} \approx \frac{d\,x^i(t+1)}{d\,\bar{x}(t)} = W^{i2}_{(:,1:nhi)}(t+1) \frac{d\,a^i(t+1)}{d\,\bar{x}(t)} \tag{3.24}$$

$$\frac{d\,\bar{x}(t)}{d\,\bar{x}(t)} = \left[\begin{array}{c} I_{(nx)} \\ \hdashline 0_{(nx*(TDLx-1),nx)} \end{array} \;\vdots\; \frac{d\,\bar{x}(t)}{d\,\bar{x}(t-1)_{(:,1:end-nx)}} \right] \tag{3.25}$$

$$\frac{d\,\bar{x}(t+1)}{d\,\bar{x}(t)} = \left[\begin{array}{c} \frac{d\,x(t+1)}{d\,\bar{x}(t)} \\ \hdashline \frac{d\,\bar{x}(t)}{d\,\bar{x}(t)_{(1:end-nx,:)}} \end{array} \right] \tag{3.26}$$

3.6.4. Action Neural Network (AcNN)

The core of the GDHP adaptive controller, the AcNN is responsible for the generation of the control input $u(t)$. Similar to the IdNN, the AcNN is also built on a

two-layered architecture, as can be seen in Figure 3.7 and in the network description in equations (3.27) to (3.29). Equivalently, the superscript a is used over all variables specifically related to the AcNN.

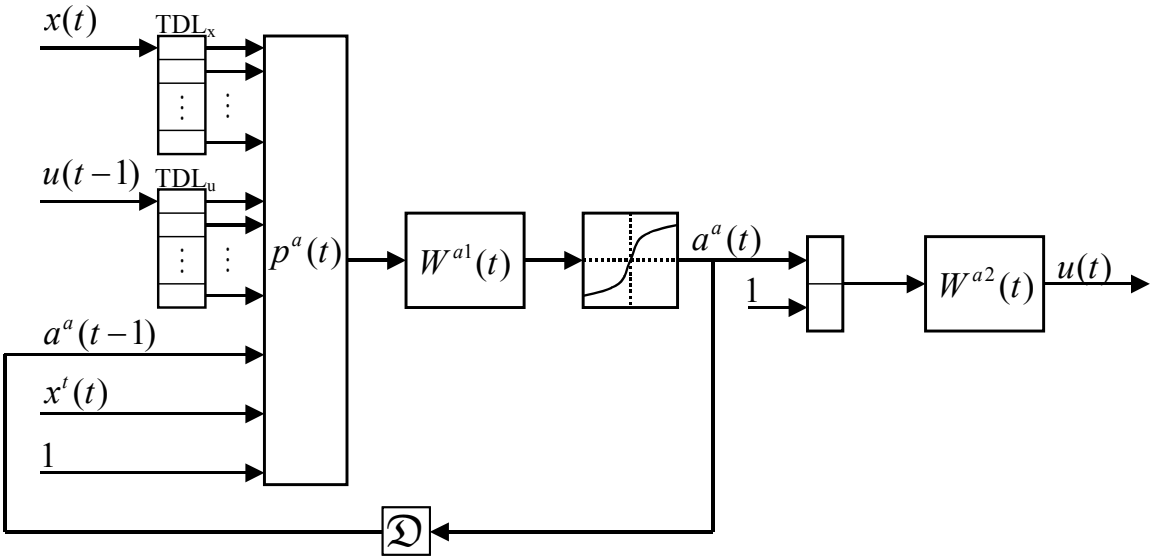


Figure 3.7. AcNN recurrent neural network architecture

$$p^a(t) = \begin{bmatrix} \bar{x}(t) \\ \bar{u}(t-1) \\ a^a(t-1) \\ x^t(t) \\ 1 \end{bmatrix} \tag{3.27}$$

$$a^a(t) = \mathrm{tansig}(W^{a1}(t)\,p^a(t)) \tag{3.28}$$

$$u(t) = W^{a2}(t)\begin{bmatrix} a^a(t) \\ 1 \end{bmatrix} \tag{3.29}$$

The training of the AcNN has the goal of producing the control sequence $u(t)$ that minimizes the cost function $J(t)$, defined in (3.2) as the sum of all future values of

the utility function $U(t)$ (3.1) with a decaying factor γ ($0 < \gamma < 1$). The diagonal matrices Q and R have the same purpose as S in the IdNN while ρ adjusts the degree at which the amount of energy spent in the control effort is penalized relative to the tracking error.

As in the IdNN, a steepest descent training algorithm was applied, resulting in the update equation (3.30). For reasons that will become clear in the description of the critic, the differentiation of $J(t)$ with respect to the weights of the AcNN is not performed directly from the infinite sum (3.2). The relationship (3.31) is used instead, resulting in equation (3.32).

$$w^a(t+1) = w^a(t) - \beta^a \left(\frac{dJ(t)}{dw^a} \right)^T \tag{3.30}$$

$$J(t) = U(t) + \gamma J(t+1) \tag{3.31}$$

$$\frac{dJ(t)}{dw^a} = \left(\frac{dU(t)}{d\bar{u}(t)} + \gamma \lambda^x(t+1) \frac{d\bar{x}(t+1)}{d\bar{u}(t)} + \gamma \lambda^u(t+1) \frac{d\bar{u}(t+1)}{d\bar{u}(t)} \right) \frac{d\bar{u}(t)}{dw^a} \tag{3.32}$$

where β^a is the learning rate of the AcNN, and $\lambda^x(t) = \frac{\partial J(t)}{\partial \bar{x}(t)}$ and $\lambda^u(t) = \frac{\partial J(t)}{\partial \bar{u}(t)}$ are outputs of the CrNN.

The next step is the calculation of the derivative of the input with respect to the weights of the AcNN. Equations (3.33-3.34) for the first layer and (3.35-3.36) for the second layer were derived in the same fashion as (3.14-3.16) of the IdNN. Equation (3.37) describes the way the full temporal derivative is obtained for both layers. It is

important to call for attention that, different from the IdNN, the AcNN is positioned in a closed loop with the plant. Therefore, in (3.33) and (3.35), the AcNN derivation path extends to include information on the dynamics of the plant, approximated by the IdNN.

$$\frac{d\,a^a(t)}{d\,w^{a1}} = \left(I - \mathrm{diag}(a^a(t))^2\right)\left(\begin{bmatrix} p^a(t)^T & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & p^a(t)^T \end{bmatrix} + W^{a1}_{(:,1:end-nx-1)}(t)\begin{bmatrix} \dfrac{d\,\bar{x}(t)}{d\,\bar{u}(t-1)}\dfrac{d\,\bar{u}(t-1)}{d\,w^{a1}} \\ \dfrac{d\,\bar{u}(t-1)}{d\,w^{a1}} \\ \dfrac{d\,a^a(t-1)}{d\,w^{a1}} \end{bmatrix}\right) \tag{3.33}$$

$$\frac{d\,u(t)}{d\,w^{a1}} = W^{a2}_{(:,1:nha)}(t)\frac{d\,a^a(t)}{d\,w^{a1}} \tag{3.34}$$

$$\frac{d\,a^a(t)}{d\,w^{a2}} = \left(I - \mathrm{diag}(a^a(t))^2\right)W^{a1}_{(:,1:end-nx-1)}(t)\begin{bmatrix} \dfrac{d\,\bar{x}(t)}{d\,\bar{u}(t-1)}\dfrac{d\,\bar{u}(t-1)}{d\,w^{a2}} \\ \dfrac{d\,\bar{u}(t-1)}{d\,w^{a2}} \\ \dfrac{d\,a^a(t-1)}{d\,w^{a2}} \end{bmatrix} \tag{3.35}$$

$$\frac{d\,u(t)}{d\,w^{a2}} = \begin{bmatrix} \begin{bmatrix} a^a(t)^T & 1 \end{bmatrix} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \begin{bmatrix} a^a(t)^T & 1 \end{bmatrix} \end{bmatrix} + W^{a2}_{(:,1:nha)}(t)\frac{d\,a^a(t)}{d\,w^{a2}} \tag{3.36}$$

$$\frac{d\bar{u}(t)}{dw^a} = \begin{bmatrix} \dfrac{du(t)}{dw^a} \\ \cdots\cdots\cdots\cdots \\ \dfrac{d\bar{u}(t-1)}{dw^a}_{(1:end-nu,:)} \end{bmatrix} \tag{3.37}$$

On Equations (3.18) and (3.32), the derivative of the tap delayed input with respect to itself was required. Equations (3.38-3.42) display how those are calculated. Since $W^a(t+1)$ is not yet available at this stage, the terms with superscript tilde are obtained using $W^a(t)$ as an approximation. Note that $p^a(t+1)$ used for the calculation of $\tilde{a}^a(t+1)$ can be generated by using the IdNN to estimate the future states of the plant assuming $x^t(t+1)$ available.

$$\frac{da^a(t)}{d\bar{u}(t)} = \left[0_{(nha,nu)} \vdots \frac{da^a(t)}{d\bar{u}(t-1)_{(:,1:end-nu)}} \right] \tag{3.38}$$

$$\frac{d\,a^a(t+1)}{d\,\bar{u}(t)} = \left(I - \mathrm{diag}\,(\tilde{a}^a(t+1))^2 \right) \tilde{W}^{a1}_{(:,1:end-nx-1)}(t+1) \begin{bmatrix} \frac{d\,\bar{x}(t+1)}{d\,\bar{u}(t)} \\ \frac{d\,\bar{u}(t)}{d\,\bar{u}(t)} \\ \frac{d\,a^a(t)}{d\,\bar{u}(t)} \end{bmatrix} \tag{3.39}$$

$$\frac{d\,u(t+1)}{d\,\bar{u}(t)} = \tilde{W}^{a2}_{(:,1:end-1)}(t+1) \frac{d\,a^a(t+1)}{d\,\bar{u}(t)} \tag{3.40}$$

$$\frac{d\,\bar{u}(t)}{d\,\bar{u}(t)} = \left[\begin{array}{c} I_{(nu)} \\ \hdashline 0_{(nu*(TDLu-1),nu)} \end{array} \vdots \frac{d\,\bar{u}(t)}{d\,\bar{u}(t-1)_{(:,1:end-nu)}} \right] \tag{3.41}$$

$$\frac{d\,\bar{u}(t+1)}{d\,\bar{u}(t)} = \left[\begin{array}{c} \frac{d\,u(t+1)}{d\,\bar{u}(t)} \\ \hdashline \frac{d\,\bar{u}(t)}{d\,\bar{u}(t)_{(1:end-nu,:)}} \end{array} \right] \tag{3.42}$$

In later developments, the information on how the future input is affected by the present states of the plant is required. For such purpose, Equations (3.43-3.47) are provided.

$$\frac{da^a(t)}{d\bar{x}(t)} = \left[0_{(nha,nx)} \;\vdots\; \frac{da^a(t)}{d\bar{x}(t-1)}_{(:,1:end-nx)} \right] \tag{3.43}$$

$$\frac{d\,a^a(t)}{d\,\bar{x}(t)} = \left(I - \mathrm{diag}\,(a^a(t))^2\right) W^{a1}_{(:,1:end-nx-1)}(t) \begin{bmatrix} \dfrac{d\,\bar{x}(t)}{d\,\bar{x}(t)} \\ \dfrac{d\,\bar{u}(t-1)}{d\,\bar{x}(t)} \\ \dfrac{d\,a^a(t-1)}{d\,\bar{x}(t)} \end{bmatrix} \tag{3.44}$$

$$\frac{d\,u(t)}{d\,\bar{x}(t)} = W^{a2}_{(:,1:end-1)}(t) \frac{d\,a^a(t)}{d\,\bar{x}(t)} \tag{3.45}$$

$$\frac{d\,\bar{u}(t-1)}{d\,\bar{x}(t)} = \left[0_{(nu*TDLu,nx)} \;\vdots\; \frac{d\,\bar{u}(t-1)}{d\,\bar{x}(t-1)}_{(:,1:end-nx)} \right] \tag{3.46}$$

$$\frac{d\,\bar{u}(t)}{d\,\bar{x}(t)} = \begin{bmatrix} \dfrac{d\,u(t)}{d\,\bar{x}(t)} \\ \cdots\cdots\cdots\cdots \\ \dfrac{d\,\bar{u}(t-1)}{d\,\bar{x}(t)}_{(1:end-nu,:)} \end{bmatrix} \tag{3.47}$$

3.6.5. Critic Neural Network (CrNN)

The third and final neural network, the critic is responsible for the estimation of the cost function $J(t)$ and of its derivatives with respect to the inputs and states ($\lambda^u(t)$

and $\lambda^x(t)$ respectively). Consistent with the notation of the other two NNs, all variables specifically related to the CrNN are marked by a superscript *c*. As shown from the network description in Figure 3.8 and Equations (3.48-3.50), the before mentioned derivatives are obtained directly as outputs of the network, instead of through backpropagation from the cost function.

$$p^c(t) = \begin{bmatrix} \bar{x}(t) \\ \bar{u}(t) \\ a^c(t-1) \\ 1 \end{bmatrix} \tag{3.48}$$

$$a^c(t) = \mathrm{tansig}(W^{c1}(t)p^c(t)) \tag{3.49}$$

$$\begin{bmatrix} \lambda^x(t)^T \\ \lambda^u(t)^T \\ J(t) \end{bmatrix} = W^{c2}(t)\begin{bmatrix} a^c(t) \\ 1 \end{bmatrix} \tag{3.50}$$

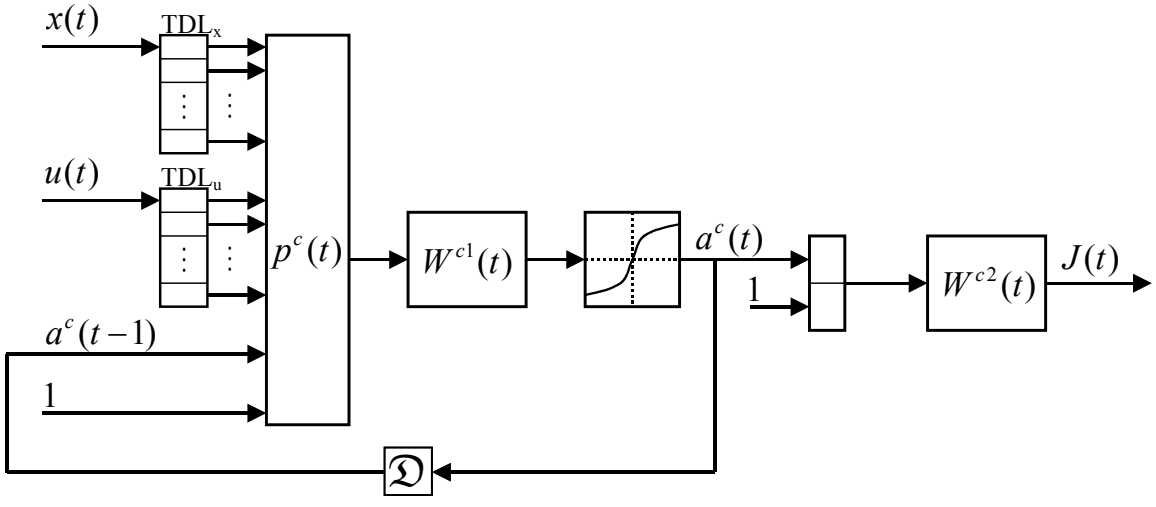


Figure 3.8. CrNN recurrent neural network architecture

The GDHP critic's weight update equation (3.51) is a combination of the training algorithms of HDP (minimizing the estimation error of $J(t)$) and DHP (minimizing the estimation error of $\lambda(t)$). Although the influence of the HDP and DHP algorithms can be decoupled in the update of the weights of the second layer, both terms equally affect all the weights of the first layer. This superposition of training approaches in the first layer of the CrNN is the main source of the synergy of GDHP [51].

$$w^c(t+1) = w^c(t) - \beta^c(1-\eta)\left(\frac{dJ(t)}{dw^c}\right)^T \left(J(t) - J^o(t)\right) - \beta^c\eta\begin{bmatrix}\frac{d\lambda^x(t)^T}{dw^c} \\ \frac{d\lambda^u(t)^T}{dw^c}\end{bmatrix}^T \left(\begin{bmatrix}\lambda^x(t) \\ \lambda^u(t)\end{bmatrix} - \begin{bmatrix}\lambda^{x^o}(t) \\ \lambda^{u^o}(t)\end{bmatrix}\right) \tag{3.51}$$

where β^c is the learning rate of the CrNN and $\eta \in [0,1]$ is a parameter that adjusts how HDP and DHP are combined in GDHP. For $\eta = 0$, the training of the CrNN reduces to a pure HDP, while $\eta = 1$ does the same for DHP.

Since the cost function $J(t)$ is a weighted sum of present and future variables, the targets $J^o(t)$, $\lambda^{x^o}(t)$ and $\lambda^{u^o}(t)$ are not analytically available when performing online learning. In order to generate values that will in time converge to the true targets, relationship (3.31) is used, resulting in Equations (3.52-3.54).

$$J^o(t) = U(t) + \gamma J(t+1) \tag{3.52}$$

$$\lambda^{x^o}(t) = \frac{\partial J^o(t)}{\partial x(t)} = \frac{\partial U(t)}{\partial x(t)} + \gamma\left(\lambda^x(t+1)\frac{dx(t+1)}{dx(t)} + \lambda^u(t+1)\frac{du(t+1)}{dx(t)}\right) \tag{3.53}$$

$$\lambda^{u^o}(t) = \frac{\partial J^o(t)}{\partial u(t)} = \frac{\partial U(t)}{\partial u(t)} + \gamma\left(\lambda^x(t+1)\frac{d\,x(t+1)}{d\,u(t)} + \lambda^u(t+1)\frac{d\,u(t+1)}{d\,u(t)}\right) \tag{3.54}$$

The next step is the calculation of the partial derivatives of the critic's outputs with respect to its weights. Equations (3.55-3.57) demonstrate how those are obtained.

$$\frac{da^c(t)}{dw^{c1}} = \left(I - \text{diag}(a^c(t))^2\right)\left(\begin{bmatrix} p^c(t)^T & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & p^c(t)^T \end{bmatrix} + W^{c1}_{(:,end-nhc:end-1)}(t)\frac{da^c(t-1)}{dw^{c1}}\right) \tag{3.55}$$

$$\begin{bmatrix} \frac{d\lambda^x(t)^T}{dw^{c1}} \\ \frac{d\lambda^u(t)^T}{dw^{c1}} \\ \frac{dJ(t)}{dw^{c1}} \end{bmatrix} = W^{c2}_{(:,1:end-1)}(t)\frac{da^c(t)}{dw^{c1}} \tag{3.56}$$

$$\begin{bmatrix} \frac{d\lambda^x(t)^T}{dw^{c2}} \\ \frac{d\lambda^u(t)^T}{dw^{c2}} \\ \frac{dJ(t)}{dw^{c2}} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} a^a(t)^T & 1 \end{bmatrix} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \begin{bmatrix} a^a(t)^T & 1 \end{bmatrix} \end{bmatrix} \tag{3.57}$$

Completing the requirements of Equations (3.53-3.54), the partial derivatives of the utility function with respect to the states and inputs are provided in Equations (3.58-3.59). Equation (3.60) shows how the full derivative of the utility function with respect to the inputs is calculated, as required in (3.32).

$$\frac{\partial U(t)}{\partial x(t)} = \left(x(t) - x^t(t) \right)^T S \tag{3.58}$$

$$\frac{\partial U(t)}{\partial u(t)} = \rho u(t)^T R \tag{3.59}$$

$$\frac{dU(t)}{d\bar{u}(t)} = \frac{\partial U(t)}{\partial x(t)} \frac{dx(t)}{d\bar{u}(t)} + \frac{\partial U(t)}{\partial u(t)} \frac{du(t)}{d\bar{u}(t)} \tag{3.60}$$

3.6.6. Complete GDHP algorithm

A key issue in all adaptive critic designs implementation is how to coordinate the online training of the three NNs. While the IdNN is trained independently since it uses information of the plant alone, the training of each AcNN and CrNN depends on the weights of the other. If no provisions are made, both networks are forced to follow a moving target, making the whole process potentially slower and likely unstable. In [33], four different strategies were discussed and compared through the application on two different test beds, demonstrating the superior performance, stability and reduced training time of a particular one that we choose to implement. Although the original work was developed for the DHP architecture, the extension to GDHP is straightforward. The strategy of interest differs from others by the fact that it utilizes two distinct NNs to implement the critic. The first (CrNN#1) outputs $J(t)$ and $\lambda(t)$ and is trained at every iteration whereas the second (CrNN#2) outputs $J(t+1)$ and $\lambda(t+1)$ and is updated with a copy of the first only once at a given period of iterations (i.e., epoch). With such training approach, it is possible to train both AcNN and CrNN continuously allowing the adaptive

critic controller to start responding to a fault as soon as it occurs. The final architecture is shown in Figure 3.9.

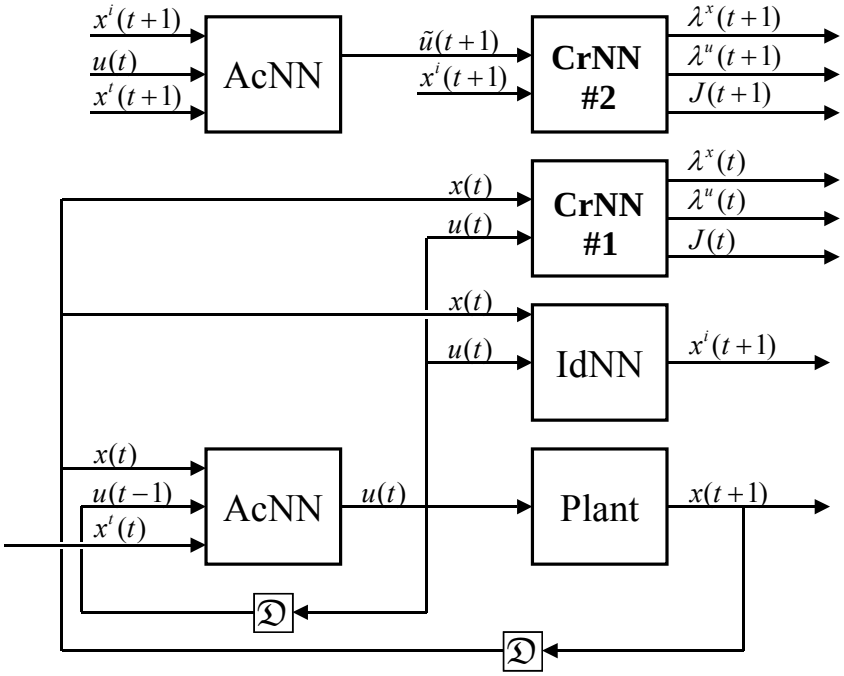


Figure 3.9. Globalized Dual Heuristic Programming

Table 3.1. Pseudocode for the presented GDHP controller

1.	Set $t=1$, e = 1. Initialize neural networks weights and network derivatives. Estimate $x^i(1)$;
2.	Sample the plant states $x(t)$ and desired trajectory $x^t(t)$;
3.	Update the weights of the IdNN by generating $w^i(t+1)$ - Equations (3.13-3.16);
4.	Feedforward through all 3 NNs (AcNN and CrNN twice) to generate in this order: $u(t)$, $x^i(t+1)$, $\tilde{u}(t+1)$, $\lambda^x(t)$, $\lambda^u(t)$, $J(t)$, $\lambda^x(t+1)$, $\lambda^u(t+1)$ and $J(t+1)$ - Equations (3.9-3.11,3.31-3.32,3.48-3.50);
5.	Calculate $U(t)$ - Equation (3.1);
6.	Backpropagate to generate $\frac{d\bar{x}(t+1)}{d\bar{u}(t)}$, $\frac{d\bar{x}(t+1)}{d\bar{x}(t)}$, $\frac{d\bar{u}(t+1)}{d\bar{u}(t)}$ and $\frac{d\bar{u}(t)}{d\bar{x}(t)}$ - Equations (3.17-3.26,3.38-3.47)
7.	Calculate $\frac{dU(t)}{d\bar{u}(t)}$ - Equations (3.58-3.59)
8.	Update the weights of the AcNN by generating $w^a(t+1)$ - Equations (3.30-3.37);
9.	Update the weights of the CrNN by generating $w^c(t+1)$ - Equations (3.51-3.57);
10.	If e = *epoch*, copy the weights of CrNN#1 to CrNN#2 and set e = 1;
11.	$t=t+1$, $e=e+1$. Return to 2.

With all the mathematical content of GDHP already available in Equations (3.1) through (3.60), a pseudocode version of the actual algorithm is presented in a condensed format in Table 3.1.

3.7. Simulation results

In order to demonstrate the capabilities of the identification network and provide a better understanding of the fine interrelations between the supervisor and DHP controller, two numerical examples are exploited. In both examples, faults are simulated by instantly or gradually changing the model of the plant. To give a better insight to the challenge of each fault scenario, linear models of fixed order similar to those employed in [38] are used here. This information however, is not used in any way during the design of the fault tolerant controller, that continues to take the plant as possessing a generic nonlinear model.

3.7.1. Identification using a recurrent neural network

The goal of the following example is to display the capabilities of the single layered recurrent network to perform the identification of linear difference systems. An input signal is supplied in the form of a fixed frequency sine wave that changes mean and amplitude only once during the simulation. Since in the final application the input to the plant generated by the actor network is not necessarily composed of a large range of

frequencies, the input with a limited spectrum represents a challenging but possible scenario in practice.

Four systems are presented in the sequence displayed in Table 3.2. The network is allowed 50 seconds for the identification of the first model, 30 for the second and 20 for the third. The fourth and final model is unstable and the applied sinusoidal input steeply drives the output to positive infinity. A variable learning rate with maximum value of 0.004 is used.

Table 3.2. Sequence of changes in the dynamics of the plant applied for the identification example

Start time (ds)	Plant dynamics
0	$y(t) = 1.810y(t-1) - 0.8187y(t-2) + 0.00566u(t-1) + 0.00566u(t-2)$
500	$y(t) = 1.810y(t-1) - 0.9000y(t-2) + 0.00566u(t-1) + 0.00566u(t-2)$
800	$y(t) = 1.810y(t-1) - 0.9048y(t-2) + 0.00242u(t-1) + 0.00234u(t-2)$
1000	$y(t) = 1.919y(t-1) - 0.9048y(t-2) - 0.00242u(t-1) + 0.00234u(t-2)$

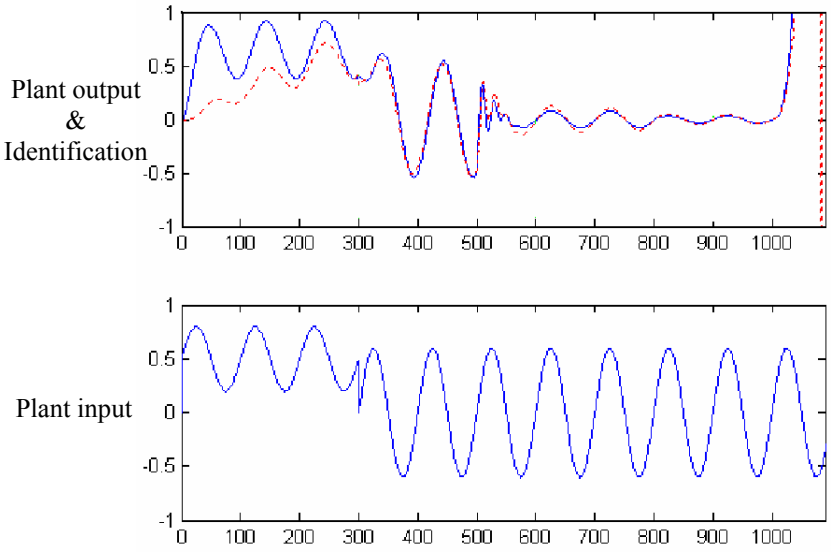


Figure 3.10. Results of the identification simulation. Plant signals are displayed in solid lines and the identification network output in dashed lines.

In Figure 3.10, the performance of the identifier can be seen. The small learning rate applied generates a slow initial reaction, but the identification signal remains close enough to the true plant output throughout the simulation in spite of the changes in the range of input and plant dynamics from model one to model three. As the fourth dynamic causes the output of the plant to grow steadily at increasing rates, it is not feasible for an identifier with a maximum learning rate to produce true identification indefinitely. Still the recurrent neural network based identifier fulfils its goal until the output becomes 45 times larger than the normal range of operation. In the complete scheme, this would allow the actor network more than 70 iterations to restructure itself in any way that would at least decrease the rate of divergence.

3.7.2. Fault Tolerant Control using a GDHP controller

This subsection brings the results of the application of the GDHP controller as a solution to the FTC problem. The power and flexibility of the combined three NNs is then demonstrated in a simulation involving a nonlinear MIMO system subject to challenging faults. Based on a numerical testbed for adaptive control of nonlinear systems introduced in [36], we propose the two-input, two-output, third-order nonlinear system described by (3.61) as the structure for the subsequent experiments.

$$
\begin{aligned}
x_1(t+1) &= \mathbf{A}x_1(t)\sin\left(x_1(t)\right)+\left(\mathbf{B}+\frac{1.5x_1(t)u_1(t)}{1+x_1^2(t)u_1^2(t)}\right)u_1(t)+\frac{2x_1(t)}{1+x_1^2(t)}u_2(t)\\
x_2(t+1) &= x_3(t)\left(1+\sin\left(4x_3(t)\right)\right)+\frac{x_3(t)}{1+x_3^2(t)}+\mathbf{C}\\
x_3(t+1) &= \left(3+\sin\left(2x_1(t)\right)\right)u_2(t)+\mathbf{D}\\
R(t+1) &= \left[x_1(t+1)\quad x_2(t+1)\right]^T,
\end{aligned}
\tag{3.61}
$$

where **A**, **B**, **C** and **D** vary according to Table 3.3, characterizing the dynamics of the system in the nominal condition, as well as under abrupt and incipient fault scenarios. In the particular case of the incipient fault, the values of **A** and **B** vary between the extremes shown in Table 3.3 in a continuous fashion through the course of 5,000 iterations. Note that the proposed faults are not limited to parameter changes alone. The introduction of new dynamics elements enables the GDHP controller to demonstrate its advanced restructuring capabilities.

Table 3.3. Modifications in the dynamics caused by the occurrence of faults

	Nominal	Abrupt Fault	Incipient Fault
A	0.9	0.9	$0.9 \Rightarrow 0.5$
B	2	-2	$2 \Rightarrow 4$
C	0	$0.7x_1(t-2)$	0
D	0	$0.5u_2(t-3)$	0

The FTC challenge is to constantly adapt the GDHP controller in order to track the trajectory defined by (3.62), while the system dynamics change in the following order: nominal → abrupt fault → nominal → incipient fault → abrupt fault.

$$R_1^r(t) = 0.5\sin\left(\frac{2\pi t}{500}\right) + 0.5\sin\left(\frac{2\pi t}{250}\right)$$

$$R_2^r(t) = 0.25\sin\left(\frac{2\pi t}{500}\right) + 0.75\sin\left(\frac{2\pi t}{250}\right) \tag{3.62}$$

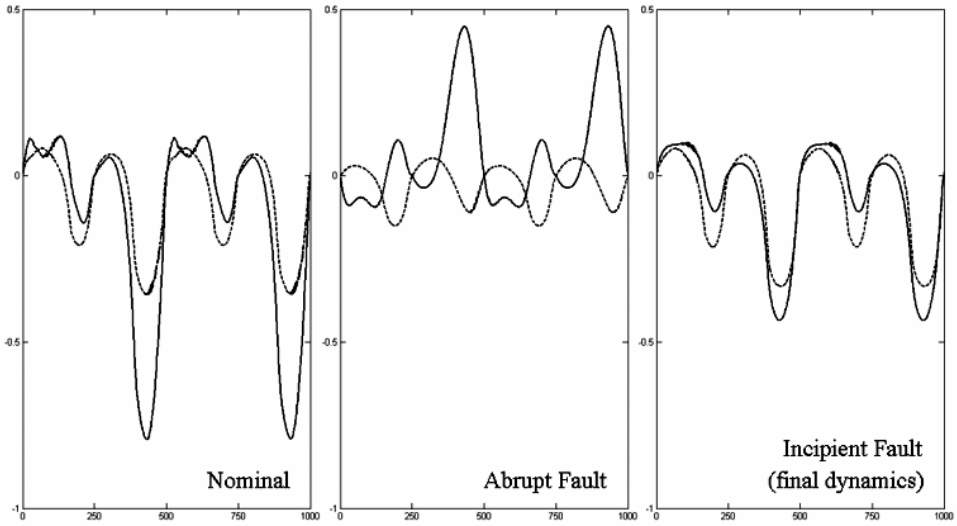


Figure 3.11. Successful control input sequences developed online by the GDHP controller for each scenario the plant assumes during the simulations. Solid lines correspond to $u_1(t)$ and dotted lines to $u_2(t)$.

Although the GDHP controller continually fine-tunes itself to reduce the tracking error, an average squared tracking error of .02 is set as a performance goal. Therefore, the reconfiguration time is defined as the interval between the occurrence of a fault (or abrupt return to the nominal condition) and the recovery of the performance goal. Figure 3.11 displays the different control efforts generated online by the GDHP controller after the

performance goal is achieved at the nominal and abrupt fault scenarios. The control inputs for the final dynamics of the incipient fault are also shown for comparison. The introduction of the incipient fault illustrates the capability of the GDHP of constantly modifying itself to account for the gradual modifications in the dynamics of the plant, while the performance is maintained throughout the process. For improvements brought by the addition of the supervisor that will become clear in the following chapter, it is important to note that the free running GDHP controller, during the transition from the abrupt fault to the nominal dynamics, required 11.4×10^3 iterations, and the transition from the incipient fault to the abrupt fault required 13.1×10^3 iterations. As an example, the output of the plant during the transition from nominal to abrupt fault is shown in Figure 3.12.

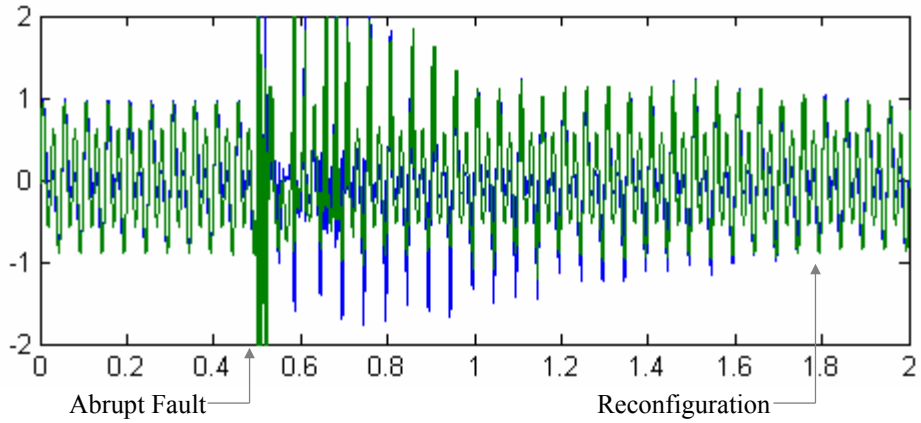


Figure 3.12. Plant output as the abrupt fault is introduced at iteration 5,000.

CHAPTER 4 – Fault Tolerant Control Supervisor

4.1. Proposed supervisor

In the proposed scheme, the goal of the supervisor is to monitor the system and manipulate the dynamic database accordingly. Measuring the quality of the control performance and analyzing its evolution through time indirectly detect faults. By augmenting this information with the knowledge of how close any model in the database approximates the current dynamics, faults are isolated into any of the currently known fault scenarios. In addition to that, faults are also automatically classified into abrupt or incipient, a key factor in the decision of when to switch to a control law present in the database.

To better understand the details of the functionality of the supervisory system, it can be divided into three layers. The first one collects and analyzes data from the plant and the controller block. The second is responsible for the decision making and the final one devises ways to implement the resolutions.

4.1.1. Quality indexes generation layer

The first layer receives the sampled output of the plant $R(t)$ and a delayed input $u(t-1)$, computes two quality indexes and indicates the known scenario that better

approximates the current dynamics. The first quality index, $q_c(t)$, measures the reconfigurable controller performance by performing a decaying integration of the primary utility function as shown in Equation (4.1),

$$q_c(t) = \int_0^t e^{-\xi_c(t-\tau)} U(\tau) d\tau \,, \tag{4.1}$$

where $0 < \xi_c < 1$ is a time decay factor. In this structure, greater time decay factors will lead to $q_c(t)$ being more affected by the quality of control actions related to instances further into the past. Concerning the occurrence of abrupt faults, adopting a ξ_c closer to the unity will lead to a more conservative supervisor that will observe the reconfigurable controller longer before adding a new solution to the DMB and will also wait for a longer period of poor performance before detecting a fault.

For the calculation of the second quality index, the delayed input is then fed into the DMB. The DMB contains information on the plant under nominal condition and under all the known fault scenarios, organized in the form of copies of the IdNN, AcNN and CrNN used to control the plant under each situation. To guarantee their specialization, no additional training is performed on networks once inserted into the DMB. Each one of the IdNNs in the DMB is then used to generate an identification error. For each of those, a decaying integration similar to Equation (4.1) is used, and the results are compared. As shown in Equation (4.2), the smallest identification error history defines the identification quality index $q_i(t)$, and the corresponding model m is appointed as the switching candidate.

$$q_i(t) = \min_{m \in M} \left(\int_0^t e^{-\xi_i (t-\tau)} \left| \hat{R}^m(\tau) - R(\tau) \right| d\tau \right), \tag{4.2}$$

where $0 < \xi_i < 1$ is a time decay factor, and $\hat{R}^m(t)$ is the vector of outputs predicted by the IdNN m, which in turn is an element of the set of M models in the DMB. As with ξ_c, the identification decay factor ξ_i can also be used to fine-tune the behavior of the supervisor. Greater identification decay factors will also lead to a more conservative supervisor in the sense that it will require more data points to conclude that the observed plant dynamics match the ones described by one of the known faults.

4.1.2. FDI and decision-making layer

In the second layer, a threshold is defined for each of the quality indexes, dividing them into high (Hq_c, Hq_i) and low (Lq_c, Lq_i) values. The threshold for $q_c(t)$ defines what is to be considered as an acceptable performance, while the one for $q_i(t)$ stipulates the degree of likeness of the input-output behavior that should be used to consider two models distinct. Four states, tagged 1 to 4, are in this way defined and the decision process illustrated in Figure 4.1 takes place. It's important to notice that in this formulation the actions of switching and adding to the database take place in the transition between states. This characteristic, added to the improved smoothness of the quality indexes bestowed by the regressive mean, aids in the generation of the hysteresis

required to prevent the automatic switching scheme to generate spurious oscillations between states.

If both indexes are low (state 1), the current reconfigurable controller is performing satisfactorily over a known environment and no action is required. While in this state, an abrupt fault may cause the performance to be degraded enough for the controller quality index to surpass its threshold. In this case, $q_i(t)$ will remain low or grow on the respective events of a known or unknown fault.

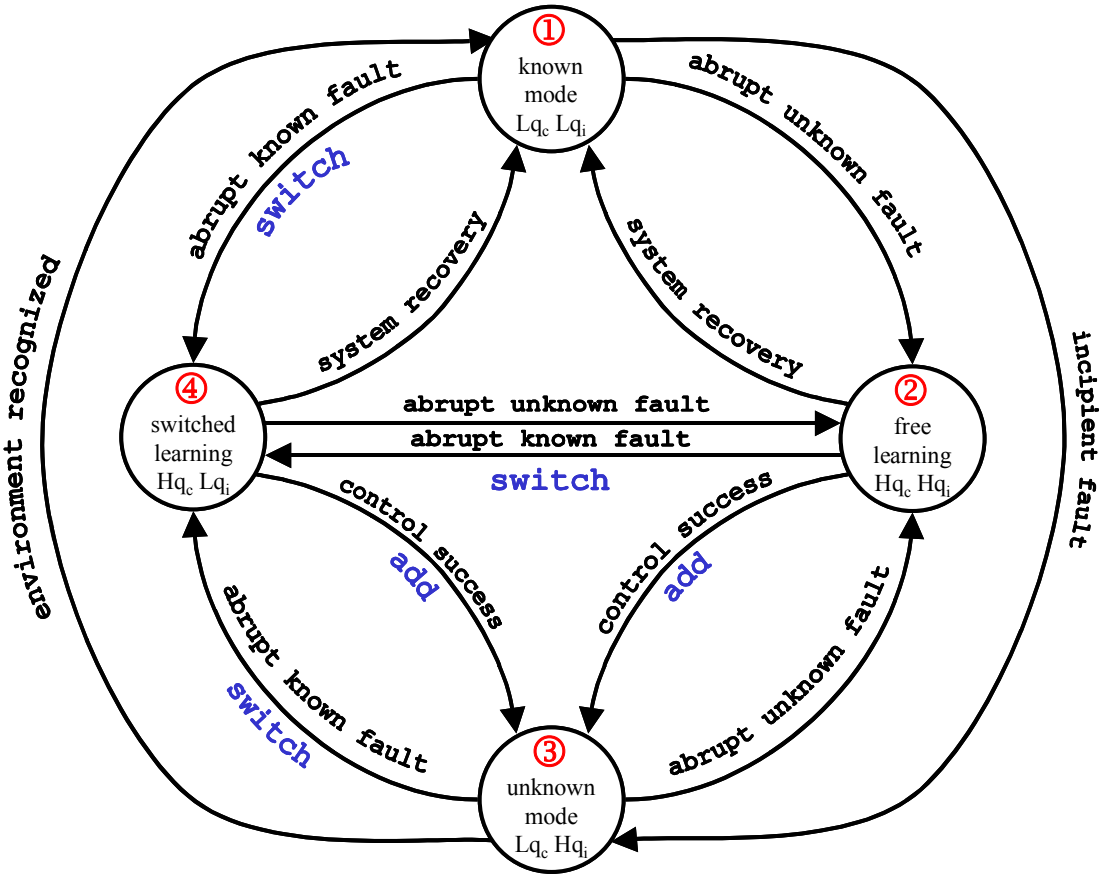


Figure 4.1. Decision graph of the second layer of the supervisory system. The states, tagged 1 to 4, are defined by the quality measures $q_c(t)$ and $q_i(t)$. The moments when the actions of switching and adding to the database are performed are shown on the graph.

If both indexes exceed the threshold (state 2), the environment has abruptly changed due to an unknown fault, and the supervisor is unable to provide any help to the DHP controller. If $q_i(t)$ remains low, there is already a set of DHP parameters in the DMB previously adapted to deal with a plant with similar dynamics and to which switching should take place. The decision process then remains in state 4 (Hq_c and Lq_i) until either the system is recovered or another fault takes place before that. If the composite fault is also a known fault, switching takes place again triggered by the change in the switching candidate appointed by the first decision layer.

Incipient faults, often connected to component aging, may be gradually adjusted by the reconfigurable controller and eventually indicate a high $q_i(t)$, even though $q_c(t)$ remains low during all the process (transition from state 1 to 3). In this case, there is no purpose in learning a new environment/controller pair since the parameters are continuously changing. As a matter of fact, if allowed to learn all the transient models, the database might rapidly grow to an intractable size.

When the DHP controller is adapting to a new environment (state 2), $q_c(t)$ is expected to decrease to the point where it crosses its threshold (transition to state 3) and a new set of parameters is added to the DMB, but two other scenarios must also be considered. The first one deals with the possibility of an abrupt known fault to happen before the first fault is completely dealt with. In this case $q_i(t)$ reaches a low value prior to $q_c(t)$ and switching to the known environment takes place. The second scenario addresses to the situation in which, due to the particular nature of the fault or controller limitations, an acceptable performance is never met for the present plant dynamics.

4.1.3. The Dynamic Model Bank layer

The third and last layer manipulates the DMB by making new entries and switching to the reconfigurable controller indicated by the fist layer, when requests arrive from the second. Switching is implemented by loading a complete set of parameters of the three neural networks (i.e., identification, action and critic networks) to the DHP algorithm currently being used. The fact that the controller is switched to one devised to a similar plant and the natural generalization capabilities of neural networks add to improve stability when the parameters are loaded as new initial conditions to the adaptive process. In the database are also stored copies of all the partial derivatives required when updating the networks using backpropagation through time. Uploading those derivatives also works to increase switching smoothness since more information about the plant new dynamics is supplied.

4.2. Performance evaluation

Throughout the course of the presented study, different FTC methods will be implemented and applied to benchmark problems. In order to compare the performance of the existing approaches and evaluate the benefits brought by the proposed techniques, more than visual inspection of a few simulation results is required.

Given a benchmark plant under a specific sequence of fault scenarios, the performance of different Fault Detection methods can be compared by the number of

misdetections and false alarms generated. A fault that goes undetected, even when the system as a whole is not significantly affected by it, reduces the available redundancy of the system. Since no indication of its occurrence is issued, the plant does not have the option of stopping for a corrective maintenance and, once the redundancy is depleted, the next fault will be unrecoverable and may lead to disastrous consequences. False alarms are generally less prone to cause such extreme consequences, however, since incorrect information is given to the supervisor, it may generate inappropriate control actions. The average and maximum detection delay times are also important parameters since detection is the first step to FTC and before that, no active response can be initiated by the controller.

In a similar spirit, fault identification quality can also be measured quantitatively by counting the number of times of misidentification and incorrect identification and measuring the identification delays. Incorrect identification will lead to control actions that may be invalid for the true dynamics of the plant with possibility of aggravating the currently detected fault scenario. Misidentification will prevent the supervisor to supply the control law already available to a known fault scenario, causing longer reconfiguration times or even unwanted responses in the cases when specific solutions are added during design time to deal with particular faults.

Representing the primary goal, availability is the most important index of success of a FTC scheme. In a simulated test sequence, availability concept can be better represented by the mean and maximum time of recovery and the number of faults a particular algorithm completely fails to address. Other performance evaluation indexes, such as required computational complexity and reconfiguration steady state performance,

may also be used to increase distinguishability and analyses of different techniques and proposed improvements.

4.3. Simulation studies

4.3.1. Discrete-time linear SISO plant

In this subsection, a numerical example based on a discrete-time SISO linear plant is used to illustrate the dynamics of the proposed FTC algorithm. Special emphasis is given to the actions of the supervisor system, which is the intelligent core of the algorithm. For the sake of simplicity and understanding in this first example, the plant consists of a linear ARMA model, which faults reflect in changes in its parameters. The models, sampled at 10Hz, used to simulate the plant under nominal operation and under each of the artificial faults are given in Table 4.1.

Table 4.1. Plant dynamics under nominal and faulty operation conditions.

Scenario	Plant dynamics
Nominal	$y(t) = 1.810y(t-1) - 0.8187y(t-2) + 0.0058u(t-1) + 0.0055u(t-2)$
Fault 1	$y(t) = 1.000y(t-1) - 0.0250y(t-2) + 0.0070u(t-1) + 0.0060u(t-2)$
Fault 2	$y(t) = 1.810y(t-1) - 0.8187y(t-2) + 0.0050u(t-1) + 0.0040u(t-2)$
Fault 3	$y(t) = 1.515y(t-1) - 0.5337y(t-2) - 0.0479u(t-1) + 0.0078u(t-2)$
Incipient	$y(t) = 1.810y(t-1) - 0.8294y(t-2) + 0.0048u(t-1) + 0.0044u(t-2)$

The incipient fault occurs over the nominal model, changing its dynamics gradually until the one given above. The simulation was carried out with the plant being abruptly changed to a different model at every 10 minutes. The goal is to follow a

trajectory composed by a sine wave that changes the amplitude randomly at every half a period. The DMB is initialized with only one model corresponding to the initial conditions of the DHP controller (small random numbers as parameters for all networks). Since in this case no beforehand information about the system is given to the plant, initially the nominal plant is treated as an unknown fault and therefore the system begins in the state where both indexes are high. Figure 4.2 shows the graphs containing the results that are discussed in detail in the following paragraphs.

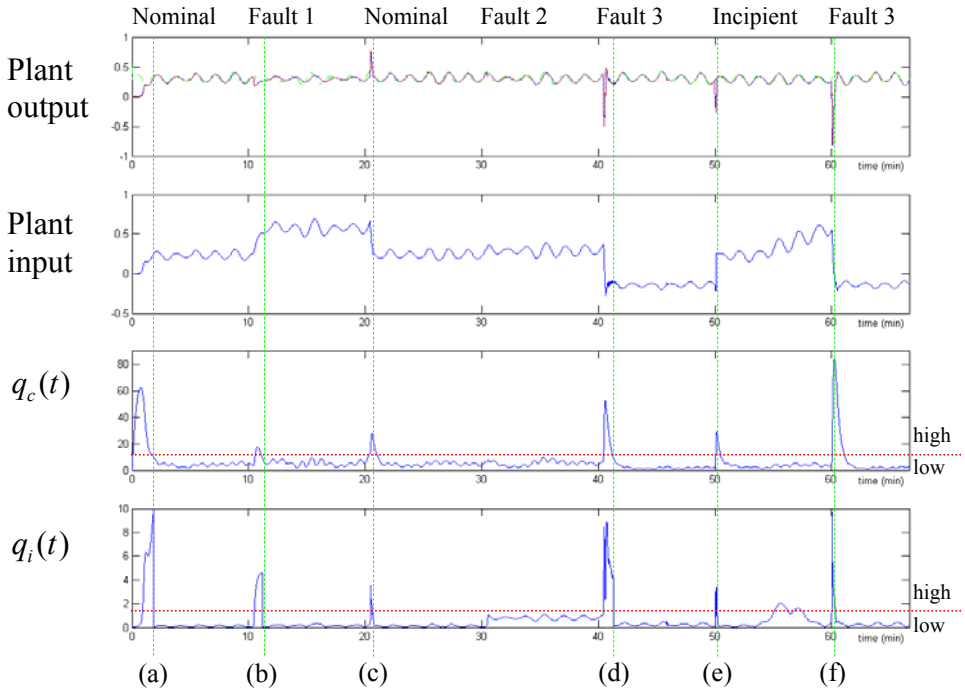


Figure 4.2. The top graph brings the desired trajectory (dashed green), the output of the plant (solid blue) and the output of the identification network while it adapts (dotted red). The second graph displays the input to the plant as calculated by the adaptive critic controller. The third and fourth graphs show the quality indexes $q_c(t)$ and $q_i(t)$ respectively, along with the thresholds used. The labels (a) to (f) indicate moments at which the supervisor acted.

After the initial transient response, as soon as $q_c(t)$ indicates a low value, the supervisor flags a control success and adds the nominal model to the DMB (indicated by

(a) in Figure 4.2). The copy of the identification network, that is now part of the DMB, generates a low identification error causing $q_i(t)$ to drop.

The identification quality index $q_i(t)$ remains low after a model is added even thought the training has been stopped and new inputs are being supplied, indicating two main achievements of the proposed FTC scheme. The first one is that the neural network used as identifier in the DHP architecture was capable of converging to represent the true dynamics of the system. The second is that the supervisor was able to recognize the proper moment when a new identifier and controller pair should be memorized.

After the first 10 minutes of simulation the first fault occurs abruptly changing the dynamics of the plant. While the adaptive controller reconfigures itself to the new scenario, both indexes grow indicating that the system is going through an abrupt unknown fault. As $q_c(t)$ drops to an acceptable level (at (b) in Figure 4.2), the first fault model is recorded along with the controller that was specifically designed on-line to deal with it.

After 20 minutes of simulation, the plant returns to the nominal mode. Due to the change in the dynamics, $q_c(t)$ increases due to the drop in the performance. On the other hand, $q_i(t)$ shows only a thin spike indicating that there already exists an element in the DMB that was previously designed to deal with a system similar (in this case identical) to the present one. Therefore switching takes place at (c) leading to a much faster response.

The second fault is introduced at 30 minutes. By comparing with the identifier adapted for the nominal plant, the supervisor concludes that the dynamics are not different enough to justify a new entry in the database. This property is of extreme importance in order to achieve a database capable of covering all the known space while

maintaining a compact set of recorded models. The third fault on the other hand, requires a major reconfiguration in the controller, and so it is also added to the DMB after convergence at (d).

After 50 minutes of simulation, the plant is instantly reverted to the nominal model and the incipient fault is applied over it. Since in the initial moments the dynamics are still similar enough to the ones of the nominal model, switching takes place at (e), shortly after 50 minutes. As the parameters of the plant are changed, the controller is capable of constantly reconfiguring itself and the tracking error remains low. As the dynamics of the plant deviate farther from the nominal ones, $q_i(t)$ increases to the point when the supervisor correctly diagnosticates the occurrence of an incipient fault. Around 57 minutes, $q_i(t)$ once again falls as the input-output relation of the plant now approximates itself to the one stored when the first fault was learned.

To illustrate the effectiveness of the algorithm when a fault presents itself for the second time, fault 3 is introduced again at 60 minutes. As soon as the environment is recognized as a known one by low values of $q_i(t)$ at (f), switching takes place generating a smother and faster response.

It is important to note that throughout this example the supervisor was capable of correctly differentiating between small changes in the dynamics and faults that required greater control law reconfiguration. Using the previously discussed decision logic, the faults were successfully classified according to their time profile and knowledge stored in the database. As the DHP controller generated identification models and control laws to counter the unknown abrupt faults, the supervisor incorporated both to the database at the adequate moments. As models were presented for the second time, switching to the

previously adapted control laws took place to accelerate recovery and improve transient response.

4.3.2. Continuous-time linear SISO plant

In the spirit of [38], for the sake of simplicity and understanding, the plant consists of a simple linear model, subject to faults resulting in changes in its parameters and order of the numerator and denominator polynomials. The models, sampled at 10Hz, used to simulate the plant under nominal operation and under each of the artificial faults, are given in Table 4.2.

Table 4.2. Nominal and Fault Dynamics

Nominal	Fault 1
$G_n(s) = \frac{1.25}{s^2 + 2.0s + 1.0}$	$G_{f1}(s) = \frac{-0.108s + 4.95}{s^2 + 36.9s + 9.5}$
Fault2	**Fault 3**
$G_{f2}(s) = \frac{0.004s + 0.99}{s^2 + 2.0s + 1.0}$	$G_{f3}(s) = \frac{-0.349s - 5.41}{s^2 + 6.3s + 2.5}$
Incipient	
$G_{inc}(s) = \frac{(0.0001t_i)s + (1.25 - 0.1t_i)}{s^2 + (2 - 0.01t_i)s + (1 + 0.11t_i)}$	

It is important to call to attention now that the incipient fault occurs over the nominal dynamics, changing it gradually with time since its occurrence t_i (in minutes). The simulation was carried with the plant being abruptly changed to a different model at every 16.67 minutes (10^4 iterations). The goal is to follow a reference trajectory composed of a sine wave that randomly changes the amplitude at every half a period.

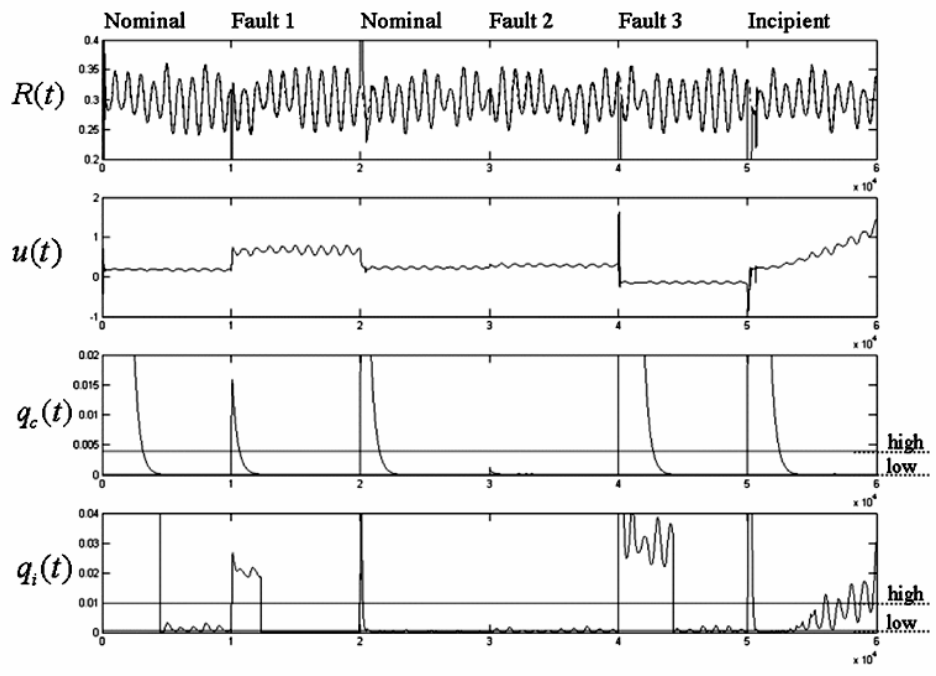


Figure 4.3. The top graph shows the desired trajectory (dashed) and the output of the plant (solid). The second graph displays the input to the plant as calculated by the GDHP controller. The third and fourth graphs show the quality indexes $q_c(t)$ and $q_i(t)$, along with the thresholds used.

Since in this case no beforehand information about the system is given to the plant, initially the nominal plant is treated as an unknown dynamics and therefore, as displayed in Figure 4.3, the system begins with both quality indexes high (State 2 of the supervisor's decision logic). After the initial transient response, as soon as $q_c(t)$ indicates a low value and the decision logic moves to State 3, the supervisor flags a control success and adds the nominal model to the DMB. The copy of the identification network, that is now part of the DMB, generates a low identification error causing $q_i(t)$ to drop sharply, leading to State 1. This sequence of events can be traced in the decision logic's state transitions highlighted in Figure 4.4 (a). For more details on the transition, the readers are referred to Figure 4.1. The identification quality index $q_i(t)$ remains low after a model is

added even though the training has been stopped and new inputs are being supplied, indicating two main achievements of the proposed FTC scheme. The first one is that the neural network used as an IdNN in the GDHP architecture was capable of converging to represent the true dynamics of the system. The second is that the supervisor was able to recognize the proper moment when a new set of neural networks should be memorized.

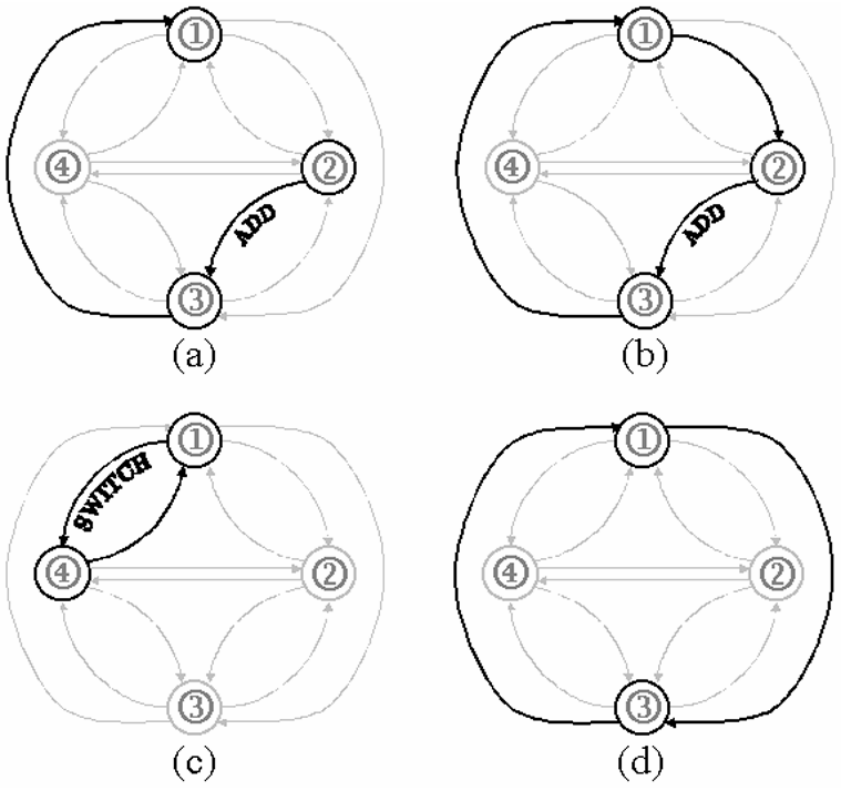


Figure 4.4. Four key transition sequences in the decision logic of the FTC supervisor. (a) adding the nominal model to the DMB; (b) adding an abrupt fault model to the DMB; (c) switching to a known solution; (d) dealing with an incipient fault.

After the first 10^4 iterations Fault 1 is introduced, abruptly changing the dynamics of the plant. While the reconfigurable controller adapts itself to the new scenario, both indexes grow (State 2), indicating that the system is going through an abrupt unknown fault. As $q_c(t)$ drops to an acceptable level (State 3), the first failure mode is recorded

along with the controller that was specifically designed on-line to deal with it. The decision logic shortly moves to State 1, as the new IdNN inside the DMB is now capable of describing the dynamics of the first fault. This sequence of events is perceived by the supervisor's decision logic through the state transitions shown in Figure 4.4 (b).

At iteration 2×10^4, the plant returns to the nominal mode. Due to the change in the dynamics, $q_c(t)$ increases due to the degradation in the performance. On the other hand, $q_i(t)$ shows only a thin spike indicating that there already exists an element in the DMB that was previously designed to deal with a system similar (in this case identical) to the present one. Following the FTC supervisor's decision logic shown in Figure 4.4 (c), switching takes place in the transition from State 1 to State 4, leading to a quicker and more precise recovery. Figure 4.5 illustrates such benefits by displaying the plant output at critical moments and comparing the results improved by supervisory action with the outcome of a free-running GDHP controller (i.e., without the FTC supervisor intervention) when faced with the same fault sequence.

Going back to the complete timeline shown in Figure 4.3, Fault 2 is introduced at iteration 3×10^4. By comparing with the IdNN adapted for the nominal plant, the FTC supervisor concludes that the dynamics are not different enough to justify a new entry in the DMB. This property is of extreme importance in order to achieve a DMB capable of covering all the known space, while maintaining a compact set of recorded models. Fault 3, on the other hand, requires major reconfiguration in the controller. At iteration 44,218 the supervisor determined that a solution to Fault 3 has been successfully designed and so it is also added to the DMB.

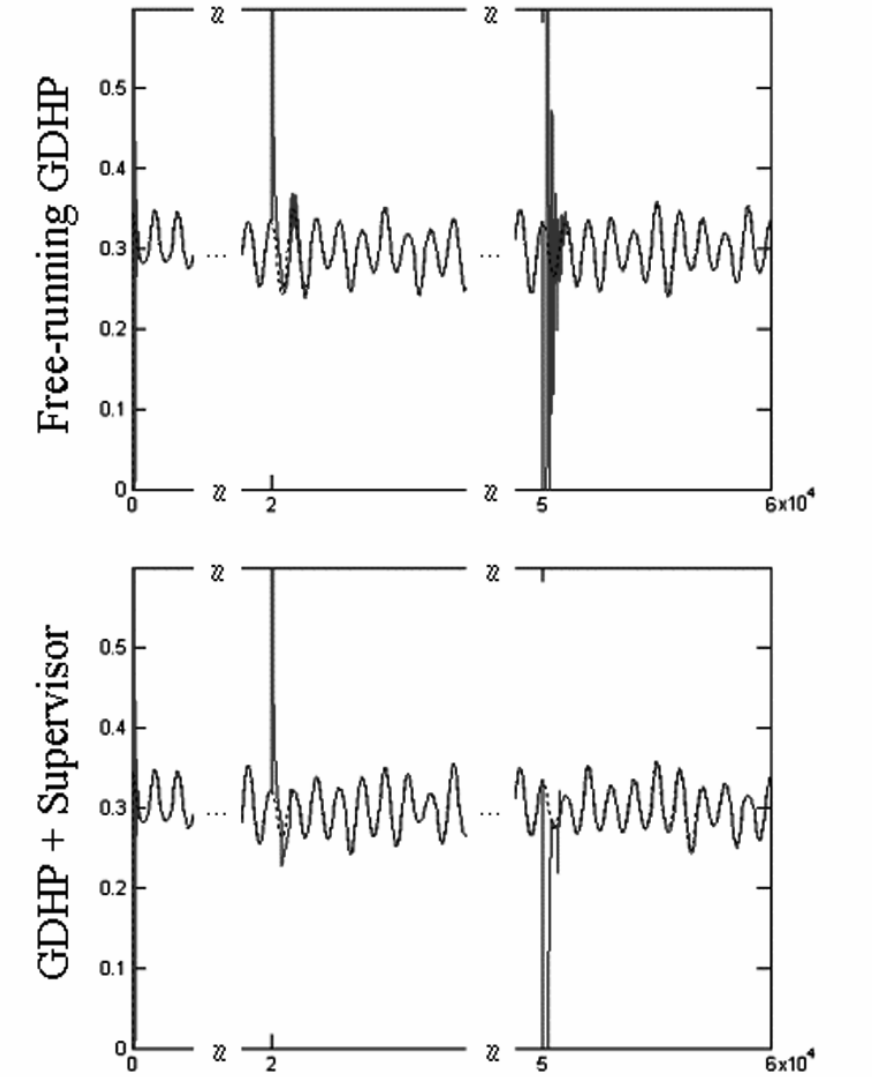


Figure 4.5. Plant output (solid) and desired trajectory (dashed) display the increase in performance and reconfiguration time brought by the application of the proposed FTC supervisor.

After iteration 5×10^4, the plant dynamics abruptly revert to nominal and are shortly followed by the gradual changes brought by the incipient fault. Since in the initial moments the dynamics are still similar enough to the ones of the nominal model, switching takes place shortly after iteration 5×10^4 following once again the path shown in Figure 4.4 (c). Figure 4.5 displays the dramatic reduction in the recovery time and oscillation brought by the switch operation. As the parameters of the plant are changing over time during the incipient fault, the controller is capable of constantly reconfiguring

itself, and the tracking error remains low. As the dynamics of the plant gradually become more different from the nominal ones, $q_i(t)$ increases to the point when the decision logic moves from State 1 to State 3 and the supervisor correctly diagnoses the occurrence of an incipient fault. The response of the decision logic to an incipient fault is shown in Figure 4.4 (d).

4.3.3. Continuous-time nonlinear MIMO plant

In this Subsection, we revisit the nonlinear MIMO plant explored in Section 3.7.2 when a GDHP controller was used as a FTC solution by itself. Recall that the plant in question is a two-input, two-output, third-order nonlinear system described by Equation (4.61) and the goal is to track the trajectory described by Equation (4.62) while the plant goes through a sequence of unexpected (from the point-of-view of the supervisor) fault scenarios.

In the first simulation, without a supervisor, the GDHP controller alone managed to produce satisfactory results provided enough time was allowed for a solution to evolve online. As commented previously however, although the GDHP controller is a successful nonlinear adaptive controller, due to the lack of long-term memory, long reconfiguration times have to be allowed even when the plant abruptly changes to previously visited scenarios. In the particular case of the simulation presented in Chapter 3, the transition from the abrupt fault to the nominal dynamics required 11.4×10^3 iterations, and the transition from the incipient fault to the abrupt fault required 13.1×10^3 iterations.

In this chapter the proposed supervisor is added to the architecture of the simulated FTC solution. Knowledge of the nominal dynamics was assumed to exist during design time, and so the DMB is initialized with knowledge of a solution for this particular scenario. On the other hand, no knowledge about the fault dynamics is introduced prior to the experiment.

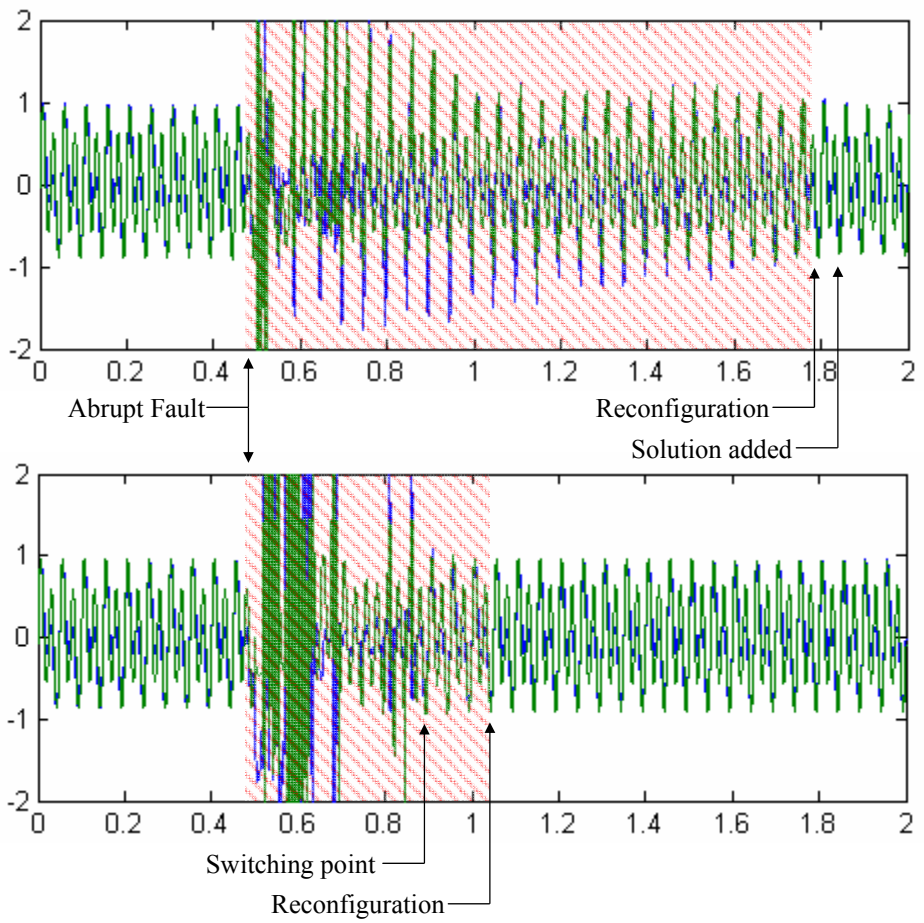


Figure 4.6. Two plots of the plant output as the abrupt fault is introduced at iteration 5,000. Top plot: the supervisor has no knowledge of the fault in the DMB. Bottom plot: the supervisor accelerates reconfiguration by switching to a previously stored solution. Reconfiguration time is indicated by the highlighted area.

The key events of the second simulation concerning the handling of the twice the abrupt fault manifests have been summarized in Figure 4.6. Forty three iterations after the

transition from the nominal to the abrupt fault dynamics, the supervisor declares the occurrence of an abrupt unknown fault. As the GDHP controller converges to a satisfactory solution for the abrupt fault scenario, the supervisor adds copies of the three neural networks to the DMB. Having acquired the knowledge of an effective control solution, the supervisor recognizes when the same dynamics appear in the transition from incipient back to the abrupt fault and performs a switch operation at iteration 3.7×10^3 after the fault took place. The supervisor's intervention led to a reduction of the reconfiguration time to 5.4×10^3 iterations.

Performing a similar operation, the supervisor also makes use of the knowledge provided during design time to accelerate the reconfiguration of the GDHP controller when the plant returns to the nominal dynamics after going through the abrupt fault scenario. Switching reduces the reconfiguration time of the transition between the abrupt fault to the nominal dynamics to 5.2×10^3 iterations. When compared to the results of the first simulation, the contribution of the supervisor reduced the reconfiguration time of the GDHP controller by a factor greater than two in all transitions in which knowledge was already present inside the DMB.

4.4. Summary

In this chapter, a multiple model approach to FTC based on an intelligent Dynamic Model Bank is proposed. The application of GDHP as a reconfigurable controller is shown to give the hierarchical algorithm the degree of flexibility required to deal with both abrupt and incipient changes in the plant dynamics due to faults. The FTC

supervisor system is used to accelerate the convergence of the method by loading new initial conditions to the GDHP when the plant is affected by a known abrupt fault. A decision logic is presented through which new fault scenarios are recognized and assimilated on-line by the DMB along with parameters for the corresponding controller. Through a synergistic integration of these essential elements the online fault tolerant control has become feasible. Finally, these properties are successfully illustrated in the in-depth exploration of numerical simulation examples.

CHAPTER 5 – Controller Malfunction Detection and Recovery

5.1. Introduction

In the previous chapters of this report, we have proposed, as a solution to the FTC problem, the use of a nonlinear adaptive controller running under a FTC supervisor shown to improve the performance of the underlining controller in the event of faults. Since the reasoning behind the performance improvement brought by the FTC supervisor is the same for any nonlinear adaptive controller, the choice of an adaptive critic design as the baseline controller is based on other independent factors. In fact, adaptive critic designs (specifically the implemented GDHP controller) are capable of achieving superior performance by combining three different NNs: an identification NN that produces information of the dynamics of faults as they occur, a critic NN to estimate the impact of the underlying control strategy over time; and an action NN to generate real-time control actions to accommodate faults as they emerge.

The questions that arise when using NNs as building blocks of an adaptive controller are how to perform online learning in an effective manner for the most diversified scenarios and how to guarantee stability. As for the first question, NN has been well regarded as an effective tool in function approximation due to its universal nonlinear mapping capability [27]. However, under the current state-of-the-art NN

designs, proof of convergence still requires that widely restrictive conditions are met, as the ones presented in [52] and [53]. Independent of its implementation, adaptation flexibility and stability are still conflicting specifications for all nonlinear adaptive controller paradigms and therefore a suitable compromise must be reached for each application. Since the goal of the presented work requires that solutions be found even when an unknown nonlinear fault becomes active, adaptation flexibility and nonlinear mapping power are essential. Therefore, the choice of NNs as function approximators, specially when applied in an adaptive critic architecture, is justified. However it is important to account for the fact that the unpredictable occurrence of faults that may assume unrestricted dynamics do not allow for any measure that will guarantee that the online training of weights of a NN will converge to an optimal configuration.

Although a GDHP architecture using NNs as building blocks represents one of the best available implementations of an universal nonlinear adaptive controller, there are still no available adapting procedures for it or any other equivalent adaptive nonlinear controller that guarantees complete stability over all fault scenarios, specially when it is taken into consideration the fact that abrupt faults inherently cause discontinuous changes in the plant dynamics. Independent of its particular implementation, the adaptive nonlinear controller then becomes a vulnerable point of the FTC architecture as it may be subject to controller malfunctions such as divergence and convergence to a local minima through the course of its online adaptation. If left unchecked, such situation may lead to loss of availability even when recovery of plant stability was still reachable.

In the previous chapters, we have demonstrated how a GDHP controller can be used to autonomously generate control solutions for unexpected faults while explicitly

producing input-output maps through the use of neural networks. Furthermore, located inside the supervisor, the DMB stores, for the nominal plant and each known fault scenario (added during design time or learned online), an input-output identification map and a controller that has been successfully designed for each specific scenario. A key characteristic that makes the proposed supervisor capable of such accomplishments is its ability to perform Fault Detection and Diagnosis of the system. Such is accomplished through a four-state decision logic based on the two previously detailed quality indexes.

In order to account for the inherent possibility of instability within the online training procedure of the GDHP controller, we propose to add to the FDD scheme Controller Malfunction Detection (CMD) capability within the nonlinear adaptive controller itself. In the case of implementations involving the online training of NNs in particular, there are two controller malfunctions that must be avoided. The first relates to the training process converging to a local minima that prevents the NN weights to reach the global optimal, while the second refers to weight divergence due to the application of the training algorithm in close loop with unknown dynamics of a faulty plant. To properly identify these two controller malfunctions and discern them from the plant faults, a third quality index is introduced. By measuring the degree of activity within the NNs that compose the GDHP architecture, the weight quality index is capable of adding to the information gathered by the other two quality indexes to achieve such goals. Once a control malfunction is detected and identified, it is then possible to use the information from the DMB to provide the training algorithm with a new set of initial conditions that represents the closest possible known dynamics and effectively prevent divergence and greatly increase the chance of recovery of stability and performance.

5.2. Controller malfunction supervisor

In this section the previously presented Supervisor capable of performing system FDD is augmented to also perform CMD and suggest countermeasures for such scenarios on-the-fly. To better understand the functionality of the supervisor, it can be divided into three layers as shown in Figure 5.1. Key differences here are the fact that a third novel quality index is also calculated in the first layer and that there are now two distinct decision logics, one for FDD and another for controller malfunction detection

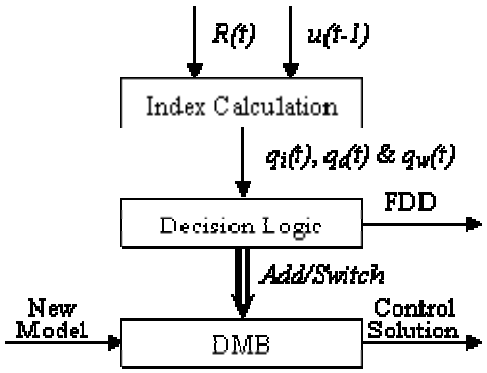


Figure 5.1. Layered structure of the proposed Supervisor with controller malfunction detection and recovery.

In addition to the control quality index $q_c(t)$ and the identification quality index $q_i(t)$, the novel quality index generated in the first layer is the weight quality index $q_w(t)$. While the control quality index, obtained through Equation (4.1), provides a numerical measure of how close has the GDHP controller been able to follow its goal, in this case trajectory tracking, the weight quality index is calculated in order to measure the amount of activity within the three continuously adapting neural networks (i.e., IdNN, AcNN and CrNN#1) as shown in Equation (5.1).

$$q_w(t) = \int_{t_s}^{t} e^{-\xi_w(t-\tau)} \left[\left\| w^i(\tau) - w^i(\tau - \Delta t) \right\| + \left\| w^a(\tau) - w^a(\tau - \Delta t) \right\| + \left\| w^c(\tau) - w^c(\tau - \Delta t) \right\| \right] d\tau \,, \tag{5.1}$$

where ξ_w is a time decay factor in the range [0,1], Δt is the sampling period and t_s is the time of occurrence of the latest switching operation. The presence of t_s in Equation (5.1) is a key feature to understand the interrelations between the detection and diagnosis of faults inside the plant and the detection of faults within the controller in what is called controller malfunctions. In essence, t_s resets the memory of the filter used in the calculation of q_w to prevent the switching operations carried out by the supervisor (in response to faults within the plant or controller) to be perceived as major changes in the weight structure, which are associated with controller divergence during neural network training.

As before, in the second layer, FTC design parameters are converted into thresholds to distinguish high (Hq_c, Hq_i) and low (Lq_c, Lq_i) values for the control and identification quality indexes. For the weight quality index, two thresholds are used to distinguish between high activity Hq_w, standard activity Sq_w and low activity Lq_w. Standard activity is adjusted as a broad range within which the activity of the networks remain during successful training.

Table 5.1. Complete Algorithm (GDHP and Supervisor)

1	Set $t = 1$. Initialize neural networks and estimate $\hat{R}(t)$. Initialize DMB with knowledge of the dynamics of the healthy plant and of known fault scenarios;
2	Sample the plant output $R(t)$ and desired trajectory $R^t(t)$;
3	Update IdNN in the direction of the minimization of the quadratic estimation error of $R(t)$;
4	In the following order, feedforward to obtain: $u(t)$ from AcNN, $\hat{R}(t+1)$ from IdNN, $u(t+1)$ from AcNN, $J(t)$ and $\lambda(t)$ from CrNN#1, $J(t+1)$ and $\lambda(t+1)$ from CrNN#2;
5	Update AcNN in the direction of the minimization of $J(t)$;
6	Update CrNN#1 in the direction of the minimization of the quadratic estimation error of $J(t)$ and $\lambda(t)$;
7	$epoch = epoch - 1$. If epoch = 0, weights of CrNN#1 are copied to CrNN#2 and *epoch* is reset;
8	Calculate the quality indexes $q_i(t)$, $q_c(t)$ and $q_w(t)$;
9	Perform controller malfunction detection (to be shown in Figure 5.1). If divergence or local minima convergence is detected, skip to step **11**;
10	Perform Fault Detection and Diagnosis through the supervisor's decision logic (shown in Figure 4.2);
11	According to the decision logic, manipulate the DMB to switch to a know solution, add a new model to it or provide a new set of initial conditions;
12	$t = t + 1$. Return to step **2**;

As shown in Table 5.1, the controller malfunction detection takes precedence over the FDD, ensuring that only a functional GDHP controller is used for fault recovery or compensation and that malfunction within the IdNN do not interfere with the FDD process. By using the information contained in the controller and the weight quality index, it is possible to verify the condition of the GDHP nonlinear adaptive controller over two different malfunctions. The first one relates to the online GDHP training converging to a local minima and therefore being incapable of reaching the optimal tracking error. Such malfunction is characterized by a high control quality index (Hq_c) matched by a low weight quality index (Lq_w). The second malfunction relates to controller divergence and is marked by both a high control quality index (Hq_c) and a

high weight quality index (Hq_w). Figure 5.2 displays a flow chart that illustrates the controller malfunction detection procedure engaged at every iteration. Note that in a healthy training situation, a high control quality index, and therefore elevated tracking error over time, would generate changes in the weight structure of the neural networks that compose the GDHP controller and therefore indicate Sq_w . In order to increase noise rejection and reduce the probability of false alarms, an observation time is used during which such conditions must remain unaltered so that a controller malfunction can be positively detected. Assuming that no control malfunctions are detected, FDD is then performed through the decision process presented in Chapter 4.

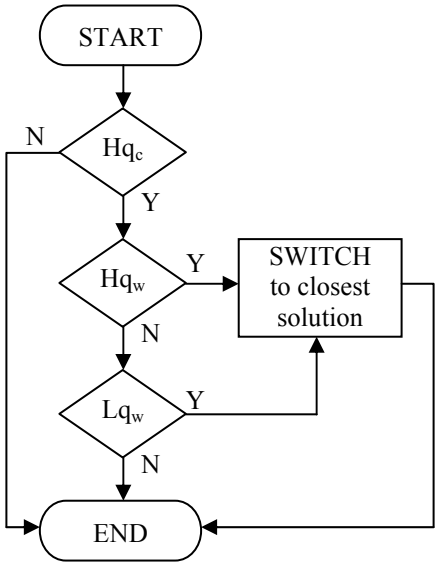


Figure 5.2. Flow chart for controller malfunction detection and response.

5.3. Numerical example

In this section, an extensive simulation is presented and discussed. The goal is to demonstrate the feasibility of implementation and the FTC benefits brought by the

proposed architecture over a controlled plant subject to one incipient and two abrupt faults. All faults introduce strong nonlinear dynamics requiring major controller reconfiguration. On certain key steps, we also show the results of parallel simulations that differ only in the fact that the supervisor is deactivated. The comparison of simulations with and without supervisory intervention serves two purposes. First, it allows for the demonstration of the dramatic reduction on the reconfiguration time and cumulative tracking error of the GDHP controller brought by the switching operation in the occurrence of faults. Second, it illustrates the supervisor's controller malfunction detection capability and its potential to advert negative consequences of such scenarios by providing new initial conditions to the online training GDHP controller.

5.3.1. Simulated system

The simulated plant possesses two inputs $u = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^T$, three states $x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$ and two outputs $R = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$ which are expected to track two independent trajectories $R^t = \begin{bmatrix} R_1^t & R_2^t \end{bmatrix}^T$. The desired trajectories are described by Equation (5.2),

$$R_1^t(t) = \mu_1 \sin\left(\frac{t\pi}{250}\right) + 0.4\sin\left(\frac{t\pi}{125}\right)$$

$$R_2^t(t) = 0.1 + \mu_2 \sin\left(\frac{(t+150)\pi}{250}\right) + 0.6\sin\left(\frac{(t+190)\pi}{125}\right), \tag{5.2}$$

where μ_1 and μ_2 assume values in the interval [0 , 0.5] randomly selected once at every 500 iterations. An example of the reference signals can be seen in Figure 5.3. For the sake of the presented simulation, tracking is considered satisfactory if the mean error over a cycle of 500 iterations is less than 0.001.

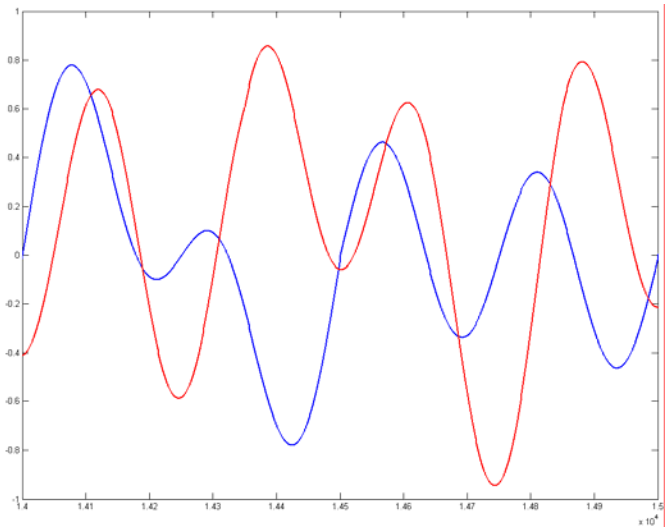
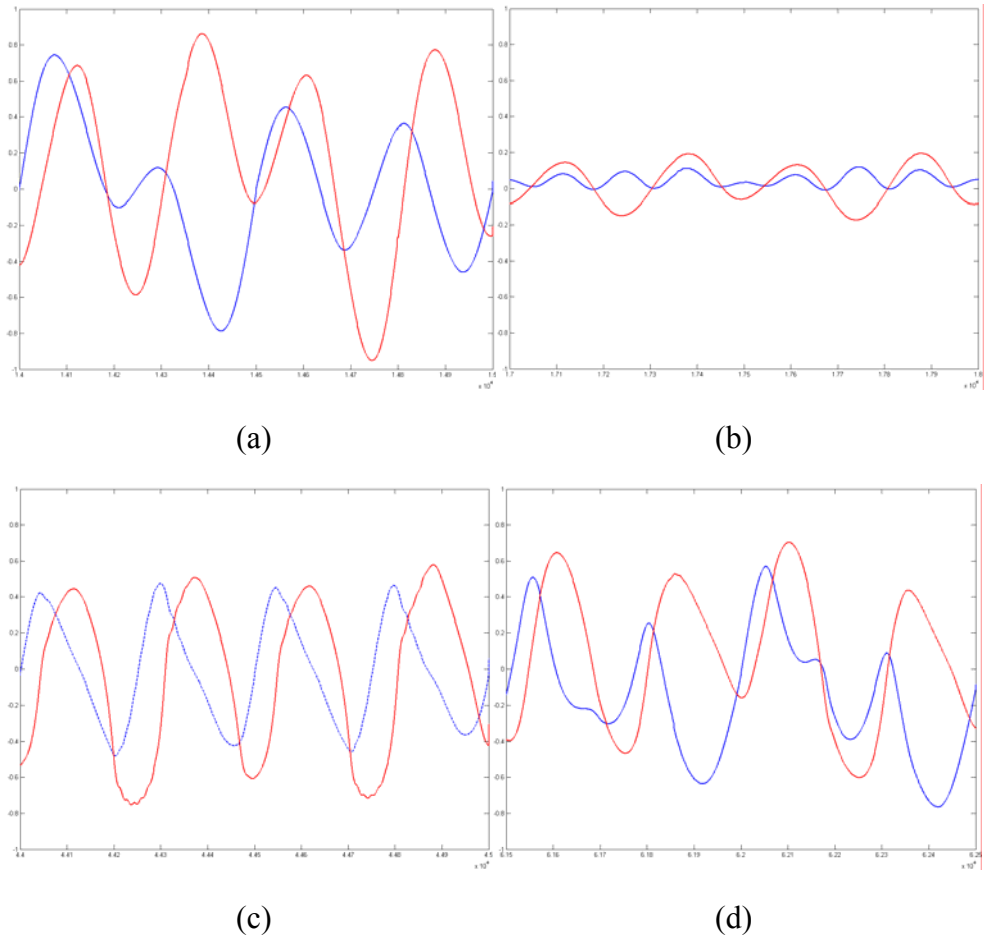


Figure 5.3. Example of reference signals $R_1^t(t)$ (blue) and $R_2^t(t)$ (red).

Under nominal conditions (i.e., no fault is active), the system's dynamics allow for the desired trajectories to be reached by directly inputting properly shifted reference signals, as described by Equation (5.3). In practice, such straightforward nominal dynamics are expected when the fault tolerant controller is mounted over a pre-designed nominal controller. For example, the nominal dynamics of the plant simulated here could represent the closed loop dynamics of the actual physical system and a non-adapting inverse-dynamics controller designed off-line for the nominal operation conditions. Furthermore, the choice of such nominal dynamics helps to make clear the impact of each

particular fault scenario. A sample input sequence that achieves trajectory tracking under the nominal scenario can be seen in Figure 5.4 (a).

$$\begin{cases} x_1(t+1) = u_1(t) \\ x_2(t+1) = x_3(t) \\ x_3(t+1) = u_2(t)\,. \end{cases} \tag{5.3}$$



(a) (b)

(c) (d)

Figure 5.4. Successful input sequences $u_1(t)$ (blue) and $u_2(t)$ (red) for different plant dynamics: (a) nominal, (b) AF1, (c) IF and (d) AF2.

Through the course of the simulation, we introduce two abrupt nonlinear faults: Abrupt Fault 1 (AF1) that changes the system's dynamics to Equation (5.4), and Abrupt

Fault 2 (AF2) that modifies it to Equation (5.5). By displaying input sequences that lead the system's outputs to the desired trajectories, Figure 5.4(b) and 5.4(c) allows the reader to visualize the amount of controller reconfiguration required under AF1 and AF2. For both faults, the changes in the dynamics take place instantly at their time of occurrence. Therefore, both constitute challenging scenarios for the online-adapting GDHP controller.

$$\begin{cases} x_1(t+1) = 1.5u_1(t) + x_1(t) - \left(x_3(t)\right)^2 - \sin\left(\dfrac{\pi}{4}u_1(t)\right) \\ x_2(t+1) = 0.2x_2(t) + x_2(t)u_1(t) + 2\dfrac{x_3(t)}{1+\left(x_3(t)\right)^2} \\ x_3(t+1) = u_2(t) + 0.6u_2(t)\sin\left(0.5x_1(t)\right) + 0.4x_3(t)\,. \end{cases} \tag{5.4}$$

$$\begin{cases} x_1(t+1) = u_1(t) + \left(x_3(t)\right)^2 \\ x_2(t+1) = x_3(t) - 0.8x_2(t)u_1(t) \\ x_3(t+1) = 1.5u_2(t) - 0.5x_3(t)\,. \end{cases} \tag{5.5}$$

To simulate the occurrence of an incipient fault (IF), the dynamics of the nominal system are gradually modified over the course of 10^4 iterations as described by Equation (5.6), where t_i is used to adjust the time of occurrence of the IF so that by the end of its interval of occurrence the nonlinear terms take full effect. The use of the tansig function creates a smooth gradual introduction of the incipient dynamics with steep changes in the middle range. Note that not only new nonlinear dynamics are introduced, but also the states of the system become coupled demanding more complex controller action. Figure

5.4(d) displays the control action required by the final dynamics of the plant in order to track the desired trajectory.

$$\begin{cases} x_1(t+1) = u_1(t) + \left(0.5 + 0.5\,\text{tansig}\left(\dfrac{t-t_i}{1500}\right)\right)\dfrac{x_1(t)}{1+\left(x_1(t)\right)^2}u_2(t) \\ x_2(t+1) = x_3(t) + \left(0.5 + 0.5\,\text{tansig}\left(\dfrac{t-t_i}{1500}\right)\right)\dfrac{\sin\left(4x_3(t)\right)}{2}x_3(t) \\ x_3(t+1) = u_2(t)\,. \end{cases} \tag{5.6}$$

5.3.2. Simulation results

The simulation starts with the plant under nominal dynamics for the first 15,000 iterations. The GDHP controller is initialized with a set of weights previously designed for the nominal dynamics. The DMB inside the supervisor is initialized with a copy of the same nominal weights and, therefore, it is initialized with the knowledge of how to identify and control the plant under nominal dynamics. Hence, both $q_i(t)$ and $q_c(t)$ start with low values (Lq_i, Lq_c) corresponding to State 1 in the FDD decision logic. In order to demonstrate the supervisor responses to all key circumstances it is designed for, faults were introduced into the system according to the schedule in Table 5.2. Although displaying the entire 65,000 iterations of the simulation would be optimal for the visualization of the complete challenge set for the proposed architecture, its length prevents it from being printed in full and therefore only key intervals are illustrated with the relevant graphs.

Table 5.2. Simulation schedule for plant dynamics

Plant dynamics	Interval of occurrence (iterations)
nominal	1 to 15000
AF1	15000 to 25000
nominal	25000 to 35000
IF	35000 to 45000
AF1	45000 to 55000
AF2	55000 to 65000

At iteration 15,000 the plant dynamics abruptly change to those of AF1. Since the GHDP controller is continuously adapted online, as soon as the dynamics are modified the weights of the three neural networks start being adjusted in order to search for a control solution for this scenario. During learning, the tracking error grows, leading to Hq_c. Concomitantly, the change in the dynamics leads also to an increase in the identification error represented by $q_i(t)$ until Hq_i is reached. In this manner, State 2 of the FDD decision logic is reached and an abrupt unknown fault is detected and correctly identified. Note that, although possible in a discrete system, both quality indexes need not reach Hq_i and Hq_c at the same iteration. For instance, in the discussed simulation Hq_c was reached before Hq_i, leading the FDD decision logic to transiently assume State 4 at iteration 15,002 before reaching State 2 at iteration 15,005. The transition through State 4 leads to a momentary misdiagnosis of the fault as an abrupt *known* fault and issues a switch command, however the fault diagnosis is corrected three iterations later and the effects of the switching are almost imperceptive since the GHDP programming has not yet had sufficient time to reconfigure from controlling the plant under nominal dynamics. Throughout the learning process, while $q_c(t)$ is high, $q_w(t)$ remains between the low and

high thresholds indication healthy levels of reconfiguration within the neural networks and therefore no controller malfunction is detected. Indeed, at 18,475 with Lq_c the GDHP controller manages to develop a satisfactory solution to the tracking problem, the FDD decision logic goes to State 3 declaring "control success" and adding the current set of weights to the DMB. Since now the supervisor possesses knowledge of the plant under AF1 dynamics, $q_i(t)$ drops below its threshold and the FDD decision logic returns to State 1 (i.e., known dynamics).

The plant returns to the nominal dynamics in an abrupt fashion at iteration 25,000, depicting a hypothetical situation in which the cause of a fault is removed from the system ceasing its effect on the dynamics. This transition differ from the one at iteration 15,000 discussed previously in the fact that the DMB has knowledge of the new dynamics and therefore can switch to the known solution as soon as it is identified. To highlight the reduction in reconfiguration time and cumulative tracking error brought by the supervisor's intervention, Figure 5.5(a) displays the trajectories of the output of a parallel simulation run without the supervisor. With only the GDHP controller striving alone to reconfigure itself, tracking performance is recovered at 1,659 iterations after the alteration in the dynamics.

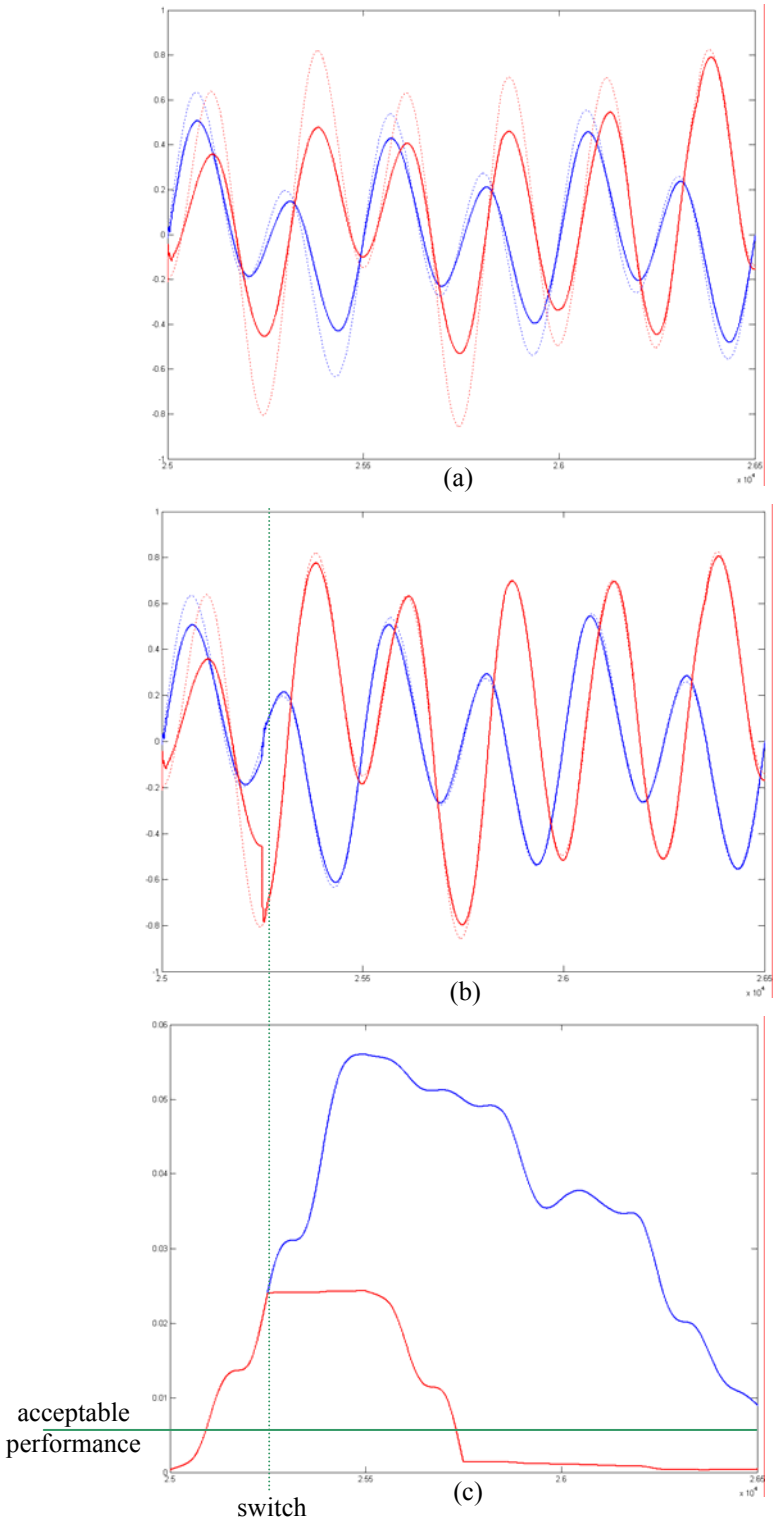


Figure 5.5. Abrupt return to nominal dynamics from AF1. Comparison of outputs $R_1(t)$ (blue) and $R_2(t)$ red) and respective desired trajectories (dotted) of simulations without (a) and with (b) supervisory intervention. Switching impact illustrated by (c), the average tracking error for the simulation without (blue) and with (red) supervisory intervention.

Figure 5.5 (b), on the other hand, shows the outcome of the main simulation in which the FDD decision logic identifies the change in the dynamics as being an abrupt known "fault" (in this case the new dynamics are of the nominal plant) at iteration 25,247 and switches the weights of all three neural networks with the ones designed for the nominal dynamics stored in the DMB since its initialization. As a consequence, the GHDP controller is able to reach the desired tracking performance at 734 iterations after the change in dynamics. In comparison, switching as determined by the supervisor led to a reduction of 55.76% in the reconfiguration time and a reduction of 74.01% in the cumulative tracking error in the first 1,500 iterations after the alteration of the dynamics as illustrated in Figure 5.5 (c).

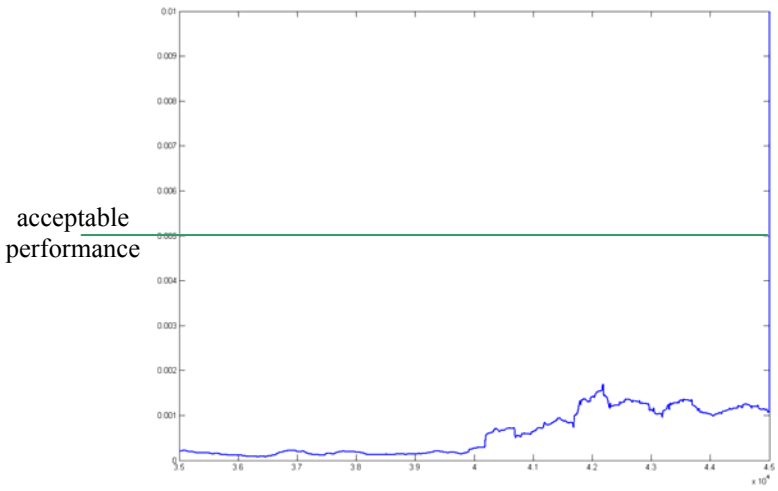


Figure 5.6. Average tracking error during IF application.

In the interval between iterations 35,000 and 45,000 IF gradually modifies the plant dynamics and is correctly identified as an incipient fault by the supervisor. The correct fault diagnosis prevents the supervisor from switching to any solutions already stored in the DMB, an action that in this case would disrupt the successful continuous

adaptation of the GDHP controller to counter the growing effects of IF. The successful maintenance of the tracking performance well below the acceptable level throughout this period is illustrated in Figure 5.6.

At iteration 45,000 the plant experiences a re-occurrence of AF1. Note that this time it transitions from IF to AF1, while in its first occurrence at 15,000 the plant was previously under the nominal dynamics. Nevertheless, the supervisor is able to correctly diagnose and identify the fault at 45,479, switch and recover tracking performance at 45,659, which represents a reduction of 72.96% over the reconfiguration time of the GDHP without supervisory intervention. Moreover, the cumulative tracking error in the first 3,000 iterations experiences a reduction of 49.34% by the switching operation. It is important to note that, different from the first discussed switching operation involving the nominal scenario, the solution to the AF1 was determined previously online by the GDHP in its first occurrence and autonomously added to the DMB. In other words, this section of the simulation illustrates that the benefits brought by the supervisor to the GDHP training extend also to re-occurring faults whose dynamics are not known during design time.

A more challenging transition occurs at 55,000 when the dynamics of the plant are abruptly modified from AF1 to AF2. In order to obtain detailed simulation results, controller malfunctions were introduced at this critical transition into the GDHP by modifying the learning rates used for the online training of the neural networks. Therefore, since the scenario in which the GDHP controller is trapped in a local minima is related to a very low gradient (based on local information), we reproduce the same effect on the training of the neural networks by drastically reducing the respective

learning rates. Similar to the presentation of the impact of the supervisor in the event of faults within the plant, Figure 5.7 shows the simulation results with (Figure 5.7 (a)) and without (Figure 5.7 (b)) supervisory intervention. Note that in order to demonstrate the impact of the supervisor alone, once reduced to generate the controller malfunction, the learning rates used for both runs (with and without supervisory intervention) are the same. As it can be seen from the simulation results, while the GDHP alone provides a poor solution to the AF2 scenario and is incapable of significantly adapting over time, maintaining a large tracking error as seen on Figure 5.7 (c). On the other hand, when the same simulation is run with the presence of the supervisor, the controller malfunction detection logic correctly determines that the online learning trapped itself in a local minima at 1,256 iterations after the introduction of AF2 and is capable of greatly reducing the tracking error over time (Figure 5.7 (c)) by intervening with a new set of initial conditions much closer to the global minima.

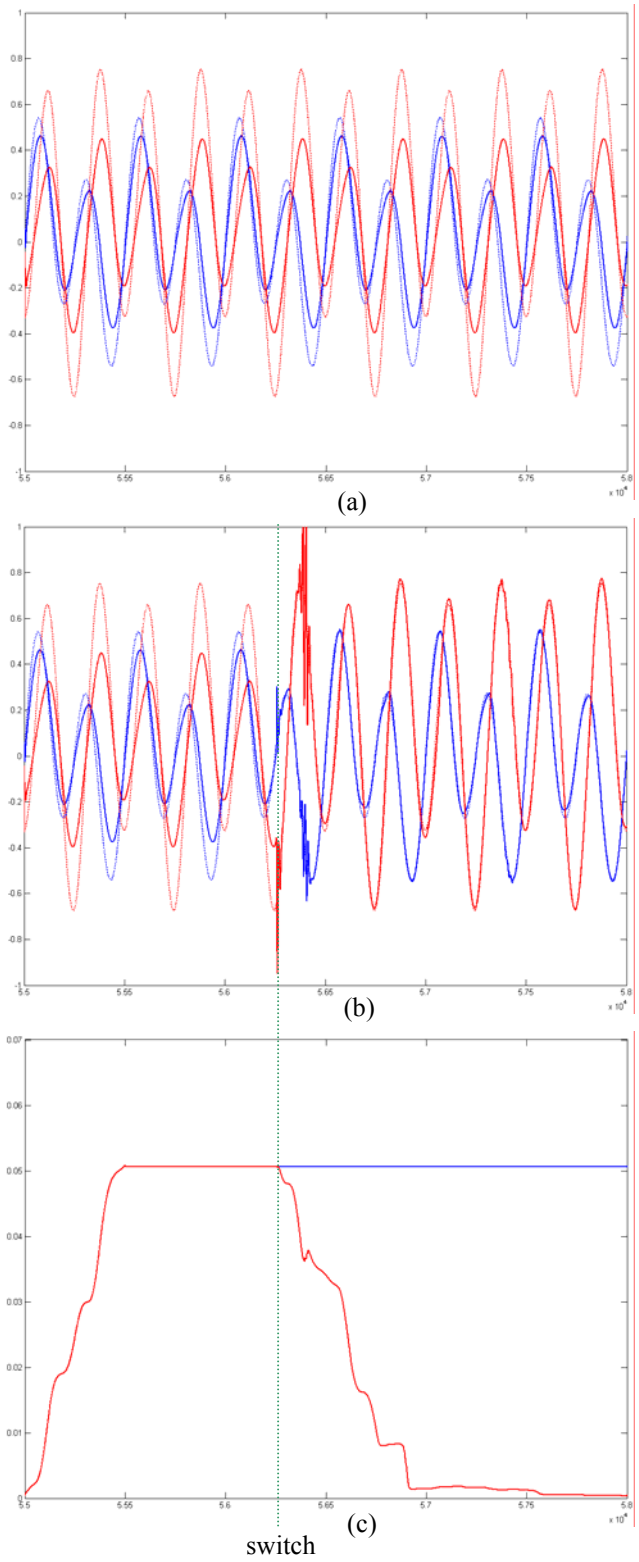


Figure 5.7. Local minima convergence, controller malfunction detection and prevention. Comparison of outputs $R_1(t)$ (blue) and $R_2(t)$ (red) and respective desired trajectories (dotted) of simulations without (a) and with (b) supervisory intervention. Graph (c) displays the average tracking error for the simulation without (blue) and with (red) supervisory intervention.

For the simulation of a controller malfunction related to training algorithm divergence we return to the same challenging transition from AF1 to AF2 at 55,000 iterations, however now all learning rates are increased by three orders of magnitude. For the gradient descent online training algorithm used here, the effects of the large learning rates are equivalent to the incongruent gradients that lead to controller divergence. Once more, to better illustrate the effects of the proposed supervisor, in a first run we deactivate it while keeping the same learning rates. By itself, the GDHP used in this simulation fails to converge to a stable control sequence. Instead, it initiates a divergent behavior that leads both outputs to values of the order of 10^3 at 42 iterations after the introduction of AF2 at which point the simulation was terminated, as shown in Figure 5.8 (a). On the other hand, in the actual simulation with the proposed supervisor activated, a controller malfunction was detected at 17 iterations after the introduction of AF2 and the supervisor was able to prevent the divergence of the online training GDHP controller by switching its weights to the dynamics witch presented the smallest identification error among those in the DMB. In this case, the weights related to the nominal dynamics were selected leading to the stable online learning of a solution for AF2 at 987 iterations after the fault occurrence as depicted in Figure 5.8 (b). Figure 5.8 (c) displays the tracking error history of both simulations as the one without supervisory intervention diverges off the scale. In conclusion, Figure 5.8 illustrates the capability of the supervisor to correctly identify controller malfunction in its early stages and act in a way that allows the controller to regain a training path that will lead to a stable solution.

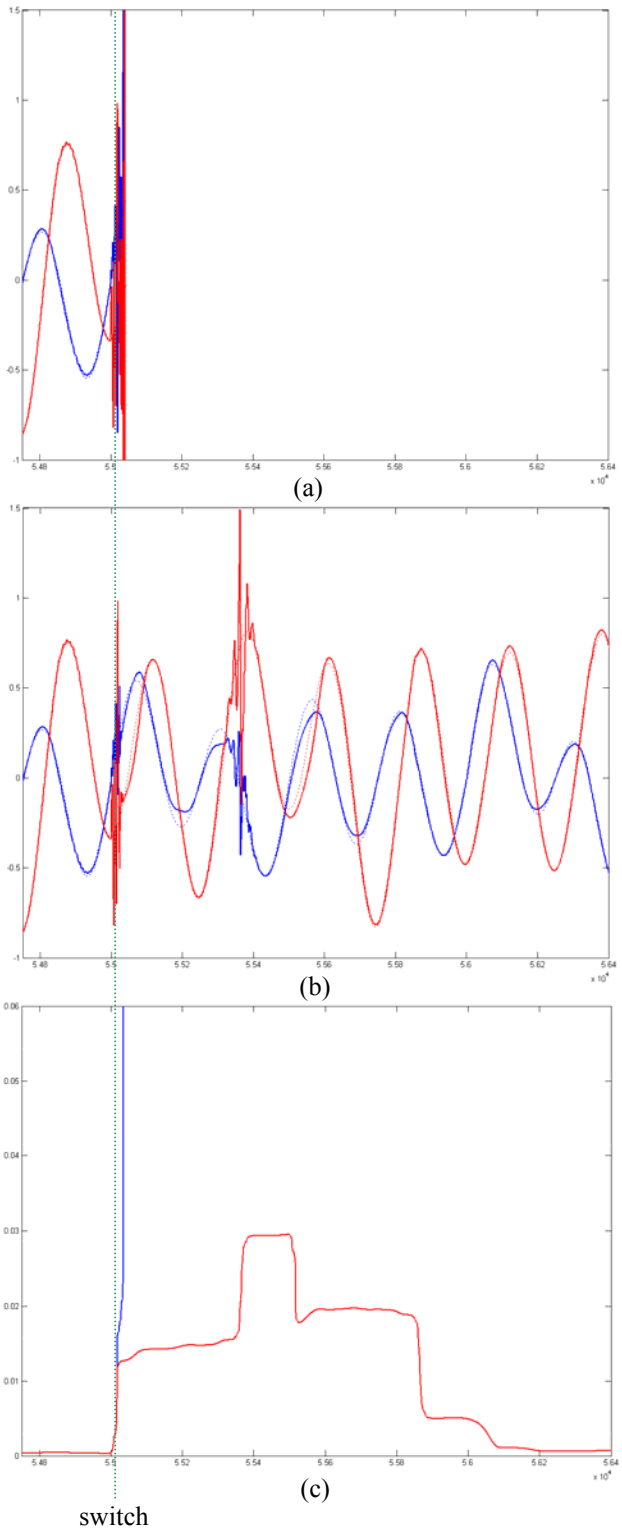


Figure 5.8. Controller divergence malfunction detection and prevention. Comparison of outputs $R_1(t)$ (blue) and $R_2(t)$ (red) and respective desired trajectories (dotted) of simulations without (a) and with (b) supervisory intervention. Graph (c) displays the average tracking error for the simulation without (blue) and with (red) supervisory intervention.

5.4. Summary

The presented work has demonstrated that the implementation of a synergistically combination of an GDHP controller and a supervisor based on three distinct quality indexes generates an efficient and reliable Fault Tolerant Control architecture. As demonstrated in the nonlinear plant simulations, the introduction of the weight quality index has made possible to distinguish between faults in the plant and controller malfunctions caused by online training divergence or local minima convergence. Further more, the Dynamic Model Bank was successfully used to generate new initial conditions to the neural network training that improve their efficacy as the supervisor autonomously acquires more nonlinear models of the plant under healthy and diverse faulty scenarios. Although the results so far have been greatly encouraging, qualitative analysis of the complete combined online stability and real-world complications is essential and will be carried out in future research. In addition, the capability of efficiently combining existing knowledge to deal with the occurrence of multiple faults is also an avenue to be explored under the presented architecture.

CHAPTER 6 – Linguistic Rule Extraction

6.1. Motivation

The collected work presented in the previous chapters amounts for a FTC solution capable of performing FDD, armed with an adaptive critic controller for the design of new solutions to unknown fault scenarios and possessing a supervisor capable of increasing the controller stability, speed and efficiency when dealing with the occurrence of faults and controller malfunctions. For a given control mission, the availability of the plant is therefore increased as we minimize the effects of faults and the reconfiguration time to apply a new control approach.

However, the occurrence of fault, although it may have its effect successfully filtered out from the point of view of the user, operator or pilot, modifies the underlying dynamics of the plant. It is safe to assume that under a fault scenario caused by the failure of a particular component (or group of components), the remaining components may be forced, during the course of following the overall control mission, to operate away from their nominal ranges. Such scenario would then lead to an increase in the probability of occurrence of other faults or the aggravation of the active fault. In the more extreme cases, event though a suitable control solution is found for a particular fault scenario, if under such fault scenario the control mission insists on particular extreme trajectories, a terminal fault for which no stabilizing controller exists may develop.

Therefore, a complete FTC solution must include the capability of providing mission planners with probabilistic predictions of fault scenarios that may affect the plant according to their choices of desired trajectories. Such probabilistic predictions should contain not only a prediction of which fault would develop, but also when it is more probable to become active. Furthermore, since such feature aims at providing humans with decision support, it is also important that such information be conveyed in a format as simplified and cognitive as possible, especially for pilot assistance. Such fault development warnings are therefore chosen to be crafted as causal rule statements using fuzzy logic. An example of such causal rules is:

IF Fault 15 (valve seal compromised) is active AND input #5 remains very low AND reference #2 remains high, **THEN** Fault 23 (loss of valve actuator) will have a 85% chance of occurrence after a delay of 30 to 45 minutes.

As an application example, if a FTC architecture manages to accommodate for the effects of a particular component fault in an army helicopter in the middle of a mission, such a feature allows mission commanders to decide whether to abort or modify such a mission given the probability of irreversible faults to develop that would lead to loss of the rotorcraft. On process industry, such a feature also has great applicability. The probability of other components braking in a process line given the previous occurrence of a particular fault and given the particular process being run supplies the operator with a key factor to assist maintenance scheduling and therefore directly impacting productivity and profitability.

6.2. Fundamental structure

When performing process management decisions, a control algorithm makes use of the knowledge of the plant in the form of a model. In most cases, this knowledge is commonly derived from first-principles and/or laboratory and pilot plant experiments; and often such "ideal" knowledge is of less than practical use under real world complications due to unaccounted factors and modeling uncertainties.

Human operators, on the other hand, make use of another type of model when in charge of process management decisions. After a long time in contact with the plant, process operators are capable of attaining some understanding of what factors govern the process and derive relationships between process variables based on intuition and past experience. This process was best described in [54] as "a cognitive skill of experienced process operators that fits the current facts about the process and enables the operators to assess process behavior and predict the effects of possible control actions." However, the knowledge attained in this fashion also presents critical deficiencies since wrong impressions on what is going on with the process will lead to operator misjudgment as documented in [55]. Furthermore, incoherencies inside such knowledge propagate itself as "mis-knowledge" or "technical folklore" are passed down from one generation of process operators to the next.

By making use of linguistic information in the form of IF/THEN logical statements or rules, Expert Systems and Fuzzy Logic Controllers (FLCs) are technologies capable of enabling better process monitoring and control. FLCs have found applications in a variety of fields such as robotics [56], automated vehicles [57] and process control [58], to name a few. Expert Systems have been used in the Chemical Process Industry

(CPI) to control, monitor [59] and understand process behaviors. Other applications of such knowledge based systems have been in operator training and for planning and scheduling of operations in control and maintenance [60], especially for getting a plant back online after a failure or abnormal operating condition.

Expert Systems can be built from knowledge inserted by human experts or acquired from historic data from the system. Knowledge bases made by polling information from experienced personnel not only incorporate the before mentioned "technical folklore", but also is intrinsically incomplete. Such rules pertain only to information that is critical or obvious to the operators; it is related to information just necessary for them to maintain desired plant conditions. Such information does not incorporate the knowledge of events that are lesser in significance or rarer in occurrence, but which affect the operation of the plant nonetheless. A complete rule base should possess information on almost all plant events that have an effect on the desired output or may change the variable under control. Finally, knowledge collected from experts is usually in the form of static rules loosely related to the real numerical world [61]. Due to its lack of a mechanism to deal with the temporal behavior of the process, the rigid, non-adaptive knowledge devised in this fashion becomes inadequate for complete supervisory control of dynamic systems. Therefore, the solution lies on the development of an algorithm capable of autonomously generating and improving a dynamic rule set for an expert system directly from process data.

It is fundamental for the modeling of a dynamic system that the model used incorporates the concept of time. Based on the widely applied Autoregressive Moving Average (ARMA) [62] models, [63] proposed to incorporate temporal relationships into

fuzzy rules by matching antecedents with consequents a fixed number of time steps in the future. In [64] the architecture was extended to allow different discrete time delays to be used for each antecedents in single consequent rules. Due to the usage of discrete time delays however, the representation capability of the rule set was largely affected by the particular choice of time delays and it displayed great sensitivity to noise, especially related to datasets composed of data sampled from continuous systems. Displaying applications related to the stock market and the weather, [65] applied stochastic pre-processing techniques to improve the meaningfulness of the data provided to the rule extraction mechanism. Based on the concept of internal clocks that biological organisms use for the learning of period and interval timing, [66] proposed the usage of a temporal membership function for the averaging of sampled data in order to generate crisp values related to fuzzy time periods. Applications of such approach have been documented in distributed adaptive routing control in packet switched communication networks [67]. Therefore, in the proposed paper, a particular fuzzy delay is assigned to the temporally averaged consequent of each rule, generating a structure of the form:

IF *condition 1* ***AND*** *condition 2* ***AND*** *condition 3…* ***THEN*** *after a certain fuzzy delay, a control variable will be such*.

The statement between the IF and the THEN conjunction is the *antecedent* while the statement after the THEN conjunction is the *consequent*.

A crucial step in the autonomous extraction of rules is the method used to validate and compare those that are created. An optimal rule should be accurate, properly describe the dynamic relationship between its antecedent and consequent, and possess enough data

to support it. In a methodology introduced in [68], three metrics based on a Truth Space Diagram (TSD) capable of encapsulating and measuring each of these three goals were introduced and tested. However, it was also shown that in the general scenario such metrics cannot be independently optimized due to inherent conflict among them.

Multiple Objective Evolutionary Algorithm (MOEA) is a tool capable of performing efficient searches on high dimensional spaces to locate the Pareto front, a set of solutions that contain the best rule for each possible tradeoff between conflicting goals. A growing research field, MOEA has already demonstrated successful applications in solving challenging benchmark problems [69] and real world applications [70].

In the presented work, three metrics developed in [68] are used under a novel dynamic treatment of the data to evaluate linguistic rules against process data. MOEA is then introduced to locate inside the high dimensional rule space the Pareto front of the antecedents that best describe (in the sense of different combination of metrics) a given consequent.

6.3. Rule evaluation

In order to provide automate mission planning decision support, a rule must display three basic characteristics: high accuracy, precise antecedent/consequent relationship, and sufficient data support. However, it is seldom possible to maximize these three characteristics at the same time. For example, if an event is observed only a single time, it is trivial to develop a rule with 100% accuracy, however it will lack support from historical data and the probability that it will describe a whole family of

similar events with comparative accuracy is small. For this reason there is a need to develop three qualitative metrics, each focusing on one of such competing characteristics.

In [68] a series of metrics were suggested, each capable of specifically representing a different quality of a rule. All metrics were designed having as a core concept an innovative rule representation denominated the Truth Space Diagram (TSD). In the first part of this section, the efficiency of the TSD is further enhanced with the introduction of a novel pre-processing strategy for the representation of the temporal behavior of the plant into the linguistic rules. The concept of the TSD is then introduced taking into consideration the enhanced temporal representation and finally the three metrics of concern are introduced.

6.3.1. Data pre-processing

In previous applications of the TSD methodology [68], data pre-processing took place in the following manner: 1) The consequent data was shifted backwards in time by fixed intervals so that each set of antecedents matched three consequent values corresponding to short, medium and long delays; 2) The crisp input-output data was fuzzified. Fuzzification proceeded through the application of Equation (6.1) by using triangular membership functions to classify each variable into three fuzzy categories – low, medium and high:

$$\mu_x^{i,j} = \begin{cases} \dfrac{a_j - x_i}{a_j - b_j}, & x_i \in \left[a_j, b_j\right] \\ 0, & \text{otherwise} \end{cases}, \tag{6.1}$$

where j=1 to 3, i=1 to n, x_i is the crisp numerical value of the i^{th} input or output variable, $\mu_x^{i,j}$ represents the fuzzy membership value of x_i in the j^{th} fuzzy category, a_j and b_j are the fuzzy set break points for category j, and n is the maximum number of datasets in the input-output data. Figure 6.1 illustrates the fuzzy classification of one variable into three fuzzy categories.

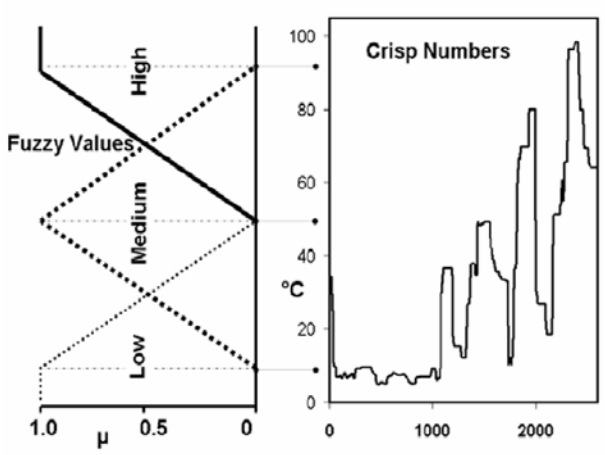


Figure 6.1. Fuzzy classification procedure for the antecedents. In this case, temperature with centers at 10, 50 and 90°C.

The fuzzification of the physical crisp data in such manner leads to great generalization capabilities, inherent noise rejection, and direct rule interpretation by human operators. Although ideal for the treatment of the antecedents, the manner through which the dynamic temporal element was incorporated into the consequent lacked such benefits. In essence, consequences for plant characteristics at any given time were only observed at discrete instants. Since a consequent's time delay contains variability as much as physical characteristics of a system, although rule extraction was possible, it required extensive data in order to determine the true correlations through time. Moreover, the previous approach relied on *accurate* knowledge on the inherent major

delays of the plant in order to set up the number of iterations that correspond exactly to the operator's understanding of small, medium and long delays.

In the present work, such deficiencies are addressed by dealing with time uncertainties in a novel manner that is different in essence from the fuzzification of physical variables. The application of such approach leads to a meaningful linguistic description that maintains all the previously stated benefits while better capturing the temporal characteristics of dynamic plants. Instead of simply shifting the data to obtain a single measurement to represent a delay, averaged values of a consequent are obtained for each fuzzy delay region trough Equation (6.2) and it is those averaged values that are then classified into the membership function of the consequent by Equation (6.3).

$$y_{\delta}^{i}(t) = \frac{\sum_{k=t_{\delta}^{1}}^{t_{\delta}^{3}} \mathrm{M}_{\delta}(k) \cdot y^{i}(t+k)}{\sum_{k=t_{\delta}^{1}}^{t_{\delta}^{3}} \mathrm{M}_{\delta}(k)}, \tag{6.2}$$

$$\mu_{y,\delta}^{i,j}(t) = \frac{a_j - y_{\delta}^{i}(t)}{a_j - b_j}, \tag{6.3}$$

where $y^{i}(t)$ is the crisp measurement of the i^{th} consequent at time t, $y_{\delta}^{i}(t)$ corresponds to its arithmetical average for a given fuzzy delay δ, $\delta \in$ [short , medium , long], t_{δ}^{i} are the fuzzy set break points ($i \in [1, 2, 3]$) for category δ, $\mathrm{M}_{\delta}(t)$ denotes the membership function of a fuzzy delay δ, and $\mu_{y,\delta}^{i,j}(t)$ is the fuzzy membership value of the i^{th} consequent for a fuzzy delay δ. An example of the application of the procedure involving Equations (2) and (3) can be seen in Figure 6.2, where the fuzzy membership

value of the consequent $y^1(t)$ is calculated at time 20s for *medium* delay and *low* temperature, i.e. $\mu_{y,m}^{1,l}(20)$.

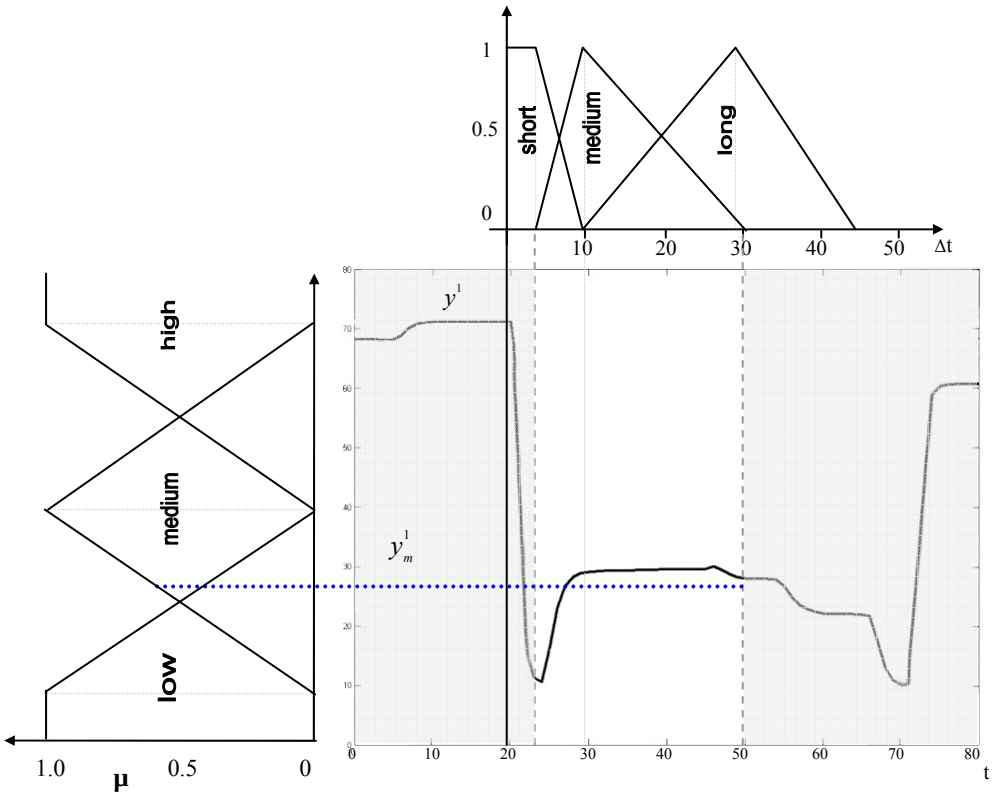


Figure 6.2. Proposed physical and temporal two-step fuzzification procedure. The figure displays a fuzzification example in which at simulation time 20 the output y^1 is evaluated for medium delay.

By processing the available data from the antecedents using Equation (6.1) and that from the consequents with Equations (6.2-6.3), crisp data is translated into linguistic variables. It is important to note that although each antecedent relates to a single linguistic variable, due to the introduction of the fuzzy delay, each consequent is represented by three fuzzy variables, each related to a different delay membership function.

6.3.2. The Truth Space Diagram

The TSD is a two-dimensional space in which a series of metrics capable of quantifying the quality of a particular cause-and-effect rule can be obtained. Each TSD relates to a single rule. For every data point extracted either from mathematical simulations, pilot plant experiments, or real-word sensor data, a point is plotted in the TSD according to its truth of the antecedent *Ta* and the truth of the consequent *Tc*. Both parameters are calculated as geometrical means of the fuzzy membership function of each variable of the antecedents and consequents. Hence, the truth space delimited by *Ta* and *Tc* is bounded between 0 and 1 in which a value equal to 0 means absolute false while a value of 1 means absolute true.

Designed in this fashion, the TSD represents a one-to-one mapping from the dataset from the real (numerical) space to a new (truth) space defined by the linguistic statements of a specific rule. The TSD can be divided into four quadrants and each quadrant provides different information about the linguistic rule. For example, consider point A in Figure 6.3. The values for *Ta* and *Tc* are high for this data point, i.e. the predicted consequent follows the appointed antecedent or the cause and effect match according to the relevant rule statement. This reveals that the information expressed in the linguistic rule is contained within the numerical data. Hence, many points in Quadrant II of the TSD reflect the validity of the rule in question. Consequentially, points in Quadrant IV show that the rule statement is false, i.e. what the antecedent of the rule express does not lead, in most cases, to the predicted consequent. An example of this can be seen from data point B in Figure 6.4. Similarly, points in Quadrant I demonstrate the incompleteness of the rule, since the predicted consequent was due to an event(s) other

than the one expressed in the antecedents of the rule. Finally, the presence of a cluster of points in Quadrant III show the possibility of that a rule is valid, however the amount of data currently available does not allow yet for a conclusion to be drawn with enough confidence. The points that lie on the vertical and horizontal axis show that either the antecedent or the consequent of a particular rule were not expressed in the data.

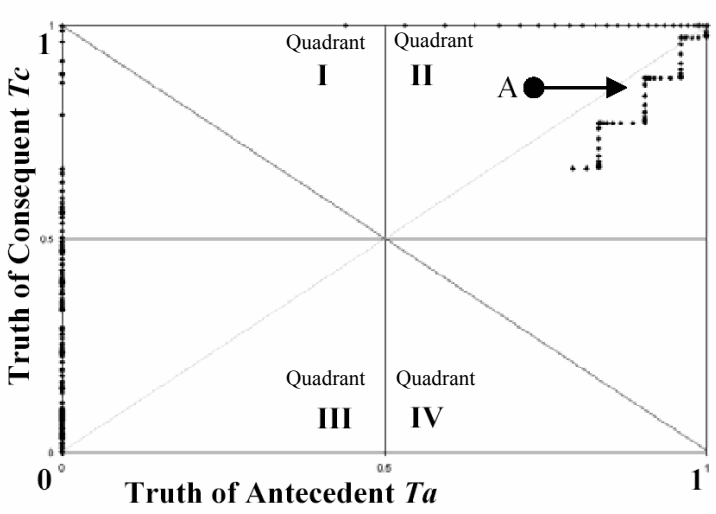


Figure 6.3. TSD for a meaningful rule extracted from process data with sufficient supporting evidence.

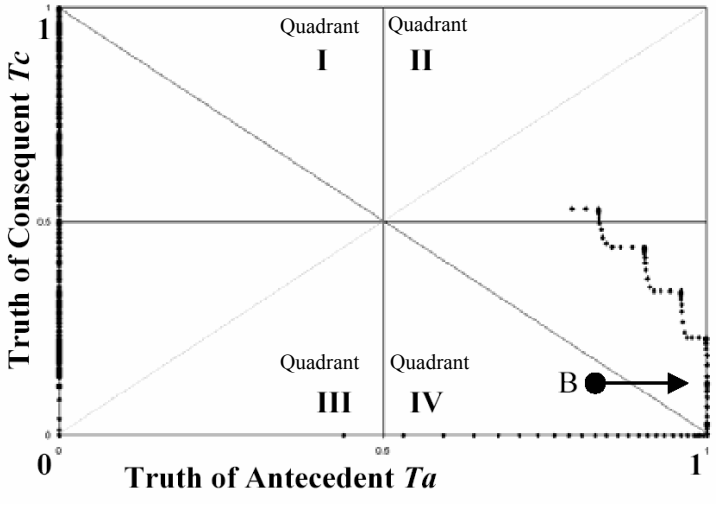


Figure 6.4. TSD for a rule that was proven inaccurate in a significant number of points in the process data.

6.3.3. Numerical metrics

As mentioned previously, the goal of the presented work is to extract rules that present high accuracy, precise antecedent/consequent relationship, and that are supported by sufficient data. By using the TSD, it is possible to obtain metrics for each of these conflicting goals. In order to transform the problem into one of minimization however, the actual metrics of interest are converted into: rule inaccuracy, antecedent/consequent mismatch, and lack of supporting evidence in the dataset. To improve the performance of the rule extraction algorithm, all metrics presented here are normalized to the interval [0,1].

Metric 1: rule inaccuracy – A rule is deemed inaccurate when its antecedent is observed but the consequence that follows after the prescribed delay does not match the predicted behavior. As mentioned previously, in the TSD this concept relates to the points in the Quadrant IV (i.e., set Q4), which relate to high truth of the antecedent but low truth of the consequent. The number of data points in Q4 (i.e., n_4) can therefore be used as a relative measure of inaccuracy, however it is necessary to normalize this number by dividing n_4 by the total number of data points in which the rule antecedents were observed with sufficient confidence, i.e. the sum of the points in Quadrant II (n_2) and in Quadrant IV (n_4). Equation (6.4) summarizes m_1, the rule inaccuracy metric.

$$m_1 = \begin{cases} \dfrac{n_4}{n_2 + n_4}, & n_2 + n_4 > 0 \\ 1, & \text{otherwise} \end{cases}. \tag{6.4}$$

Metric 2: antecedent/consequent mismatch – from a good rule it is expected that the value of *Tc* should match the value of the *Ta*. In other words, the intensity in which the antecedents are observed should be equal to the intensity of the resulted consequent. In real world scenarios however, *Tc* is affected by both the quality of the rule and the quality of the available data (e.g., noise corruption). By analyzing the data points in Quadrant II (i.e., set Q2), it is possible to measure antecedent/consequent mismatch directly by summing all distances from each data point in it to the diagonal of the TSD. Defining Ta_i and Tc_i respectively as the truth of the antecedent and the truth consequent for rule i, $\|.\|$ as the Euclidian norm, and since 0.3536 is the maximum distance to the diagonal, this second metric is stated as shown in Equation (6.5):

$$m_2 = \begin{cases} \dfrac{\sum_{i \in \mathrm{Q2}} \|Ta_i - Tc_i\|}{0.3536 \cdot n_2}, & n_2 > 0 \\ 1, & \text{otherwise} \end{cases} . \tag{6.5}$$

Metric 3: lack of supporting evidence in the dataset – Since there is a need for sufficient information inside the available data for any conclusion to be drawn, this metric is crucial for the success of any data driven rule extraction method. Using the TSD representation, Equation (6.6) is built for this purpose.

$$m_3 = 1 - \frac{n_{TSD}}{n_{data}}, \tag{6.6}$$

where n_{TSD} is the total number of data points mapped in the TSD excluding points on the abscissa ($Ta = 0$); and n_{data} is the total number of data points available in the dataset.

6.4. Rule extraction

Evolutionary Algorithms (EA) is commonly regarded as a family of stochastic search procedures that is inspired by computational models of natural evolutionary processes to develop computer based optimization problem solving systems [71]. Being a population based algorithm, in EA each candidate solution is represented as an individual. When evolving towards better solutions, the individuals that better meet the optimization goal (individuals with greater *fitness*) have a greater probability of being selected to take part in the creation of the individuals of the new generation.

For problems that have multiple conflicting goals that cannot be directly combined into a single scalar measure of fitness, Multiple Objective Evolutionary Algorithm (MOEA) provides a method through which a population of solutions can evolve towards a set of solutions within which no solution is better than another in all optimization goals. By defining that an individual *dominates* another when at least one optimization goal is closer to the ideal values and all others are equal or closer to it, than those of the other individual, such a set can be referred to as the non-dominant set. The non-dominant set among all possible solutions is called the Pareto front and its determination is then the ultimate goal of MOEA. Therefore, as shown in Equation (6.7), in MOEA the fitness F is a vector of the optimization goals, in this case represented by the three goodness metrics.

$$F = [m_1, m_2, m_3]. \tag{6.7}$$

In the presented work, MOEA in the form presented in [69] is used once for every consequent to evolve an initial random population of related rules towards the Pareto front of the tri-dimensional space defined by *F*. Therefore, for each consequent, a set of equally good (in the sense of the minimization of the three previously defined metrics) antecedents is extracted based on their relative success.

Table 6.1. MOEA pseudocode

1. Generate initial population;
2. Evaluate the metrics of all individuals and rank them;
3. for (*i*= 1: *maximum_generation*)
4. Choose parents with probability inversely proportional to their ranks;
5. Perform crossover operation on parents to generate new individuals;
6. With probability equal to the mutation rate, perform mutation procedure;
7. Evaluate all three metrics on the new individuals;
8. Update population ranking.

As laid down in the pseudocode in Table 6.1, the first step of implementing an MOEA algorithm is the generation of the initial population of candidate solutions. In order to guarantee an unbiased and diverse population while maintaining a low computational demand, 20 initial individuals are generated with random antecedent values, clearly 20 is an *ad hoc* choice that needs to be quantified in future research. In the following step, the three metrics are calculated for the rules formed by the antecedents of

each individual and the consequent related to the current MOEA run. It is also in this step that each individual is assigned a rank value according to their relative success in minimizing the elements of the fitness vector F (i.e., the concept of Pareto optimality). In particular, the ranking scheme discussed in [72] is implemented, in which an individual is assigned a rank value equal to one plus the number of individuals it is dominated by.

The third step in the presented pseudocode is the first in its main loop and it relates to the selection of two individuals that will be involved in the generation of new individuals to the population. The selection is performed stochastically by assigning a greater selection probability to individuals with smaller rank values, and therefore individuals with greater fitness. The individuals in this way chosen are denominated parents and, in Step 4, part of their individual solutions are exchanged in the operation termed crossover. Through the crossover operation, two new individuals (solutions) are formed, combining elements of both parents. In the following step, mutation, another biologically inspired process, may affect with a specific probability (defined as the mutation rate) the newly generated individuals. In MOEA, mutation takes place by randomly modifying an arbitrary portion of the solution related to a given individual. Independent of the occurrence of mutation in Step 5, on Step 6 the fitness vector F is evaluated for the two new individuals, followed by the updating of the ranks of all individuals in the population. A generation is then concluded and the algorithm returns to Step 3 until a maximum number of generations is reached.

After MOEA generates a set of non-dominant rules for each consequent, thresholds are used over each metric to eliminate outliers and establish minimum acceptable performances (e.g., minimum degree of accuracy required of a rule). Another

post-MOEA data processing involves removal of time-redundant information from the rule set. If, for instance, a consequent should develop quickly and remain unchanged for a long time throughout the dataset, the antecedents would be credited with both short and long-term effects even though the long-term effect is only a matter of persistence. Time redundancy then refers to rules with equal antecedents and physical consequents, but with different consequent delays. In such cases, the rule related to the longer consequent delay is removed.

6.5. Simulation results and discussion

To demonstrate the process of extracting temporal cause and effect relationships, data was acquired from a Hot and Cold water simulator shown in Figure 6.5. The simulator incorporates real world dynamics such as transport and measurement delays and is capable of adding deviations such as measurement bias and process drifts that have an ARMA stochastic behavior, noise and valve "sticktion". The simulation was nonlinear, had multiple inputs and its dynamics (such as hydrodynamic delay) depended upon operation conditions. For the purpose of generating data, the four input variables were manipulated, the flow of each input tube (F_1 and F_2) changing randomly at every 20 seconds and the input temperatures (T_1 and T_2) changing randomly at every 40 seconds. The periods of manipulation of the variables were shifted so as not to lead into two changes occurring at the same time. Their effect on the temperature at the output of the mixer stream (T_3) was measured over time. All flow variables were restricted to the interval [0,30] kg/min and the temperature variables to [0,100] ^{o}C.

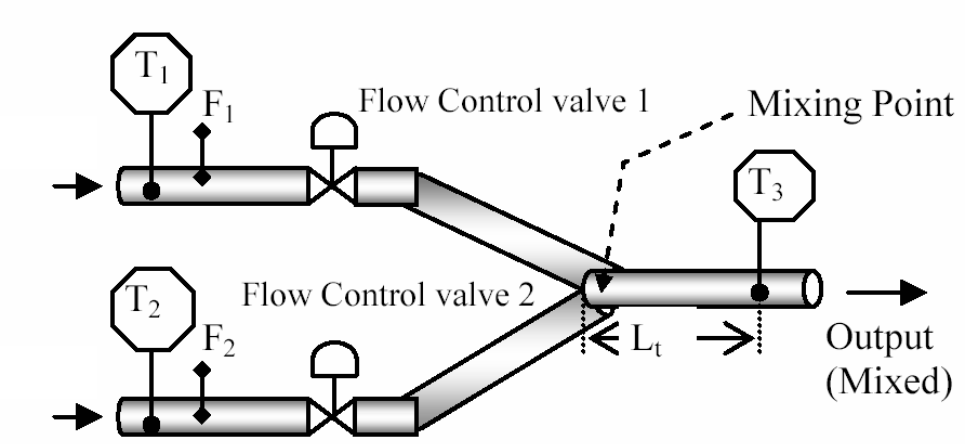


Figure 6.5. The hot and cold water simulator used for validation of the rule extraction algorithm.

According to the proposed data pre-processing procedure, physical variables were fuzzified with centers at 10, 50 and 90 ^{o}C for the temperatures and at 2.5, 15 and 25 kg/min for low, medium and high flow rates respectively. For the fuzzy delay, centers were placed at 3, 7 and 20 seconds for short, medium and long delays respectively. Note that long delay rules will be harder to calculate since the input variables will change randomly at faster rates. The dataset is intentionally devised in this form to challenge the rule extraction procedure with data of different degrees of quality.

The proposed MOEA based on the three selected metrics was implemented over the pre-processed data generating a non-dominant set of rule candidates for each consequent. An initial population of 20 individuals was allowed to evolve through the course of 200 maximum generations. The individuals received a rank equal to one plus the number of individuals it was dominated by. At each generation, parents were chosen according to a probability inversely proportional to their rank. For the generation of new individuals, crossover was implemented with a single crossover point and a mutation rate of 0.01 was used. An elitism scheme was implemented to guarantee that all best solution candidates were preserved during the evolution process. Figure 6.6 displays the obtained

non-dominant set containing 13 antecedent combinations relating to high temperature at T_3 after a long delay.

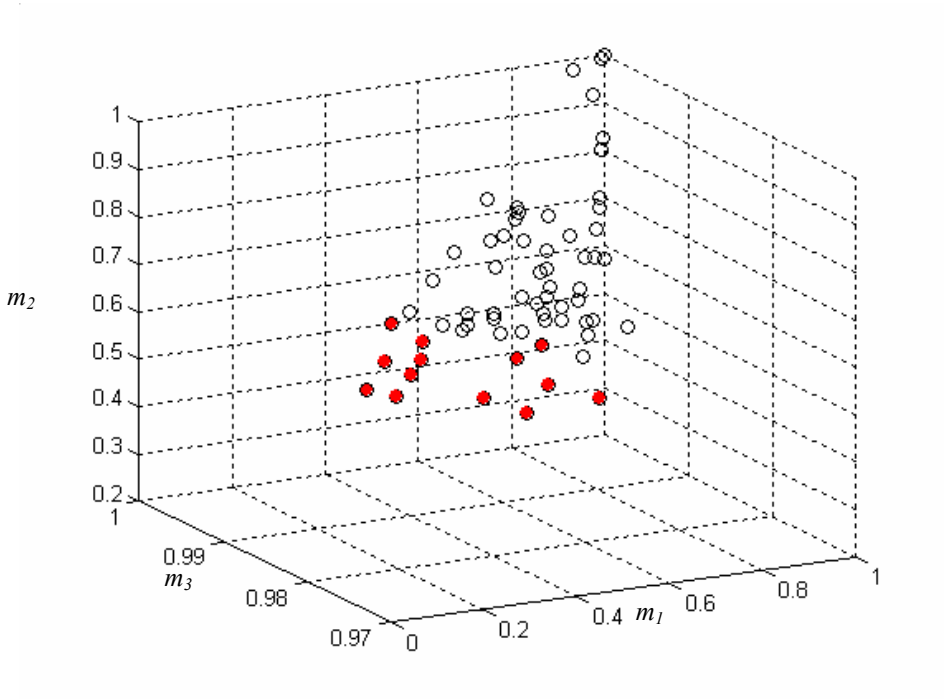


Figure 6.6. Distribution of individuals related to a single consequent in the metric space at generation 200. Filled circles form the non-dominant set.

For post-processing, the minimum acceptable accuracy of the extracted rules was set at 90%, a minimum of 1% of the information inside the observed dataset was necessary to validate a rule, and a maximum spread of 0.6 around the diagonal of the TSD was allowed. In terms of the minimization metrics m_1, m_2, and m_3, the corresponding thresholds were 0.1, 0.99 and 0.6 respectively. Finally, rules that were time-redundant were removed to generate the final rule set.

Through the outlined process, the presented algorithm was capable of extracting 49 rules out of a possible set of 729 rule combinations (containing both "good" and "bad" rules). Some examples of the obtained rules are shown below:

- IF T_1 is high AND F_1 is medium AND T_2 is medium AND F_2 is low THEN after a medium delay T_3 will be high.

- IF T_1 is low AND F_1 is low AND T_2 is low AND F_2 is low THEN after a long delay T_3 will be low.

- IF T_1 is medium AND F_1 is low AND T_2 is high AND F_1 is high THEN after a short delay T_3 will be high.

Good rules are those that express the phenomenologically-based, cause-and-effect mechanism as a logical relation between their antecedent and consequent parts. Consequentially, bad rules are defined as those that are inconsistent with the process phenomena. Therefore, in order to evaluate the quality of the 49 extracted rules, each one of them had its antecedents implemented in the simulator and those that demonstrated matching consequents were deemed good rules. As a result, 5 of those rules were rejected demonstrating a success ratio of 89.8% of the proposed rule extraction algorithm. Moreover, most rejected rules pointed to borderline consequents (e.g., the measured T_3 would be 78^{o}C, when the maximum value acceptable for a medium fuzzy range was 70^{o}C). Such scenarios reflect the choice of fuzzy membership function centers, left at the discretion of the operator.

As mentioned previously, any rule extraction procedure can only produce results as good as the data provided. This simulation was intentionally designed to provide much sparser and more noise corrupted data for the extraction of rules related to long delays. Among the 44 good rules in the final set, only 6 of those portrayed long delays, while the expected from a fully representative dataset would be one third of the total. Since the

algorithm minimizes inaccuracy (m_1) while at the same time evaluating the amount of supporting evidence (m_3), the lower number of long delay rules extracted demonstrates the success of the procedure in avoiding unsupported rules to be presented to the operator in the final set.

6.6. Summary

As demonstrated though the application of the procedure on the data collected from the simulated hot and cold water mixer, the proposed rule extraction procedure succeeded in autonomously generating a viable rule set from a less than completely representative data set. The use of MOEA as an optimization algorithm allowed for three conflicting metrics to be evaluated simultaneously leading to the final extraction of optimal non-dominant rule sets. Both pre-processing, involving the representation of each rule inside a TSD, and post-processing, which allowed for the removal of time-redundant rules, were applied successfully and with beneficial outcomes. The presented work represents the foundations of an ultimate goal to achieve rules capable of describing the probability of a fault to occur after a certain time range given the choice of desired trajectories and their related plant input signals. Integration with the FDD/CMD Supervisor is also a crucial step to be taken.

CHAPTER 7 – Description, Initialization and Tuning of 12 FTC Design Parameters

7.1. Introduction

Every control problem has a set of goals and constrains specific to itself that affect the choice of the control paradigm to be applied and the actual implementation the chosen approach. Maximum overshot, maximum acceptable reference tracking error and actuator saturation limits are common example of such peculiarities that in the practice of control systems translate to controller design parameters. Even though not all control design approaches are capable of converting all forms of real world specifications directly into design parameters, most will at least recognize their importance and discuss how different design decisions affect such particular points of interest. For instance, even though we might know at design time the saturation limits of an actuator, there is no readily available design parameter in classical PID control that will lead directly to a design of a controller that will not overstep such boundaries during the course of its operation. However, if after implementation the actuator saturation is interfering in a significant way with the main control goals, there are guidelines of how to modify the existing control parameters in order to generally reduce the chance that such limits are reached.

The design process of a fault tolerant controller also starts from gathering a list of control goals, specifications and constrains and then translating those into design parameters that will tune each fault tolerant controller to its particular application. Specifications related to FDD include fault detection delay [73], AKF identification dealy [74], and maximum false alarm and miss-diagnosis rates [75]. Examples of specifications related to the controller response to a fault also exist in the literature, including maximum controller reconfiguration delay [76], maximum acceptable performance loss under a fault scenario [77], maximum increase in control effort under a fault scenario [78] and maximum overshot during reconfiguration [3]. However, two major problems can be seen in the field nowadays. First, no single approach in the field has provided enough design flexibility to cover a significant number of FTC specifications, nor clearly indicated how particular design choices affect each and every one of those. Second, to the best of our knowledge, no FTC approach has provided any design procedure through which its parameters can be adjusted to achieve a specific set of FTC goals.

In this chapter we aim to address the two previously stated deficiencies in the proposed FTC architecture. To accomplish such, the first step is to modify the quality indices used by the FTC supervisor in order to improve their capability to represent the various aspects related to FDD and the switching and learning operations governed by the supervisor's two decision logics. Such modifications then result in the creation of 12 FTC design parameters, which lead to sufficient design flexibility to affect a series of FTC goals, such as maximum fault detection delay and maximum acceptable tracking error under a fault scenario.

After detailed description of the effect of each design parameter in twenty six different aspects of the final FTC response, an offline procedure for the determination of values for all 12 design parameters is presented. Since the proposed FTC approach is designed to deal with AUFs, there is not sufficient information at design time to guarantee that the offline determined design parameters will fulfill all FTC specifications once applied in the real world. Instead, the offline parameter design determination procedure provides suitable initial conditions and a table look-up method is proposed for efficient parameter tuning during actual implementation.

This chapter is divided as follows. Section 7.2 introduces extended versions of all three quality indexes, each offering 4 design parameters, and details how modifications in each design parameter affect the final FTC response. Section 7.3 presents the proposed offline procedure for the generation of suitable initial conditions for the FTC design parameters and describes the online tuning procedure. Section 7.4 then presents the results of the application of the proposed methodology to fulfill the FTC specifications os a simulated plant subject to nonlinear AUFs. This chapter is closed by final conclusions in Section 7.5.

7.2. Extended Quality Indexes

In its original formulation the supervisor gathered the necessary information from the plant in order to perform FDD through two distinct quality indexes, termed controller and identifier quality indexes. In a subsequent chapter, a third quality index, the weight quality index, was introduced in order to gather information on the health of the

adaptation process an in this manner provide advanced information fundamental to CMD. Although successful in performing their duties as stated in their respective chapters, the quality indexes in their original formulation lack the flexibility to address the FTC application concerns raised in the previous section of this chapter. Therefore, before discussing how to adjust our FTC architecture to the requirements of each FTC application, modifications to all three quality indexes will be introduced. Such modifications do not alter their nature, but increase the number of design parameters to make it possible for the user to, for example, adjust how fast AUF are classified without affecting the classification speed of AKF.

This section discusses the particular effect of each of the proposed design parameters on the ultimate response of the proposed FTC solution. The reader is assumed to be familiar with the role of each quality index and the workings of both FDD and CMD decision logics presented on earlier chapters.

7.2.1. Identification quality index

Focusing first on the identification quality index, one of the key aspects of its new formulation is that it contains a set of two distinct filter parameters, applied to the filter depending if the measured identification error is greater (γ_i^u) or smaller (γ_i^d) than the value of $q_i(t\text{-}1)$. Also, instead of using a single threshold to create the logic regions of high and low values of $q_i(t)$, two distinct thresholds are used: the Hq_i that signals that the quality index is high after it goes beyond it, and Lq_i that signals that the quality index is low after it goes below it. The use of the two threshold levels not only allows for different

responses to be adjusted, but also creates a hysteresis region that contributes to the index noise rejection. Equation (7.1) brings the new formulation in its discrete form.

$$q_i(t) = \gamma_i \min_{m \in M} \left\| \hat{R}^m(t) - R(t) \right\| + (1-\gamma_i) q_i(t-1), \tag{7.1}$$

where M is the total number of models present in the DMB at time t, $R(t)$ is the output of the plant, $\hat{R}^m(t)$ is the output of the plant predicted by the DMB model m, the norm $\|\cdot\|$ is defined as the sum of the absolute value of all vector elements, and γ_i assumes the value of γ_i^d if $\min_{m \in M} \left\| \hat{R}^m(t) - R(t) \right\| \leq q_i(t-1)$ or γ_i^u otherwise.

In this manner, the identification quality index presents the user with four design parameters that can be adjusted independently in order to modify the response of the fault tolerant controller to match its goals in a given application. The ultimate adjustment of each design parameter is the subject of the next section, but first it is important to explore how changes in each one of the identification design parameters affects the response of the fault tolerant controller as a whole. For instance, since γ_i^u is used when the identification error over all models in the DMB is increasing, using a greater value leads to faster AUF classification. On the other hand, by decreasing its value we obtain greater noise rejection since (7.1) acts as a low pass filter on the identification error, which is directly affected by the quality of the plant's output signal. Moreover, by decreasing γ_i^u, significant identification error is required to be present for a greater length in time in order to substantially impact $q_i(t)$ decreasing the chance of AKFs being transitorily misclassified as AUFs in the interactions immediately after the occurrence of a fault before a sufficient amount of input-output data is collected.

The effects of the other quality index filter parameter γ^{id}, on the other hand, are different. Since it dictates how fast a low identification error can affect $q_i(t)$, increasing it leads to faster AKF classification. However, decreasing its value reduces the chance of AKF misclassification before a significant number of input-output data points are collected. This point is particularly interesting because it is conceivable that certain fault scenarios only demonstrate a significant impact on the plant dynamics at very specific regions of the state space and therefore it is important not to draw precocious classifications before at least one reference cycle is observed in full. On the other hand, although the length of time of a reference cycle is known, as the plant dynamics change abruptly due to a fault, there are no guarantees on the section of the state space the plant will cover during the controller adaptation process. Moreover, adjusting the filter parameter only alters the rate of decay. The actual time that the quality index $q_i(t)$ takes to assume a low logic value is a function of Lq_i value as well as the actual identification error history.

The limit over which values of $q_i(t)$ are considered high is determined by the Hq_i threshold. Increasing its value leads to higher noise rejection, but different than the decrease of γ_i^u which leads to the rejection of high frequency noise, higher values of Hq_i reject noise based on its amplitude. Increasing the value of this design parameter also decreases the chance of transitory misclassification of AKFs as AUFs caused by peaks in $q_i(t)$ after the occurrence of a fault and before a model with low identification error can be fully expressed. On the other hand, decreasing its value also has two beneficial effects. First, a lower Hq_i value leads to faster classification of AUFs. Second, it also causes a

reduction on the chance of transitory and permanent misclassification of AUFs as variations of AKFs already stored in the DMB.

Finally, we focus on the fourth identification design parameter: the lower $q_i(t)$ threshold Lq_i. Increasing its value leads to faster AKF classification since the threshold will be reached sooner as $q_i(t)$ decreases. Also, an increase in its value leads to an expansion on the region that accounts for the variability of each particular fault scenario. If this threshold is made too small, two instances of the same fault, an increase in the friction of an actuator joint for example, can be classified as two separate faults, each with their own model and control solution in the DMB, if the friction coefficient were to differ slightly. Situations such as these are undesirable since the control solutions are close enough from each other that the difference can be quickly dealt with by the underlying adaptive controller, but the DMB would be incapable of providing closed initial conditions through switching because the second occurrence would be classified as an AUF. Moreover, the high specificity of the models within the DMB cause by too low Lq_i causes a greater number of models to be added to the DMB, which may result into serious memory and processing issues. On the other hand, decreasing the value of Lq_i leads to better discernment (less chance of misclassification) between fault scenarios and the nominal dynamics, as well as overall greater noise rejection due to the increase of the hysterisis region.

7.2.2. Controller quality index

The controller quality index (q_c(t)) is reformulated into a structure similar to the novel identification quality index presented in (7.1). As it can be seen in (7.2), q_c(t) also makes use of two distinct filter parameters (γ_c^u and γ_c^d), making it possible for the quality index to respond differently for increasing and decreasing tracking errors. As with its identification counterpart, two thresholds, Hq_c and Lq_c, are used to respectively determine levels of logic high and low values independently.

$$q_c(t) = \gamma_c U(t) + (1-\gamma_c) q_c(t-1), \tag{7.2}$$

where $U(t)$ is the value of the utility function at time t, and γ_c assumes the value of γ_c^d if $U(t) \le q_c(t-1)$ or γ_c^u otherwise.

Within the proposed supervisor, the pair formed by q_i(t) and q_c(t) are responsible for the determination of the four states that compose the FDD decision logic. Therefore, as outlined for the identification quality index, the values of the four design parameters of q_c(t) will also have a direct impact on the supervisor's estimation of the plant's health and determination of when to perform the actions of switching and adding.

Following the same order of analysis used in the identification design parameters, we start by focusing on γ_c^u. As it can be expected, increasing its value makes higher tracking errors to translate more quickly into higher q_c(t) values, making the detection of all faults faster. Furthermore, higher values will also increase the chance of detecting faults with short persistence, whose detrimental effect might go unnoticed if the tracking error with short time span is filtered out before completely expressed. On the other hand,

smaller values of γ_c^u provide greater noise rejection, preventing high frequency measure and transmission noise to be interpreted as faults in the plant.

As for the filter parameter for decreasing q_c(t) values, γ_c^d, increasing it leads to a faster detection that the acceptable operational performance has been recovered, a situation that leads to the addition of a new fault solution to the DMB in the event of an AUF. It is important to make reach this decision fast enough so that an effective way to respond to a particular fault can be learned before the dynamics of the plant change once more due to an aggravation of the fault or in the event that the fault is transitory by nature. However, if set too high, this filter parameter will lead to a greater chance of a control solution to be added to the DMB before an entire reference cycle is covered, leading to the learning of incorrect solutions in the event of faults that only affect the dynamics of the plant in strong enough fashion in certain regions of the state space. As for noise rejection in the determination of the plant's health, smaller values of γ_c^d will grant it for a similar reason as γ_c^u provides noise rejection to fault detection.

Once q_c(t) extends beyond Hq_c, its high logic value is associated in the decision logic with situations of free and switched learning. Increasing Hq_c leads to a greater margin of acceptance of suboptimal tracking error and amount of control effort. This also leads to greater noise rejection in all frequency range when performing fault detection. Smaller values for the Hq_c threshold lead to faster fault detection and higher chance of successful detection. As a matter of fact, it set too high, Hq_c can cause less detrimental faults not be detectable, independent of the amount of time the fault remains active.

The controller quality index lower threshold Lq_c, makes the decision to add a new control solution to the DMB to be reached sooner if its value is increased. Increasing it

also leads to a greater chance of adding such solutions independent of the time required. In extreme circumstances, if made too small, the Lq_c threshold may not allow a control solution to be added to the DMB if the fault is severe enough that the minimum reachable U(t) is higher than the adjusted threshold. On the other hand, decreasing the value of Lq_c leads to a smaller chance of adding a new solution before the whole reference cycle has been covered. Smaller levels for this threshold also provide solutions with greater quality and greater specificity, while also granting increased noise rejection.

7.2.3. Weight quality index

The primary role of the weight quality index (q_w(t)) is to provide measure of the degree of activity within the weights of the neural networks that compose the ACD controller. This information is then used in conjunction with q_c(t) in the CMD decision logic to determine the health of the adaptation process of the base line controller. As with the previous two quality indexes, q_w(t) was also extended to admit different filter responses for increases ($\gamma_w{}^u$) and decreases ($\gamma_w{}^d$) in the amount of network activity, as shown in (7.3) in its discrete formulation. Different than the other two quality index, q_w(t) already possessed two distinct thresholds (Lq_w and Hq_w) since it was already important for the CMD decision logic to distinguish between high, low and normal levels of activity.

$$q_w(t) = \gamma_w \Delta w(t) + (1-\gamma_w) q_w(t-1), \tag{7.3}$$

where γ_w assumes the value of $\gamma_w{}^d$ if $\Delta w(t) \le q_w(t-1)$ or $\gamma_w{}^u$ otherwise, and the variation of the weights of the identification, action and critic NNs $\Delta w(t)$ is defined in Equation (7.4).

$$\Delta w(t) = \left\| w^i(t) - w^i(t-1) \right\| + \left\| w^a(t) - w^a(t-1) \right\| + \left\| w^c(t) - w^c(t-1) \right\| \tag{7.4}$$

Varying the four weight design parameters modifies how q_w(t) interprets the degree of activity within the NNs. Starting with $\gamma_w{}^u$, increasing it leads to a faster detection of controller malfunction due to online training algorithm divergence and a faster representation of the actual NN activity after a switching event by the supervisor. On the other hand, due to the same switching events and also due to abrupt changes in the plant dynamics caused by faults, spikes of high activity might surface in the NNs even during healthy operation and decreasing $\gamma_w{}^u$ leads to smaller chances that such spikes will translate to incorrect identification of adaptation divergence.

Similarly, if increased, $\gamma_w{}^d$ leads to a faster CMD in the event of local minima convergence. However, if decreased, it leads to a smaller chance of incorrect stagnation or incorrect true minima convergence detection before all state space covered by the reference cycle is explored.

Because the weight update equation in the backpropagation online training algorithm of GDHP is a function of the weights in the previous iteration, training divergence leads to a steady growth in the activity of the NNs as defined in (7.3). Therefore, independent of the value chosen for Hq_w, a divergent controller will always lead q_w(t) over it, correctly detecting and identifying the controller malfunction.

However, decreasing it leads to a faster detection and therefore greater chance that the recovery process can take place while the diverging pattern is restricted to the internal weights of the NNs and has not yet affected the input to the plant severely. On the other hand, increasing its value reduces the chance of incorrectly detecting a divergent behavior during normal training activity as a result of an abrupt fault.

Modifications on Lq_w have effects similar to the ones of the high threshold. Increasing it leads to faster CMD in the event of local minima convergence. On the other hand, decreasing it provides lesser chance that the adaptation process is seen as if had already converged while significant adaptation is still taking place, constituting a controller malfunction misdetection.

A summary of the effects on the performance of the proposed FTC architecture caused by all weight quality index design parameters as well as those from $q_c(t)$ and $q_i(t)$ can be found in Table 7.1.

7.3. FTC Design Parameters' Initialization Process

In the previous section, the three quality index used by the proposed FTC supervisor were revised to extended formulations that possess a total of 12 design parameters. While such flexibility is necessary in order to allow the user to adjust the response supervisor to each particular FTC application, it also creates the challenge of how to proceed in the adjustment of each and all parameters. As shown previously, each one of the design parameters affects the response of the supervisor in multiple and

sometimes conflicting ways, making the offline adjustment of such parameters non-trivial.

Table 7.1. Summary of effects of the 12 design parameters on the proposed architecture.

	Lq_i	Hq_i	γ_i^d	γ_i^u
increase	• Faster AKF classification. • Expansion on the region that accounts for fault variability.	• Less chance of transitory misclassification of AKFs as AUFs. • Greater noise rejection (amplitude).	• Faster AKF classification.	• Faster AUF classification.
decrease	• Less chance of misclassification by model generalization. • Greater noise rejection (amplitude).	• Faster classification of AUFs. • Less chance of transitory and permanent misclassification of AUFs as AKFs variations.	• Less chance of transitory AKF misclassification. • Greater noise rejection (frequency).	• Greater noise rejection (frequency). • Less chance of transitory misclassification of AKFs as AUFs.
	Lq_c	Hq_c	γ_c^d	γ_c^u
increase	• Faster decision to add a new solution to the DMB. • Greater chance of adding suboptimal solutions to the DMB.	• Greater margin of acceptance of suboptimal tracking error and/or amount of control effort. • Greater noise rejection (amplitude).	• Faster decision to add a new solution to the DMB.	• Faster fault detection. • Greater chance of detecting faults with short persistence.
decrease	• Less chance adding solutions to the DMB before all reference cycle is explored. • Greater noise rejection (amplitude).	• Faster fault detection. • Higher chance of successful detection.	• Less chance adding solutions to the DMB before all reference cycle is explored. • Greater noise rejection (frequency).	• Greater noise rejection (frequency).
	Lq_w	Hq_w	γ_w^d	γ_w^u
increase	• Faster detection of controller malfunction (local minima).	• Less chance of incorrectly detecting a divergent behavior during normal adaptation activity.	• Faster detection of controller malfunction (local minima).	• Faster detection of controller malfunction (divergence). • Faster representation of actual NN activity after switching.
decrease	• Less chance that the adaptation process is seen as if had already converged while significant adaptation is still in progress.	• Faster detection of controller malfunction (divergence).	• Less chance of incorrect convergence detection before all reference cycle is explored.	• Less chance that high activity spikes are misclassified as divergent behaviors.

As a matter of fact, to precisely determine the value of all design parameters at design time can be impossible to perform in applications that concern themselves with the occurrence of unknown faults. Therefore, the proposed procedure for the determination of the 12 design parameters that fulfill user defined FTC specification is performed in two parts: offline determination of initial values and online parameter tuning.

In the offline determination of initial values, a sequence of synthetic faults is simulated over the plant's nominal dynamics and the supervisor's design parameters are adjusted one at a time through the intricate procedure introduced in this section in order to generate a simulation response that achieves the following key user's FTC specifications:

- **Maximum acceptable tracking error and/or permissible control effort under the nominal scenario**: this specification determines the actual control goal of the plant under nominal operation conditions, not to be compromised by the addition of the FTC adaptive controller or supervisor.
- **Maximum acceptable tracking error and/or permissible control effort under a fault scenario**: in this specification the focus is on the performance of the plant under fault scenarios, which in some applications can be allowed to be somewhat smaller than the nominal scenario and still be considered an applicable solution.
- **Maximum fault detection delay**: fault detection is the first step in any active intervention by the FTC supervisor or human operator and therefore must be the fastest information to be gathered.

- **Maximum acceptable reconfiguration time for AKFs**: faults that are known during design time have pre-computed solutions that can be used to quickly recover the performance of the plant. The proposed FTC supervisor is capable of achieving this goal by switching to a solution stored in the DMB, but only after the AKF is correctly identified and classified.
- **Maximum fault identification delay for AUF**: although faster to be identified than known faults, AUFs present a greater FTC challenge since the baseline controller must by itself determine a solution through online adaptation. Identifying AUFs early allows more time for human operators to perform changes in the control mission or consider initiating safe shut-down procedure in the event of an uncontrollable fault.
- **Minimum observation time before adding a new model to the DMB**: this last specification depends mainly on the length of the desired reference cycle, but can also be determined by the frequency in which an intermittent known fault affects the plant.

Since the offline simulation makes use of the same adaptive critic controller used in the online application, it is expected to behave similarly when facing the synthetic fault set and the actual faults it will encounter in real world operation. However, unmodeled factors of the nominal plant (such as measurement noise) and specific effects of particular fault scenarios can make a set of offline determined design parameters to violate some of the before mentioned specifications. Furthermore, other effects of the design parameters listed in Table 7.1 that are not as crucial to the efficiency of the approach or are harder to

express in numbers at design time, such as how generic should each model within the DMB be or the amount of measurement noise present during a fault scenario, cannot be adjusted prior to actual application of the FTC architecture to the plant. Therefore, the goal of the offline simulation is to generate a set of initial values that can be fine tuned during actual operation by using Table 7.1 as a guideline.

The process of offline determination of initial values start with the generation of the synthetic fault sequence. Since ACD base line controller is based on a universal approximator structure, it makes no assumptions on the structures of the faults and therefore there is no need to implement overly complex dynamics for the synthetic faults. As a matter of fact, in the example demonstrated in the following section, even though the plant is expected to face nonlinear faults in actual application, the synthetic fault set is composed exclusively of faults with linear dynamics. It is important however that the synthetic faults have equal number of inputs and outputs that the nominal plant dynamics.

One important design choice to be taken prior to the procedure for determination of initial values of the design parameters is to establish the complexity of the base line adaptive controller. In the case of the proposed architecture, this translates to choosing the size and architecture of the three NNs that compose the GDHP adaptive controller. The number of weights, their configuration and choice of training algorithm will all have an effect on how the design parameters affect the response of the supervisor, so it is essential to select the appropriate values in advance.

Having set the complexity of the base line adaptive controller, the next step is to simulate the synthetic linear fault sequence and record the responses of the three quality indexes. In order to observe the reaction of the indexes to both known and unknown

faults, as well as allow full expression of the adaptive controller training process, the supervisor's operations of switching and adding are disabled during the simulation.

The simulation starts with the plant with the nominal dynamics, where it remains until the base line adaptive controller provides a control solution that provides a performance level equal or higher to the minimum acceptable under nominal conditions. Once such level is reached, the simulation is paused and the developed nominal control solution and identification model are added to the DMB. Doing so provides the supervisor with a control solution for the nominal scenario that fulfills one of its FTC specification and also makes all subsequent times the plant assumes the nominal dynamics to be interpreted as a known scenario by the supervisor and generate the related responses in all quality indexes. As part of the information required later for the determination of initial values for the filter parameters of $q_i(t)$, through the course of the simulation, store in file (not within the DMB) copies of the weights of the IdNN after convergence is reached in each presented fault scenario.

As made clear in Table 7.1, although relevant distinctions exist, many threshold alteration effects are shared with the filter parameters of the same quality index. Therefore, in the proposed methodology, initial values for thresholds are defined first based on exclusively individual effects, followed by the filter parameters, whose determination focuses primarily on temporal effects. For this reason, during the simulation all filter parameters are set to 1, resulting in filters with no memory and in practice allowing the threshold to be set according to the unfiltered information.

The methodology for the determination of suitable initial values for the thresholds involves the gathering of responses in the simulations that represent limits in the

expression of the unfiltered quality indexes during certain situations relevant to each threshold. For the identification and controller ones, these limits cannot be used directly as the thresholds as they cannot be expected to represent overall limits for all possible fault scenarios, including those unknown during design time. In order to provide suitable initial conditions that, once again, can be fine tuned once applied in practice, the actual initial condition values for the identification and controller thresholds are obtained from within the range defined by the obtained limits. We have found in simulated experiments that using 25% and 75% of the gap between the measured limits produce suitable values for the low and high thresholds respectively. An over bar is adopted in the notation to indicate a threshold limit, as in $\overline{Lq}_i$ being the measured limit of Lq_i, the lower identification quality index threshold.

The next steps deal with obtaining the threshold limits from the simulation data for the identification and controller thresholds. Justifications for the setting of such limits are intrinsically connected to the information they contribute to the FDD decision logic, described in detail in Section 4.1.2. Starting with the lower identification threshold, as it is related to the identification of known faults, $\overline{Lq}_i$ is obtained from the maximum observed value (after the transitory response) when the plant enters a known scenario, in this case, when the plant returns to the nominal scenario from a fault scenario. In order to obtain $\overline{Hq}_i$, it is necessary to first measure the maximum value assumed by $q_i(t)$ in each fault (and therefore unknown) scenario. The limit for the higher identification threshold is the obtained from the minimum of such measurement, as it indicated the minimum response expressed by an AUF.

The determination of the limit $\overline{Lq}_c$ can actually be made directly from the FTC specifications, by taking the value of the utility function $U(t)$ that corresponds to the minimum acceptable performance under a fault scenario. On the other hand, $\overline{Hq}_c$ is related to the transient performance degradation used to, among other things, detect the occurrence of a fault, and therefore must be obtained from the simulation data. In particular, it assumes the value of the minimum $q_c(t)$ peak observed following a change in the dynamics caused by either the introduction of a fault scenario or a return to the nominal dynamics.

The weight quality index threshold is involved exclusively in the CMD decision logic presented in Section 5.2. Different from the previously discussed thresholds, the ones related to $q_w(t)$ deal with the extreme situations of controller malfunction due to divergence and local minima convergence and therefore can have initial values extrapolated directly from the data. The lower threshold purpose is to inform the CMD decision logic of when all NNs have converged. Therefore, a suitable initial value can be obtained from the overall minimum among the maximum values assumed by $q_w(t)$ during the last reference cycles in each scenario. The higher threshold, on the other hand, serves the purpose of detecting divergent behavior within the NNs characterized by an increased degree of activity. Due to the nature of NN learning algorithm divergence, the activity within a NN, as translated into $q_w(t)$, continuously increases. Therefore, any threshold value applied to Hq_w will eventually be crossed in the event of a divergent controller malfunction. However, the smaller the value attributed to it, the sooner $q_w(t)$ will indicate to the CMD decision logic that the learning process is no longer stable and greater are the chances that corrective measure can be taken while the effects of the malfunction remain

internal and before the operation of the plant is compromised. Nevertheless, it is important to ensure that this threshold will not in any event be achieved during healthy adaptive controller operation, otherwise its efficiency could be compromised due to incorrect supervisory intervention. Therefore, the initial value for Hq_w is chosen as one order of magnitude higher than the maximum $q_w(t)$ value observed throughout the simulation.

Having adjusted all 6 thresholds to suitable values given the levels observed in the simulated response, the next half of the proposed approach involves the adjustment of filter parameters that will control the time for the quality indexes to cross their respective thresholds in a manner to fulfill the temporal FTC specifications. Although adjusting the filter parameters may seem a computationally expensive process, testing for different filter values required few computations since for $q_c(t)$ and $q_w(t)$ the signals to be filtered are already available as the unfiltered versions obtained during simulation, and the filters themselves are not more complex than first order low pass filters. Starting with the filter parameters of $q_c(t)$, γ_c^d should be adjusted so that, from the last moment the U(t) peaks over the higher threshold, at least one reference cycle is covered before the filtered quality index reaches its lower threshold level, as observed in all scenario transitions. The reason for this build-in delay of one cycle is that a low value of $q_c(t)$, among other functions, indicates that a suitable solution was found for an AUF and that therefore such solution can be added to the DMB. In the event that a fault becomes significantly active only in a portion of the state space traveled by the reference cycle, it is possible that a controller will display a poor performance only in such portion. Unless a complete cycle

is observed before fully judging the efficacy of the control solution, a model that do not properly address to region-specific fault may be erroneously added to the DMB.

The desired fault detection time can be achieved by adjusting γ_c^u in order to regulate the delay between the introduction of a fault scenario and the time the filtered $q_c(t)$ reaches its higher threshold. Adjusting the filter parameter to provide a fault detection delay less or equal to the specified one in all fault scenarios in the simulations provides a suitable initial value. For the weight quality index, γ_w^d should be adjusted in order to provide a wait at least three reference cycles before allowing the lower threshold to be reached. As it can be expected, the reasoning behind this step is similar to the determination of the initial value of γ_c^d, with the difference that more reference cycles are required due to the manner these two quality indexes interact in the CMD decision logic. For γ_w^u, a value of 1 can be maintained as an initial value for this design parameter due to the fact that the higher threshold is already positioned far from signals obtained during healthy learning and to the fact that early CMD is critical for its recovery.

The calculation of the effect of different filtering values on the response of $q_i(t)$ is slightly more computationally expensive than the other two quality indexes since in order to provide sufficient data, the identification quality index must be calculated as if all fault scenarios were AKF. It is with this purpose that the weights of the IdNN at the end of each scenario are store and used here to calculate the identification error of all models throughout the simulation. For γ_i^d, experiment with different values in order to provide less than the maximum acceptable identification delay for AKFs as observed in all scenario changes in the simulation. Note that this delay has a direct impact on the specification for the maximum acceptable reconfiguration time for AKFs since the

supervisor can only increase the reconfiguration speed of such faults after they were correctly identified and classified. Finally, adjust γ_i^u so as to provide less than the maximum permissible identification delay for AUFs.

Table 7.2. Summary of the proposed procedure for the initialization of FTC design parameters.

Simulation Setup
• Start the simulation with the plant under the nominal scenario until the baseline controller produces a solution capable of providing the plant with a tracking performance equal or superior to the required nominal performance.
• Switching is deactivated in the supervisor. Adding models and solutions to the DMB is also deactivated after a model of the nominal dynamics is added in the first part of the simulation.
• Store the unfiltered values of $q_w(t)$ and $q_c(t)$ so that different filter parameters can be applied later without the need of further data acquisition.
• Pre-set all filter parameters to 1 (no memory)
• Apply a series of linear synthetic faults, compensating for the application's number of inputs, outputs and order of the nominal dynamics.
• Store in file (not in the DMB) copies of the weights of the IdNN at the last iteration within each fault scenario.
FTC Design Parameters' Initialization Procedure
1. Obtain $\overline{Lq}_i$ from the maximum value observed (after the transitory response) when the plant returns to the nominal scenario from a fault scenario.
2. Measure the maximum $q_i(t)$ value observed during each fault scenario after the initial transitory peak. Obtain $\overline{Hq}_i$ from the lesser of such measurements.
3. Calculate $\overline{Lq}_c$ using the desired $U(t)$ level that corresponds to the acceptable performance when the plant is under a fault scenario.
4. Obtain $\overline{Hq}_c$ from the minimum $q_c(t)$ peak observed following a change in the dynamics of the plant.
5. Obtain Lq_i and Hq_i by setting them at respectively 25% and 75% from the difference between $\overline{Lq}_i$ and $\overline{Hq}_i$. Do the same for the $q_c(t)$ equivalents.
6. Set Lq_w to the minimum among the maximum $q_w(t)$ value observed in the last reference cycles of each scenario.
7. Set Hq_w to one order of magnitude higher that the maximum observed $q_w(t)$ value.
8. Using the previously defined control thresholds and the stored $q_c(t)$: 8.1. Set γ_c^d so that at least one full reference cycle is covered before Lq_c is reached. 8.2. Set γ_c^u in order to provide less that the permissible maximum fault detection delay.
9. Using the previously defined weight thresholds and the stored $q_w(t)$: 9.1. Set γ_w^d so that at least three reference cycles are covered before Lq_w is reached. 9.2. Set γ_w^u to 1 (no memory).
10. Retrieve the copies of the weights of the IdNN obtained for each fault scenario and use those to again generate $q_i(t)$ as if the supervisor had solutions for the nominal as well as all fault scenarios within the DMB during simulation time. Using the previously defined identification thresholds and the stored identification models: 10.1. Set γ_i^d to provide less than the maximum permissible AKFs identification delay. (direct impact on maximum permissible AKFs reconfiguration delay). 10.2. Set γ_i^u to provide less than the maximum permissible identification delay for all AUFs.

Table 7.2 summarizes the simulation details and the procedure to generate suitable initial conditions for all 12 FTC thresholds and filter parameters as described in detail above. Having concluded this stage, the supervisor is configured to be applied to the actual plant. As mentioned previously, it is possible that in the event of AUFs not all FTC specifications will be fulfilled, as it is possible that unmodeled system properties, such as measurement noise, will cause the quality indexes to respond inaccurately. To correct such, the user is directed to use Table 7.1 as a look up table and make the necessary adjustments on the design parameters to rectify specific deficiencies. The only exception is the case in which after implementation, the chosen baseline adaptive controller fails to have the complexity necessary to provide solutions with less than the minimum acceptable performance to AUFs. In such a case there is need to repeat the presented offline procedure for the determination of a new set of initial values based on a more powerful adaptive controller (e.g. increased number of layers or neurons).

7.4. Simulation Results

In this section we demonstrate how the proposed procedure can be used in practice to configure the 12 FTC design parameters of the described approach before its actual application to the nonlinear complex plant we ultimately desire to provide fault tolerance. In particular, the plant of interest has 2 outputs, 2 inputs, 3 states, and will be subject to nonlinear faults that will greatly change its dynamics. The offline simulation procedure will be conducted by applying a sequence of synthetic linear faults to the nominal model of the plant in order to generate information to be used to produce initial

values for 12 FTC design parameters that aim at achieving the FTC specifications listed in Table 7.3.

Table 7.3. Summary of FTC specifications. Temporal values expressed in number of iterations (it.) and in terms of the length of the cyclic reference period (ref.).

FTC specification	limit
Maximum acceptable tracking error under the nominal scenario	0.02
Maximum acceptable tracking error under a fault scenario	0.05
Maximum fault detection delay	75 it. (0.15 ref.)
Maximum fault identification delay for AUF	100 it. (0.2 ref.)
Maximum acceptable reconfiguration time for AKFs	3500 it. (7 ref.)
Minimum observation time before adding a new model to the DMB	750 it. (1.5 ref.)

For the sake of understanding, we assume in this simulation that the plant already has a stable efficient controller designed for the nominal dynamics and that the FTC architecture is mounted over the loop containing the plant and the nominal controller. Under these circumstances, we can assume without any loss of applicability, that the nominal dynamics are described by Equation (7.4).

$$\begin{cases} x_1(t+1) = u_1(t) \\ x_2(t+1) = x_3(t) \\ x_3(t+1) = u_2(t) \end{cases} \tag{7.4}$$

where $u_1(t)$ and $u_2(t)$ are the inputs of the plant and the states $x_1(t)$ and $x_2(t)$ compose the output of the plant, also represented in vector form in $R(t) = \begin{bmatrix} x_1(t) & x_2(t) \end{bmatrix}^T$.

In this demonstration, we assume that the desired minimum performance of the plant under the nominal scenario corresponds to U(t)=0.02, where U(t) is chosen as shown in (7.5) focusing exclusively on the tracking error of the output. On the other hand, a performance of U(t)=0.05 is still considered acceptable under a fault scenario.

$$U(t) = \left(R(t) - R^t(t)\right)^T \left(R(t) - R^t(t)\right) \tag{7.5}$$

where $R^t(t)$ is the desired reference signal with a cycle of 500 iterations described in Equation (7.6)

$$R^t(t) = \begin{bmatrix} \sin\left(\frac{\pi}{250}t\right) + 0.4\sin\left(\frac{2\pi}{250}t\right) \\ 0.1 + \sin\left(\frac{\pi}{250}(t+150)\right) + 0.6\sin\left(\frac{2\pi}{250}(t+190)\right) \end{bmatrix} \tag{7.6}$$

7.4.1. Adjusting initial FTC design parameter values using simulated linear faults

Directly from the FTC specifications in table 7.3 it is possible to directly set $\bar{L}q_c = 0.05$, however in order to set the values of the remaining thresholds, it is necessary to run a simulation sequence using the same adaptive controller we seek to apply to deal with faults in the actual plant. Taking into consideration the presence of 3 states, 2 inputs and 2 outputs, and assuming a certain degree of complexity for the unknown fault scenarios, all three neural networks that compose the ACD adaptive controller are created as 2 layered recurrent neural networks with 40 hidden neurons. Having set the base line adaptive controller, the next step is to determine a simulation sequence with synthetic faults represented by linear systems with the same dimensions as the nominal plant. Equations (7.7) to (7.10) describe the dynamics of abrupt faults 1 to 4 respectively. Table (7.4) displays the simulation sequence and the timing of each fault.

$$\begin{cases} x_1(t+1) = -0.44x_1(t) - 0.09x_2(t) - 0.01x_3(t) + 2u_1(t) \\ x_2(t+1) = x_3(t) - 2u_2(t) \\ x_3(t+1) = x_1(t) \end{cases} \tag{7.7}$$

$$\begin{cases} x_1(t+1) = -0.67x_1(t) - 0.11x_3(t) + 5u_1(t) \\ x_2(t+1) = 0.1x_2(t) + 2u_2(t) \\ x_3(t+1) = 0.3x_1(t) \end{cases} \tag{7.8}$$

$$\begin{cases} x_1(t+1) = -0.19x_1(t) - 0.1x_2(t) - 0.17x_3(t) + 1.2u_1(t) \\ x_2(t+1) = 0.9x_1(t) + 1.32u_2(t) \\ x_3(t+1) = x_2(t) - 0.7u_1(t) \end{cases} \tag{7.9}$$

$$\begin{cases} x_1(t+1) = 0.2x_1(t) - 0.96x_2(t) + 0.4x_3(t) + u_1(t) \\ x_2(t+1) = 0.9x_1(t) + 1.5u_2(t) \\ x_3(t+1) = x_2(t) + u_1(t) \end{cases} \tag{7.10}$$

Table 7.4. Simulation sequence for the linear synthetic fault set.

Active time interval	Plant dynamics
0 to 15000	Nominal - Equation (7.4)
15000 to 25000	Abrupt Fault 1 - Equation (7.7)
25000 to 35000	Abrupt Fault 2 - Equation (7.8)
35000 to 45000	Nominal - Equation (7.4)
45000 to 55000	Abrupt Fault 3 - Equation (7.9)
55000 to 65000	Abrupt Fault 4 - Equation (7.10)
65000 to 75000	Nominal - Equation (7.4)

Following the directives laid down in the previous section, the simulation was run for the total 75000 iterations. Analyzing the transitory behavior of $q_c(t)$ after the introduction of each scenario, the minimum peak value was observed when the plant returned to the nominal scenario at iteration 65000 (Figure 7.1), setting $\bar{H}q_c$ at 0.860.

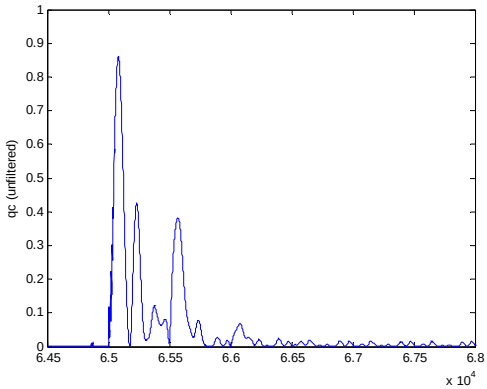


Figure 7.1. Unfiltered $q_c(t)$ transitory response as it returned to the nominal scenario at iteration 65000.

After the transitory response, the maximum value of $q_i(t)$ observed while in the nominal scenario was of 0.318, establishing in this manner $\overline{Lq}_i$. The measurement, obtained in the first time the plant returned to the nominal scenario after 35000 iterations, can be seen in Figure 7.2. Moving forward, Figure 7.3 shows the response of $q_i(t)$ while abrupt fault 3 is active. The maximum value observed in this scenario (after the transitory response generated by the abrupt change in dynamics) was the minimum among all other scenarios, determining the limit of the high $q_i(t)$ threshold, $\overline{Hq}_i$, to be set at 0.575.

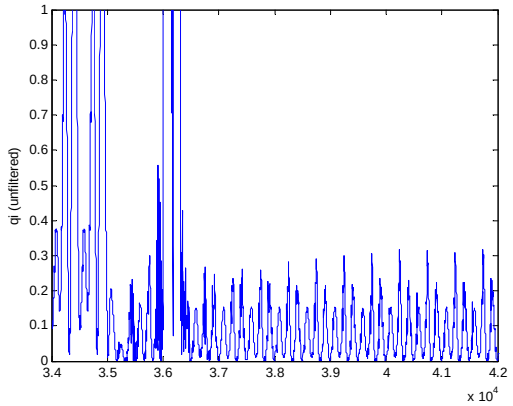


Figure 7.2. Response of the unfiltered identification quality index as the plant returns to the nominal scenario at iteration 35000.

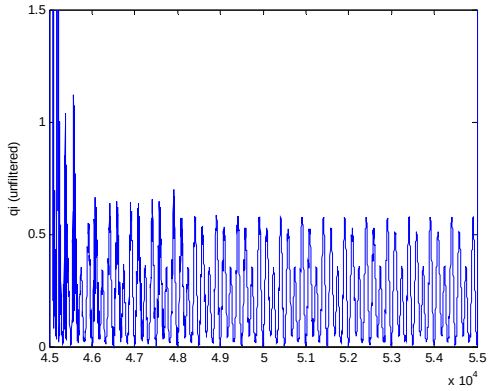


Figure 7.3. The unfiltered identification quality index during abrupt fault 3.

Having obtained the limits for the identification and controller thresholds, it is then possible to determine the initial value recommendations for these thresholds as described in the previous section. The calculation results lead to the values listed in Table 7.5.

Table 7.5. Initial values for the identification and control thresholds calculated from observed limits.

Threshold	Initial Value
Lq_c	0.253
Hq_c	0.658
Lq_i	0.382
Hq_i	0.511

In order to determine the initial values for the weight quality index thresholds, an exploration of the unfiltered $q_w(t)$ generated during the simulation (Figure 7.4) is required. The maximum observed peak value of the weight quality index during operation in the absence of controller malfunctions was 566.5, leading to the adjustment of the higher threshold Hq_w to 5665. For the lower threshold, we focus on the maximum

response observed after controller convergence at each scenario. Figure 7.5 displays the minimum of such measurements, adjusting the initial value of the lower weight threshold Lq_w to 3.75.

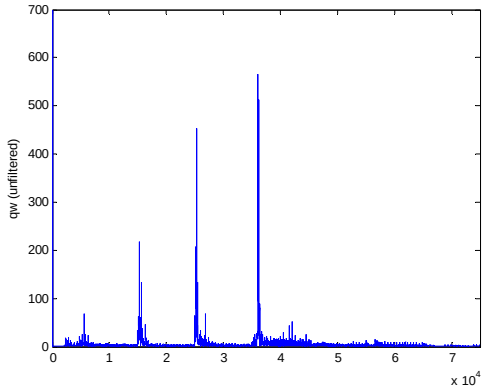


Figure 7.4. Resulting unfiltered weight quality index from the synthetic fault sequence simulation.

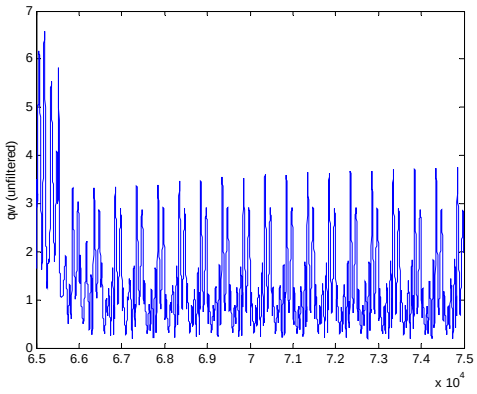


Figure 7.5. Detail of the response of $q_w(t)$ during a period when no faults are active in the plant. The minimum $q_w(t)$ response after controller convergence can be seen in this graph.

Having set initial values for all thresholds, the next step is the configuration of the filter parameters. In order to generate the relevant data, identification models of all synthetic linear scenarios visited in the simulation were added to the DMB through the process described in the previous section. The identification quality index was then recalculated independently to allow different filter parameters to be tested in order to

fulfill the FTC specifications. Figure 7.6 illustrates the distinct difference between $q_i(t)$ before and after filtering with its initial parameters. On Figure 7.7 it is possible to see how, using the thresholds defined previously, the filtered $q_i(t)$ correctly identifies all scenarios as known and classifies correctly which of the models in the DMB represents the current observed dynamics. Note also that all transitory periods during which the models are misclassified occur within period during which $q_i(t)$ transitorily misidentifies an AKF as an AUF. This behavior is expected and desired since neither correct identification nor classification can occur until sufficient data is made available and while an AKF remains transitorily misidentified as an AUF the switching command is not issued by the FTC decision logic.

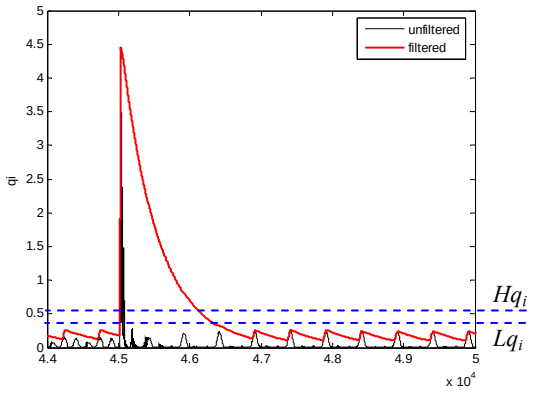


Figure 7.6. Comparison between unfiltered and filtered identification quality indexes. The horizontal dashed lines indicated the adjusted threshold levels. The simulation section displayed in the graph draws attention to the introduction of an AKF at iteration 45000.

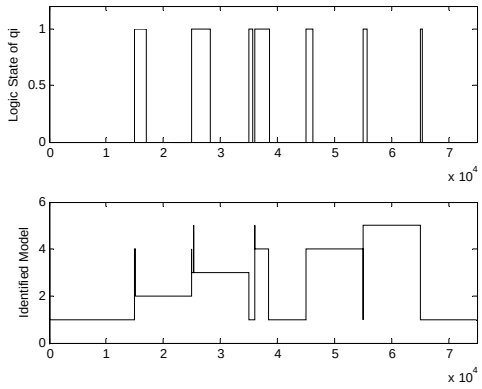


Figure 7.7. On the top graph, the logic state, low (0) or high (1), of $q_i(t)$ throughout the simulation. The bottom graph displays the model identified as active at each iteration; model 1 pertains to the nominal dynamics, while 2 to 5 pertain to the four fault scenarios.

Adjusting γ_i^u to 0.8 provided a maximum AUF identification delay of 68 iterations (less than the specified 100 iterations), observed after the system's dynamics abruptly changed at iteration 65000 (Figure 7.8). In order to provide the means for the final configured supervisor to be able to comply with the desired maximum reconfiguration time for AKF, correct fault identification must occur before a switching operation takes place. Therefore, γ_i^d was set to 0.002, which produced a maximum AKF identification delay of 3204 iterations among all scenario changes experienced in the linear dynamics simulation (as shown in Figure 7.9).

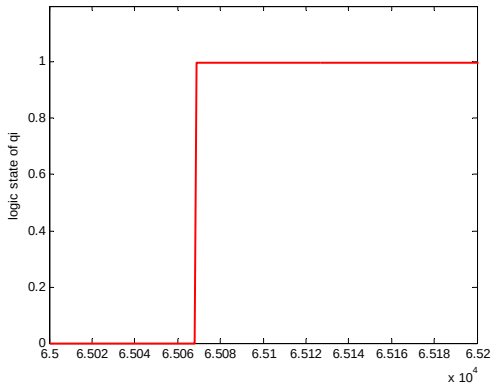


Figure 7.8. Longest AUF identification delay (after γ_i^u adjustment) observed here as the time taken by $q_i(t)$ to assume its high logic value.

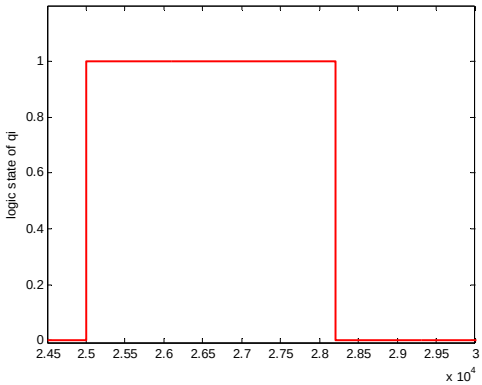


Figure 7.9. Identification quality index logic state reacting to the introduction of an AKF at iteration 25000.

In order to achieve fault detection under the specified maximum delay, γ_c^u was set to 0.6, resulting in a maximum observed fault detection delay of 55 iterations (less than the required 75 iterations). Triggered by a high logic value of $q_c(t)$, the maximum observed fault detection delay can be seen in Figure 7.10. A series of values for γ_c^d were tested, checking for the resulting minimum number of iterations observed before a control success is declared by a return of the logic value of $q_c(t)$ to low. Adjusting the parameter to 0.002 provided a minimum observation time of 853 iterations (beyond the specified 750 iterations) as seen in Figure 7.11. The resulting filtered qc(t) can be seen in Figure 7.12 and the resulting state transitions throughout the simulation base on the previously defined thresholds are displayed in Figure 7.13.

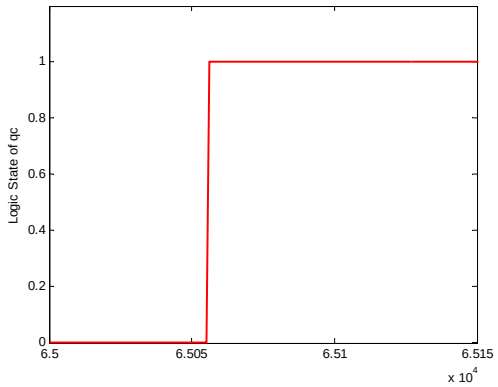


Figure 7.10. Change in the logic state of $q_c(t)$ in response to the change in the dynamics of the plant at iteration 65000.

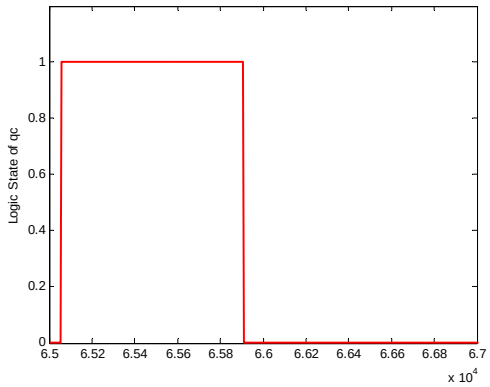


Figure 7.11. Logic state of $q_c(t)$ following introduction of new dynamics at iteration 65000 and subsequent performance recovery.

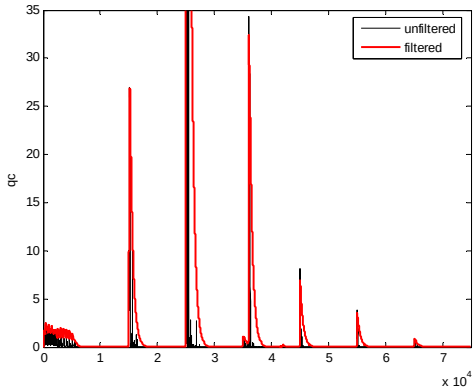


Figure 7.12. Comparison between $q_c(t)$ before and after filtering using the chosen parameters.

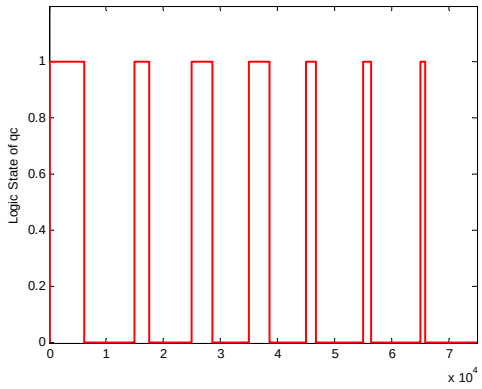


Figure 7.13. Logic values, high (1) and low (0), expressed by $q_c(t)$ throughout the simulation.

Following the proposed procedure, the adjustment of the final two design parameters take place by keeping $\gamma_w{}^u$ as 1 (unfiltered) and varying $\gamma_w{}^d$ in order to prevent a low logic value to be expressed before sufficient observation time is allowed. In this case, given the specified minimum observation time for models to be added to the DMB, $\gamma_w{}^d$ was modified until a minimum observation time of 1620 iterations was obtained, leading to a value of 0.0007 for the filter parameter (Figure 7.14). A comparison of the response of the filtered and unfiltered $q_w(t)$ though the course of the entire simulation can be seen in Figure 7.15.

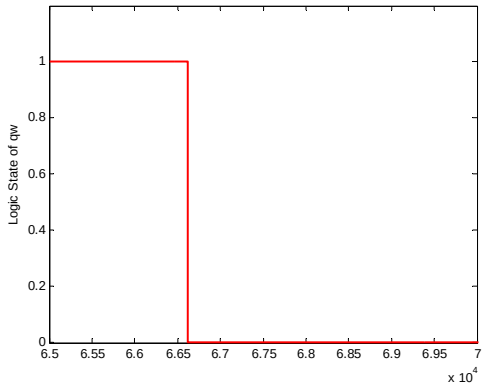


Figure 7.14. Logic state of $q_w(t)$ (low (0), normal (1) and high (2)) depicting the healthy activity in the adaptive critic controller following the introduction of new dynamics and subsequent convergence.

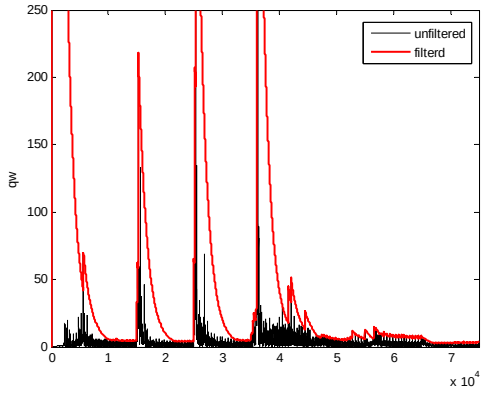


Figure 7.15. Comparison between filtered and unfiltered $q_w(t)$ throughout the whole simulation.

Having concluded the determination of initial values for all 12 proposed FTC design parameters, Table 7.6 summarizes the results as they would be used in practice to configure the FTC supervisor for application to the real world plant.

Table 7.6. Initial values for the 12 proposed FTC design parameters.

Threshold	Initial Value	Filter Parameter	Initial Value
Lq_c	0.253	γ_i^u	0.8
Hq_c	0.658	γ_i^d	0.002
Lq_i	0.382	γ_c^u	0.6
Hq_i	0.511	γ_c^d	0.002
Lq_w	3.75	γ_w^u	1
Hq_w	5665	γ_w^d	0.0007

7.4.2. Applying the configured supervisor to a plant subject to nonlinear faults

In this demonstration, the real world plant was simulated by a plant subjected to nonlinear fault scenarios of greater complexity than the linear ones used in the linear synthetic fault sequence. Through the course of the simulation, the plant was subject to two novel AUFs and one incipient fault. It is important to note that no incipient faults were present in the synthetic linear fault set used for initialization of the design

parameters. In addition, one of the AUFs was presented twice in order to present the FTC architecture with the challenge to learn a solution for an AUF in its first occurrence, and then apply it directly once the fault is recognized as an AKF in its second occurrence.

The timeline for introduction of fault scenarios can be seen in Table 7.7. Equations (7.11) and (7.12) describe the dynamics of abrupt fault 1 and abrupt fault 2, respectively. Finally, the incipient fault is described as a gradual change in the nominal dynamics (Equation (7.4)) leading to the dynamics described in Equation (7.13) at the end of its active interval.

Table 7.7. Simulation sequence of actual implementation.

Active time interval	Plant dynamics
0 to 15000	Nominal - Equation (7.4)
15000 to 25000	Abrupt Fault 1 - Equation (7.11)
25000 to 35000	Nominal - Equation (7.4)
35000 to 45000	Incipient Fault
45000 to 55000	Abrupt Fault 2 - Equation (7.12)
55000 to 65000	Abrupt Fault 1 - Equation (7.11)

$$\begin{aligned} x_1(t+1) &= x_3^2(t) + u_1(t) \\ x_2(t+1) &= x_3(t) - 0.8x_2(t)u_2(t) \\ x_3(t+1) &= -0.5x_3(t) + 1.5u_2(t) \end{aligned} \tag{7.11}$$

$$\begin{aligned} x_1(t+1) &= 0.5x_1(t) - 0.7x_3^2(t) + 1.5u_1(t) - 0.001\sin\left(\frac{\pi}{4}u_1(t)\right) \\ x_2(t+1) &= 2x_3(t) + x_2(t)u_1(t) + 0.2\frac{x_2(t)}{1+x_3^2(t)} \\ x_3(t+1) &= 0.04x_3(t) + u_2(t) + 0.6u_2(t)\sin\left(\frac{x_1(t)}{2}\right) \end{aligned} \tag{7.12}$$

$$x_1(t+1) = u_1(t) + 1.25u_2(t) + \frac{x_1(t)}{1+x_1^2(t)}$$
$$x_2(t+1) = \left(1 + 0.5\sin\left(4x_3(t)\right)\right)x_3(t) \tag{7.13}$$
$$x_3(t+1) = u_2(t)$$

Throughout the simulation, the supervisor, operating with the design parameters stipulated in the previous subsection, was capable of correctly identifying all fault scenarios and perform the operations of adding new models to the database and switching to known solutions in a manner to increase the efficiency of the FTC response. Table 7.8 provides a timeline of the supervisor's responses, including the information made available to the user and the actions taken in interaction with the base line controller and the DMB.

Table 7.8. Information gathered and actions taken by the supervisor.

Iteration	Supervisor's Responses
15002	Abrupt Unknown Fault
18936	Control Success - ADD model 2
18937	Performance recovered (model 2)
25046	Abrupt Known Fault - SWITCH to model 2
25087	Abrupt Unknown Fault
25513	Abrupt Known Fault - SWITCH to model 1
25930	Performance recovered (model 1)
43896	Incipient Fault
45012	Abrupt Unknown Fault
48470	Control Success – ADD model 3
48471	Performance recovered (model 3)
55035	Incipient Fault
55046	Abrupt Unknown Fault
55503	Abrupt Known Fault - SWITCH to model 2
56086	Performance recovered (model 2)

The first specification posed in this demonstration indicates a maximum acceptable tracking error for the plant during nominal operation. Comparing the two times the nominal scenario becomes active during the simulation, the largest observed tracking error after the transient response caused by the change in dynamics was of 0.004, safely below the desired 0.02. Figure 7.16 displays the value of the instantaneous tracking error (as available in $U(t)$) during the last 5 cycles of the referred scenario, while Figure 7.17 displays the actual outputs of the plant as they closely follow the desired sinusoidal trajectories.

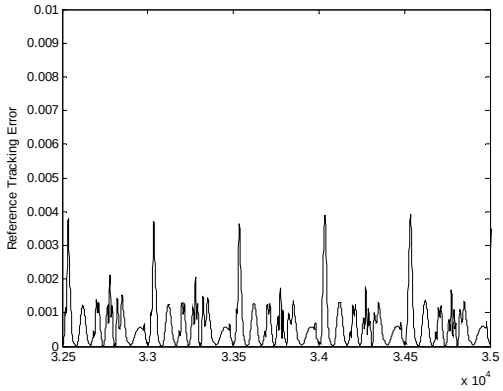


Figure 7.16. Reference tracking error during the last 5 cycles in the nominal scenario.

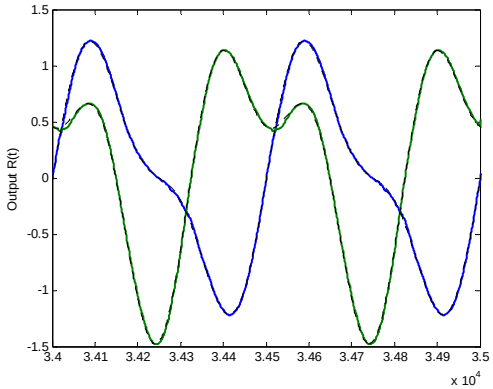


Figure 7.17. The plant's two outputs during nominal operation. Reference signals plotted in dashed lines.

The tracking error during a fault scenario was also a concern in the specification for this demonstration. The maximum observed tracking error in the last 5 cycles among all fault scenarios was observed at the reintroduction of abrupt fault 1. The measured value of 0.034, although higher than the one achieved during nominal operation, is still below the specified maximum of 0.05. Figure 7.18 display the referred tracking error, and Figure 7.19 brings the slightly deteriorated, but still acceptable, outputs of the plant under the same fault scenario. Had the baseline controller failed to fulfill any of these first two specifications, the designer would have to expand the NN structures and/or modify learning parameters and obtain new initialization values for the design parameters through the same process taken in the previous subsection. Being not the case, the analysis of the FTC response continues.

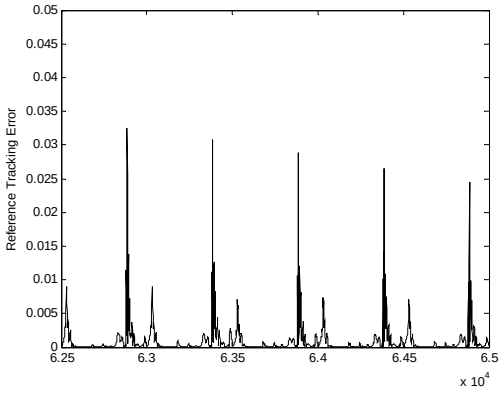


Figure 7.18. Maximum reference tracking error observed during the last 5 cycles over all fault scenarios.

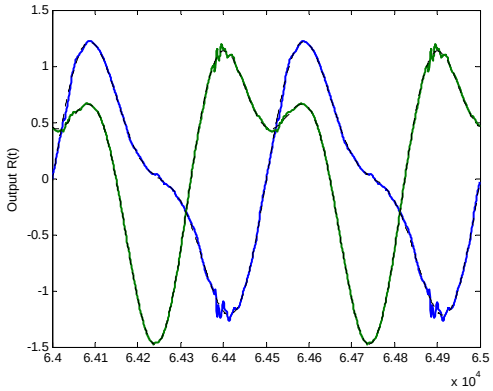


Figure 7.19. The plant's two outputs during the fault scenario with maximum observed tracking error. Reference signals plotted in dashed lines.

The maximum observed fault detection delay can be obtained directly from Table 7.8 by measuring the maximum time gap between the introduction of a fault scenario and the time the supervisor detects a fault. It is important to clarify that the goal of fault detection is merely to indicate that a fault is present, not to identify it. Frequently, due to the fact that the quality indexes measure independent signals and are independently configured, on the process of moving to the correct state within the decision logic, depending on which quality index responds faster, a series of transitory responses may be reached. For instance, it is possible for an AKF to be transitorily identified as an AUF in the first moments after its detection before $q_i(t)$ drops enough for the supervisor to determine if the fault scenario is already known within the DMB. The presence of responses such as these are expected and both proposed decision logics are built taking such transitory behavior in account. Therefore, fault detection delay is measured until the detection of a fault, independent if it was correctly identified at the onset or not. With this in mind, the maximum observed fault detection delay was of 42 iterations (below the specified maximum of 75 iterations), recorded at the re-introduction of abrupt fault 1.

The referred event can also be graphically seen in Figure 7.20 as fault detection takes place in the instant $q_c(t)$ crosses over the its high threshold.

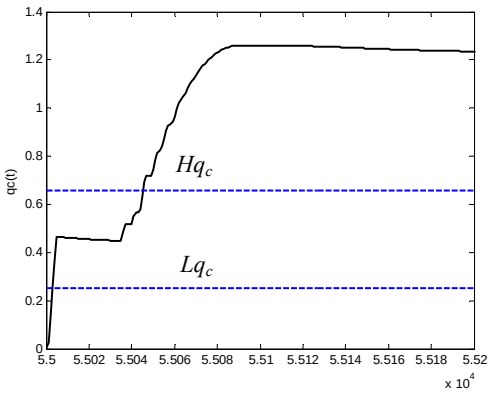


Figure 7.20. Response of $q_c(t)$ in the first 200 iterations after introduction of abrupt fault 1. Maximum observed fault detection delay occurs at 46 iterations after the fault introduction as the quality index crosses Hq_c.

For the maximum AUF identification delay we focus on the supervisory response (as shown in Table 7.8) during the first occurrence of the two abrupt faults at iterations 15000 and 45000. The maximum observed delay was of 12 iterations, well below the specified limit of 100 iterations.

One key aspect of the proposed FTC architecture and an FTC specification in this demonstration is the amount of time allowed for reconfiguration of an AKF. In the simulation, the abrupt fault 1 is introduced twice and the supervisor must successfully learn a solution in the fault's first occurrence, autonomously add it to the DMB and present it in the fault's second occurrence fast enough in order to provide a shorter reconfiguration delay. In the simulation, an actual reconfiguration delay for AKFs of 1086 iterations was achieved, making it less than one third the specified maximum of 3500 iterations and almost one fourth of the reconfiguration time taken during the first occurrence of 3937 iterations. A graphical indication of how important the choice of

suitable thresholds and filter parameters is for the efficiency of the supervisor can be found in Figure 7.21, which depicts the response of the controller quality index during the second occurrence of the abrupt fault 1.

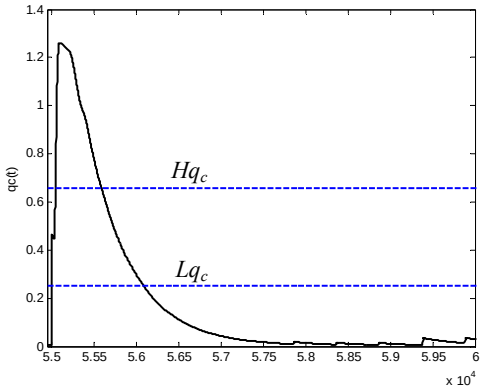


Figure 7.21. Faster reconfiguration time through switching operation on the second occurrence of abrupt fault 1.The reconfiguration time of 1086 iterations is achieved when $q_c(t)$ moves below Lq_c.

The final FTC specification of this demonstration deals with the minimum time desired for observation of the behavior of the plant before a new solution is added to the DMB. This time gap is obtained from the last moment the tracking error goes below the value specified by the lower qc(t) threshold, until the time the new control solution is added to the DMB. This process occurs twice in the simulation, once for each abrupt fault. As seen in Figure 7.22, the minimum observed wait time for the addition of a new model to the DMB occurred during abrupt fault 2 and had the value of 1948 iterations, a number safely larger than the required minimum of 750 iterations. Table 7.9 summarizes the results of this demonstration by comparing the stipulated FTC specifications and the achieved results. The methodology for generating suitable initial values for all design parameters was successful in configuring a FTC supervisor capable of achieving all specifications without the need of further tuning.

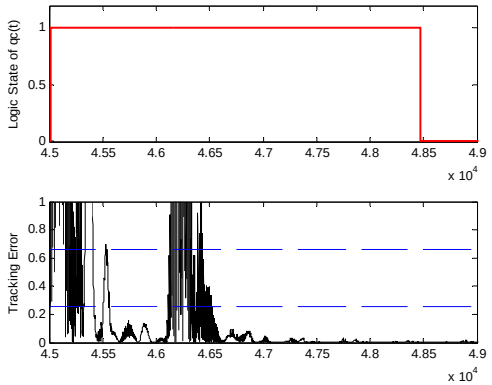


Figure 7.22. Logic state of $q_c(t)$ (top) and unfiltered tracking error with controller thresholds (bottom) provide a visualization of the observed minimum wait time to add a solution to the DMB.

Table 7.9. Comparison of FTC specifications and achieved simulation results.

FTC specification	desired	achieved
Maximum acceptable tracking error under the nominal scenario	0.02	0.004
Maximum acceptable tracking error under a fault scenario	0.05	0.034
Maximum fault detection delay	75 it. (0.15 ref.)	46 it. (0.09 ref.)
Maximum fault identification delay for AUF	100 it. (0.2 ref.)	12 it. (0.02 ref.)
Maximum acceptable reconfiguration time for AKFs	3500 it. (7 ref.)	1086 it. (2.17 ref.)
Minimum observation time before adding a new model to the DMB	750 it. (1.5 ref.)	1948 it. (3.89 ref.)

7.5. Conclusion

Although any successful FTC approach must be tuned to address faults in specific ways to match the requirements of each plant, no existing approach provides neither sufficient flexibility, nor a guideline of how to adjust design parameters in order to fulfill fundamental FTC specifications. In this chapter, all quality indexes used by the supervisor in the proposed architecture were extended in order to provide a sufficient degree of flexibility to the supervisor's response. Then, in order to regulate the response

of the supervisor to match six key FTC specifications, a methodology for the generation of suitable initial conditions for 12 design parameters was presented, along with a look-up table approach for the fine tuning of such parameters during actual implementation. An evaluation of a simulated implementation of the methodology reveals that the initial conditions set for the design parameters successfully adjusted the supervisor, providing a FTC response that fulfills all specifications even when previously unknown nonlinear faults are introduced.

CHAPTER 8 – Stability Concerns

8.1. Introduction

After the introduction of all major elements of the proposed FTC architecture, including the base line adaptive critic nonlinear controller and the FDD & CMD supervisor, it is now possible to better discuss the stability issues involved in certain building blocks of the architecture and the reasoning behind the choices that lead this research to make use of them.

One important aspect of out work is the capability of handling AUFs and the fact that we aim at making as little restrictions as possible on the nature and form of such faults. As a consequence, the dynamics of the plant we desire to make fault tolerant may change greatly form one instant to another, at any instant, and such change may bring previously unknown nonlinearities that may be of paramount importance for its control within the region of the state space traveled by the desired trajectory. It is clear then that to accomplish fault tolerance under such scenario there is need to employ an adaptive controller. Moreover, since the dynamics of the faults under consideration are not known at design time, such adaptation must occur online, updated constantly at every iteration. And finally, since no assumptions are made on the format of the nonlinearities introduced by the AUF, the required adaptive controller should possess universal approximation

capabilities in order to have the power to develop suitable control solutions provided their existence.

8.2. Literature Survey

To this day, Neural Networks have already been applied to a large number of control problems. The existence of training algorithms that can be used for online adaptation and their known universal approximation capability make them strong candidates for a base line adaptive controller for the FTC problem we outlined previously. However, stability and convergence of NN controllers for FTC applications remains an issue. The goal of this section is to explore and discuss some of the prominent research conducted on stable and convergent NN architectures, taking special focus on ACDs, and analyze them for their capability of handling AUFs.

8.2.1. Adaptive critic control with optimal solution convergence guarantee

Using a similar approach than used previously for Q-learning [79], Liu and Balakrishnan [80] approached the online training of the action and critic neural networks in a throughout mathematical study leading to the necessary conditions for the neural network to converge to proven optimal solution.

In their study they have limited themselves to the regulation problem of fully observable linear time-invariant, discrete, multivariable systems. In addition, an initial stabilizing linear controller is assumed to exist before the adaptation takes place.

Furthermore, a full accurate model of the plant is assumed to be available, the reason why the convergence of an IdNN or its effect on the convergence of the remaining two NNs was not studied. Such restrictions allowed the authors to explore in detail the adaptation equations of the AcNN and CrNN of a DHP-equivalent adaptive critic design.

Two conditions were identified as necessary to guarantee the convergence of the NNs when trained together in an iterative manner. In particular, one of the conditions that focuses specifically on the AcNN was shown to be relaxed by the addition of a learning rate parameter. The formulation makes clear that adjusting such learning rate is capable of directly affecting the convergence of the method. However, its determination is linked to the previously mentioned restrictions of the study.

As a final result, it was also shown that the ultimate two conditions can be re-written in the form of the well known discrete algebraic Riccati equation and the process to adapt such networks in the direction of the optimal solution is equivalent to the iterative method to solve the aforementioned equation.

Although an interesting result to the field of ACD in the sense that it confirms the soundness of the approach in its basic sense, it provides no means of extension to the control of systems with nonlinear dynamics as several of the discarded higher order derivatives would have to be considered and further interaction between the NNs could provide training deadlocks. More important to its application to FTC, it does not consider the fact that an IdNN must be adapted concomitantly in the event of faults, nor the implications of not possessing precise the plant's dynamics information at all times.

8.2.2. Practical stability issues with adaptive inverse mapping control

This approach focuses on the application of an online adapting algorithm to generate an inverse map to be applied in parallel to a predetermined sub-optimal controller with the goal of increasing tracking performance. Chen and Chang [81] have explored some practical stability issues related to the implementation of such control architecture to the control of nonlinear time-invariant discrete-time systems. Although the discussion applies to any other NN structure used to generate such maps, the published work focuses on a Cerebellar Model Articulation Controller (CMAC), which is essentially an adaptable look-up-table method.

Fundamental to the approach is the sub-optimal stabilizing controller (e.g. a PID controller) designed off-line and fully capable of generating an acceptable control solution to the plant. Once inserted however, CMAC takes over the main part of the control action, leaving the off-line computed controller to address to fine-tuning, leading to a reported great increase in tracking performance. The architecture that combines the sub-optimal stabilizing controller and the CMAC can be seen in Figure 8.1. It is interesting to note that the formulation of the method requires yt(t+1) to be available at time t, a requirement shared by the ACD formulation presented in this thesis. Through a series of simulation experiments, Chen and Chang studied the effects on the CMAC adaptation stability of the aggressiveness of the off-line designed controller, the CMAC learning rate and its quantization and generalization parameters.

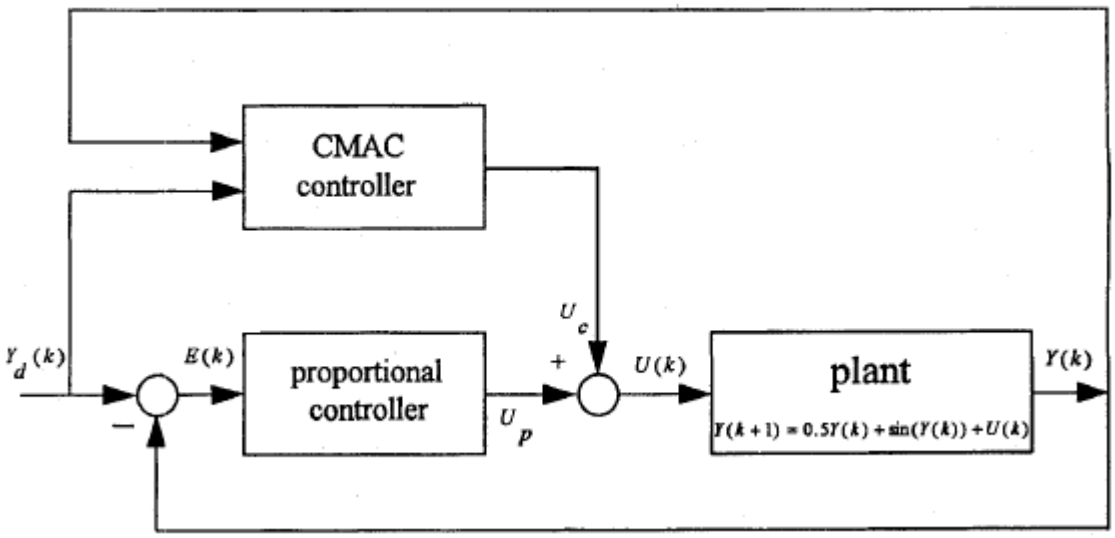


Figure 8.1. The CMAC control system.

As a conclusion, it was reported that the major cause of instability in the online learning of CMAC in an inverse mapping architecture rests on its continued training after the tracking error has been reduced. A deadzone was then created around the inverse dynamics identification error, effectively stopping training after a certain performance is reached. Although the approach was effective in simulations to prevent the training to diverge, the authors raise two difficulties with its implementation. First, it can be difficult to determine when to stop CMAC learning to achieve optimal or near-optimal tracking error. Second, once the CMAC stops learning it becomes incapable of responding to further changes in the reference.

One could implement such deadzone approach to any NN control architecture, including the ACD architectures in the work presented in this thesis. However, due to constant interaction between the three NNs in a full ACD, instability may still occur even after training is stopped. Furthermore, when faults are capable to change the dynamics of a system at any time, the question of when to stop adaptation and when to resume it

becomes even more complex. Moreover, the approach proposed in the literature that involves off-line experimentation for the determination of the deadzone size become infeasible when unknown faults are considered.

8.2.3. Adaptive control solution for systems with stochastic uncertainties with guaranteed signal boundedness in probability and almost sure convergence

This approach focuses on the regulation problem of SISO systems in strict-feedback form subject to time-varying uncertainties. Such uncertainties are assumed to be linearly parameterized , where the unmeasured parameters are generated by stochastic differential equations. The work developed by Arslan and Basar [82] builds up on previous work extending stability proofs to additive and multiplicative uncertainties by using non deterministic notions of stability and performance.

As for the strict-feedback controller itself, although it is perceived that an optimal solution to the problem can only be achieved by a stochastic nonlinear adaptive controller, the literature so far has limited itself to tackle the stability issues of simpler linear in the parameter adaptive controllers which lead to tractable, however sub-optimal, controllers.

It is important to note that although such formulation shares with faults their stochastic nature, the stochastic uncertainties here discussed are well described in structure and operate over systems with fully known dynamics. On the other hand, the work developed in this area holds similarities to the application of ACD in the presented work in the sense that the regulation efficiency is measured by a “risk-sensitive cost”, a

sum of all future regulation errors equivalent to the HBJ equation. Having said that, it is important to note that for the stated problem proof of boundedness in probability and asymptotically zero convergence has been developed, provided that all random sources of parameter uncertainty are independent and that all sources of parameter uncertainty vanish after the origin is reached. Clearly such conditions cannot be satisfied in the field of FTC since faults are prone to interact with each other leading to consequences different than their additive effect and also cannot be expected to stop affecting the system after its effect has been dealt with by the controller. Nevertheless, stochastic notions of stability and performance may hold the key to providing sound guarantees in the face of unknown fault scenarios.

8.2.4. Asymptotically stable Hamilton-Jacobi neural network control for constrained systems

Outside of the ACD scope, Hamilton-Jacobi equations have also been applied with reported success to other neural control applications. One example of such is the work of Lewis and Abu-Khalaf [83], which pertains to the offline design of static nonlinear neural controllers for constrained systems. In particular, their work focused on linear [83] and nonlinear [84] systems subject to input saturation, constrained states and limited disturbances. For such, a state feedback controller was devised through successive iterations between the AcNN and the Hamilton-Jacobi equation.

In order to guarantee asymptotically stability in this particular design, the nonlinear dynamics of the system are assumed to be known and to be Lipschitz continuous. Furthermore, an initial stabilizing control is required as the neural network

solution development is restricted to a compact set within the initial controller's asymptotic stability region.

The capability to generate asymptotically convergent controllers to nonlinear systems under the effect of disturbances and constrains can be interesting to this thesis research as a passive FTC approach or offline refinement tool to known faults control. However, it is important to spell out that this methodology cannot tackle nonlinear systems that can make some of the interactive equations unsolvable, even though they are controllable. Furthermore, although designed to address constrained systems, a common effect of faults on healthy sub-systems, the method only provides simulation results that show that with its application the states are less likely to go over the constrains bounds than the original stable controller.

8.2.5. Globally Convergent ACD for stable linear systems

While still in its very early stages, research on proven stable and convergent online adaptation approaches for ACDs has already produced some promising preliminary results. In particular, we call for attention the work of Murray, Cox, Saeks and Lendaris [42] developed in collaboration between the academic and corporate organizations. The focus of such work was the development of a globally convergent nonlinear ACD (referred as an approximate dynamic programming architecture) to be applied to the very practical problem of designing of a controller for the autolander of the NASA X-43 research aircraft, without a priori knowledge of its flight dynamics.

It is important to note though that although the controller is nonlinear in its formulation, its stability and convergence were only proven to be fully applicable for the control of stabilizable time-invariant linear systems. Focusing on linear systems simplified the approach taken in two significant points. The first point deals with the off-line training of the controller. Such training relies on a training set of input-output of the plant and therefore can only cover a limited portion of the state space. For nonlinear systems, any convergence proof following the presented methodology remains true only for the space covered by the training set, however for a linear plant it is possible to extend the result of a bounded set of training points to all state space, making the results global. The second simplification point rests on the fact that for a linear plant the proposed training algorithm reduces itself to the Newton-Raphson iterative method for solving the Riccati Equation.

Furthermore, it is also necessary that the states of the plant be available for feedback and that, in order for no information at all of the dynamics of the plant be necessary, the plant must be augmented with a known pre-compensator at the cost of increasing its dimensionality. Finally, the plant is also assumed to be exponentially stable at the origin.

With all restriction mentioned, the proof of global convergence follows through a Liapunov approach that guarantees the decay of both controller error signal and convergence of the cost-to-go estimate. Since the result provides a decay statement only, it is implied also that a stabilizing cost functional / control law pair be pre-designed in order to serve as initialization parameters for the algorithm. What makes it possible to apply the Liapunov-based approach to an unknown system is the construction of a linear

model of the plant based on n input-output pairs collected from the plant, where n is equal to the number of dimensions of the plant. Clearly though, this identification process crucial to the controller development must be carried out off-line.

Provided that the plant dynamics are known and that all other assumption are observed, the authors argue that it is possible to apply the same off-line controller design with guaranteed convergence for nonlinear systems, although the solution to the interactive procedure for controller and critic design might not have a solution as tractable as the one for the Riccati Equation. From the perspective of FTC, perhaps the greatest limitation however rests on the fact that the controller design procedure must be carried out off-line, providing no means to manage time varying systems.

8.3 Handling the Stability Concerns

The goal of this section was not to list all work develop on the subject of stability and convergence of NNs in control, but to cover enough relevant recent achievements of some of the leaders in the field to paint a picture of the current state-of-the-art. In so doing, we have shown that significant results have been achieved, producing NN controllers capable of achieving their goals with both stability and guaranteed convergence. However, such results come with severe limitations on the NN capabilities and restrictions on the problems they are capable of tackling. In particular, we have discussed results that are limited to linear plants, nonlinear plants with known dynamics and very strict and limited parameter variation, and learning algorithms that are only applicable off-line.

Although such restrictions may seem overly limiting, it is still possible today to make use of some proven stable NN control architectures in conjunction with the proposed Supervisor in order to design FTC solutions, provided the system for which the FTC is designed and its goals are equally restricted. For instance, one could implement the previously discussed results in [42] to develop control solutions for a linear system and a series of linear AKF and use the proposed supervisor to switch between those. The stability and convergence results of [42] could then be extended to the switched system using an approach such as the one developed in [28], or perhaps by extending the concepts of stochastic stability presented in [82].

However, our major goal is to study the feasibility of a FTC solution capable of handling faults that not only can be nonlinear, but also are unknown at design time. Online training and the ability to respond to AUF are fundamental capabilities that we seek to achieve. For that, we make use of NNs as the building blocks of our nonlinear controllers due to the existence of online training algorithms and their universal approximator property [85]. And in order to manage the unknown dynamics with potential elevated degree of complexity, we have chosen to implement NN control in the most advanced ACD architecture, GDHP. And although no complete stable and convergent approach exists today to tackle the objectives we have set, it is important to make clear that the developed architecture can also be used on smaller scale in conjunction with more restrictive adaptive controllers for projects with narrower goals.

CHAPTER 9 – Conclusion and Future Work

9.1. Conclusion

The presented work has demonstrated that the implementation of a synergistic combination of a reconfigurable controller and a FDD/CMD supervisor based on three distinct quality indexes generates an efficient and reliable FTC architecture. Based on a multiple model approach, the proposed architecture is centered on an intelligent Dynamic Model Bank for fault models and control solutions. The application of adaptive critic designs as reconfigurable controllers is shown to give the hierarchical algorithm the degree of flexibility required to deal with both abrupt and incipient unknown changes in the plant dynamics due to faults. The proposed supervisor system is used to accelerate the convergence of the method by loading new initial conditions to the GDHP when the plant is affected by a known abrupt fault. Moreover, the developed FDD decision logic is capable of recognizing new fault scenarios and assimilating them on-line to the DMB, along with parameters for the corresponding controller. The introduction of the weight quality index has made possible to distinguish between faults in the plant and controller malfunctions caused by online training divergence or local minima convergence. Furthermore, the Dynamic Model Bank was successfully used to generate new initial conditions to the neural network training that improve their efficacy as the supervisor autonomously acquires more nonlinear models of the plant under healthy and diverse

faulty scenarios. Directly affecting the response of the supervisor and its manipulation of the DMB, a methodology for initializing and tuning distinct parameters of the quality indexes that allows for application-specific key FTC specifications to be achieved was presented. Finally, a series of key steps that form the basis for the fault development information extraction module capable of providing the probability of occurrence of future faults to the user, were also addressed in this report.

9.2. Future work

9.2.1. Improvements on supervisor – unified decision logic

As presented earlier, the supervisor is instrumental in managing a series of key features in the proposed FTC architecture. By manipulating the DMB, the supervisor is able to reduce the reconfiguration time of abrupt known faults as well as prevent controller malfunctions from developing to the point of degrading the plant's tracking performance significantly. However, neither benefit can take place as fast as or as accurately as the FDD and CMD decision logics can ascertain the current status of the plant. Performing both FDD and CMD in a way to match the specifications and requirements of each plant is critical to the success of the whole FTC architecture. Moreover, it is fundamental that the supervisor be capable of distinguishing between plant faults and controller malfunctions since the counteraction for each scenario is different.

At the present stage, at every interaction CMD is performed first and then, provided no controller malfunctions are detected, the FDD decision logic is used to

evaluate the status of the plant. Although justifiable, this sequential approach may cause additional delay on the detection of faults and/or controller malfunctions when they occur close to each other in time. More importantly, the currently implemented approach does not account for all possible transitions from states within one decision logic to states within the other. For example, if a fault in the plant occurs after a controller malfunction is detected, it will only be detected by the FDD decision logic once the referred controller malfunction is addressed properly.

Combining both decision logics into a single structure requires the analysis of all possible 24 directional transitions between the 3 states of the CMD decision logic and the 4 states of the FDD decision logic. Once all transitions are analyzed for their significance and assigned different actions (e.g., add and switch) as required, a new paradigm must be used to express such connections since the graphical tools used so far will reach their representational limit. With benefits to the response of the whole FTC architecture, we feel that such supervisor improvements are a critical avenue for future work.

9.2.2. Ultimate development of the Fault Development Rule Extraction module

As the first step in the direction of developing the final component of the proposed FTC architecture capable of providing the user with fault development probabilities, a novel method for linguistic rule extraction was presented in Chapter 6. Making use of an innovative TSD representation, a multiple objective rule search and temporal fuzzyfication, rules such as the following example can be extracted directly from raw sensor data:

IF T_1 is high AND F_1 is medium AND T_2 is medium AND F_2 is low **THEN** after a medium delay T_3 will be high.

Although many of the key aspects have already been tackled, such as temporal information extraction and representation, some additional work must be taken to reach ultimate fault development rules such as:

IF Fault 15 (valve seal compromised) is active AND input #5 remains very low AND reference #2 remains high, **THEN** Fault 23 (loss of valve actuator) will have a 85% chance of occurrence after a delay of 30 to 45 minutes.

In particular, there is a need to add a probabilistic estimate of occurrence in the model of the consequent (predicted fault). Such a probability must account not only for the chance of a fault to occur or not, but also for the probability of it taking place within the indicated time frame.

Moreover, in order to be able to infer on the chance of occurrence of a fault, it becomes much more meaningful to take into consideration the number of independent occurrences of the fault, as opposed to the time spent by the plant in all occurrences of the fault as currently being measured by the first metric. In the TSD representation, an isolated fault occurrence, given a particular set of antecedents stated by a rule, corresponds to data points moving over time into the second quadrant, remaining inside of it for a period of time and leaving it. It is then possible to call dynamic behaviors of

data points within the TSD a *trip*. A trip into and out of the second quadrant ($trip_2$) corresponds to an occurrence of the fault as dictated by the rule, while a trip involving the fourth quadrant ($trip_4$) corresponds to an event marked by a fault warning being issued (antecedents are observed) without being followed by the occurrence of the fault after the stated delay (consequent not observed). Therefore, a more meaningful metric to measure rule inaccuracy for the extraction of fault development information might be Equation (7.1) in substitution of Equation (6.4).

$$m_1^{'} = \begin{cases} \dfrac{trip_4}{trip_2 + trip_4}, & trip_2 + trip_4 > 0 \\ 1, & \text{otherwise} \end{cases}, \tag{7.1}$$

A final point worth mentioning in this plan of future work pertains to the temporal fuzzyfication. While in general a linguistic fuzzy term might be ideal for providing a human with a fuzzy delay, for FTC application it is crucial to clearly state the time frame another fault is expected to occur. Changing the fuzzy linguistic term to a specific range involves, in a first level, the use of a different membership function shape for the temporal fuzzyfication. More advanced and innovative approaches such as probabilistic temporal histograms might also be considered in order to better transmit to the user the probabilistic distribution of the chance of occurrence of a fault over time.

9.2.3. Improvements in the initialization procedure of the FTC design parameters

Although the presented methodology for the adjustment of the extended quality indexes produced satisfactory simulation results achieving all desired specifications, there are still points that can be improved, both in the offline initialization and the online tuning procedures. For instance, it could be possible to isolate certain specific design parameters, such as the filter parameters of the controller quality index, and produce worst case calculations of their impact on the affected specifications based on the previous choice of threshold levels. Although such calculations would necessarily produce conservative estimates of the impacts of different values, they may represent even more computationally efficient ways to calculate filter parameters, which are proposed here to be obtained through successive trials using the data obtained during the simulation of the synthetic linear fault set.

Another potential point of improvement rests on the indentification quality index. In the current architecture, $q_i(t)$ is used both in the FDD decision logic to determine when the system is operating in a known scenario *and* to determine which specific dynamics is active. Therefore, while adjusting for the maximum fault identification delay for AKFs, the discernment among fault scenarios is also being affected. The problem with this approach is that it is only possible to clearly determine which of the known scenarios is currently active by applying $q_i(t)$ filter parameters within a certain range. For example, if γ_i^d is set too high, models that may produce low prediction error in only certain portions of the state space, may be temporarily indicated as match to the current dynamics and will deteriorate the base-line controller response when switched to an incorrect solution.

Therefore, it may be interesting to study in future work the possibility of separating the model identification task from the identification quality index with the goal of increasing is specificity and allowing greater flexibility of its parameters. Such a task would not be trivial though, since there would be need to somehow ensure that $q_i(t)$ would only determine that a known fault had occurred after the independent measure determined that the current model had been correctly isolated.

BIBLIOGRAPHY

1. K. Åström, P. Albertos, M. Blanke, A. Isidori, W. Schaufelberger and R. Sanz (Eds.), *Control of Complex Systems*, Springer, London, UK, 2001.
2. M. Blanke, R. Izadi-Zamanabadi, S. Bøgh and C. Lunau, "Fault-tolerant control systems – a holistic view," *Control Engineering Practice*, vol. 5, no. 5, pp. 693-702, 1997.
3. M. Blanke, R. Izadi-Zamanabadi and T. Lootsma, "Fault monitoring and re-configurable control for a ship propulsion plant," *International Journal of Adaptive Control Signal Processing*, vol. 12, pp. 671-688, 1998.
4. M. Blanke, M. Staroswiecki and N. Wu, "Concepts and methods in fault-tolerant control," *Proc. American Control Conference*, pp. 2606-2620, 2001.
5. G. Yen and P. Meesad, "An effective neural-fuzzy paradigm for machinery condition health monitoring," *IEEE Trans. System, Man and Cybernetics, Part B: Cybernetics*, vol. 31, no. 4, pp. 523-536, 2001.
6. Y. Diao, "Fault tolerant systems design using adaptive estimation and control," *PhD dissertation*, The Ohio State University, Columbus, OH, 2000.
7. D. Prokhorov, R. Santiago and D. Wunsch, "Adaptive critic designs: a case study for neurocontrol," *Neural Networks*, vol. 8, no. 9, pp. 1367-1372, 1995.
8. A. Kokkinaki and K. Valavanis, "Error specification, monitoring and recovery in computer-integrated manufacturing: an analytic approach," *Proc. IEE Conference on Control Theory and Applications*, pp. 499-508, 1996.
9. S. Qin and W. Li, "Detection and identification of faulty sensors with maximized sensitivity," *Proc. American Control Conference*, pp. 613-617, 1999.
10. P. Kabore and H. Wang, "Using an equivalent feedback control of the residuals for fault detection and identification," *Proc. Conference on Decision and Control*, pp. 4466-4471, 1999.

11. R. Isermann and P. Ballé, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control Engineering Practice*, vol. 5, no. 5, pp. 709-719, 1997.
12. S. Edwards, A. Lees and M. Friswell, "Fault diagnosis of rotating machinery," *Shock and Vibration Digest*, vol. 30, no. 1, pp. 4-13, 1998.
13. H. Aldridge, "Robot position sensor fault tolerance," *NASA Technical Memorandum 110314*, Langley Research Center, Hampton, VA, 1997.
14. S. Leonhardt and M. Ayoubi, "Methods of fault diagnosis," *Control Engineering Practice*, vol. 5, no. 5, pp. 683-692, 1997.
15. D. Juričić, A. Žnidaršić and D. Füssel, "Generation of diagnostic trees by means of simplified process models and machine learning," *Engineering Applications of Artificial Intelligence*, vol. 10, no. 1, pp. 15-29, 1997.
16. P. Frank and B. Köppen-Seliger, "New developments using AI in fault diagnosis," *Engineering Applications of Artificial Intelligence*, vol. 10, no. 1, pp. 3-14, 1997.
17. Y. Zhang and J. Jiang, "An interacting multiple model based fault detection, diagnosis and fault-tolerant control approach," *Proc. Conference on Decision and Control*, pp. 3593-3598, 1999.
18. Y. Diao and K. Passino, "Intelligent fault tolerant control using adaptive schemes and multiple model," *Proc. American Control Conference*, pp. 2854-2859, 2001.
19. S. Bøgh, "Fault tolerant control systems – a development method and real-life case study," *PhD dissertation*, Aalborg University, Aalborg, Denmark, 1997.
20. M. Mahmoud, J. Jiang and Y. Zhang, "Analysis of the stochastic stability of fault tolerant control systems," *Proc. Conference on Decision and Control*, pp. 3188-3193, 1999.
21. W. Rugh, "Analytical framework for gain scheduling," *IEEE Control Systems Magazine*, vol. 11, no 1, pp. 79-84, 1991.
22. C. Lopez-Toribio and R. Patton, "Takagi-Sugeno fuzzy fault-tolerant control of a non-linear system," *Proc. Conference on Decision and Control*, pp. 4368-4373, 1999.

23. I. Konstantopoulos and P. Antsaklis, “An eigenstructure assignment approach to control reconfiguration,” *Proc. IEEE Mediterranean Symposium of Control and Automation*, pp. 328-333, 1996.
24. Y. Zhang and J. Jiang, “An interacting multiple-model based fault tolerant detection, diagnosis and fault-tolerant control approach,” *Proc. Conference on Decision and Control*, pp. 3593-3598, 1999.
25. S. Kanev and M. Verhaegen, “A bank of reconfigurable LQG controllers for linear systems subjected to failures,” *Proc. Conference on Decision and Control*, vol. 4, pp. 3684-3689, 2000.
26. I. Konstantopoulos and P. Antsaklis, “An optimization approach to control reconfiguration,” *Dynamics and Control*, vol. 9, no. 3, pp. 255-270, 1999.
27. K. Narendra and S. Mukhopadhyay, “Adaptive control of nonlinear multivariable systems using neural networks,” *Neural Networks*, vol. 7, no. 5, pp. 737-752, 1994.
28. K. Narendra and O. Driollet, “Adaptive control using multiple models, switching, and tuning,” *Proc. Adaptive Systems for Signal Processing, Communications, and Control Symposium*, pp. 159-164, 2000.
29. K. Narendra and K. Parthasarathy, “Identification and control of dynamical systems using neural networks,” *IEEE Trans. on Neural Networks*, vol. 1, no. 1, pp. 4-27, 1990.
30. E. Sanchez and M. Bernal, “Adaptive recurrent neural control for nonlinear systems tracking,” *IEEE Trans. on Systems, Man and Cybernetics*, vol. 30, no. 6, pp. 886-889, 2000.
31. E. Kosmatopoulos, M. Christodoulou and P. Ioannou, “Dynamical neural networks that ensure exponential identification error convergence,” *Neural Networks*, vol. 10, no. 2, pp. 299-314, 1997.
32. G. Kulawski and M. Brdyś, “Stable adaptive control with recurrent networks,” *Automatica*, vol. 36, pp. 5-22, 2000.
33. C. Hwang and C. Lin, “A discrete-time multivariable neuro-adaptive control for nonlinear unknown dynamic systems,” *IEEE Trans. on Systems, Man and Cybernetics*, vol. 30, no. 6, pp. 865-877, 2000.

34. S. Ge, C. Wang and Y. Tan, "Adaptive control of partially known nonlinear multivariable systems using neural networks," *Proc. IEEE International Symposium on Intelligent Control*, pp. 292-297, 2001.
35. G. Puskorius and L. Feldkamp, "Automotive engine idle speed control with recurrent neural networks," *Proc. American Control Conference*, pp. 311-316, 1993.
36. K. Narendra, J. Balakrishnan and M. Ciliz, "Adaptation and learning using multiple models, switching and tuning," *IEEE Control Systems Magazine*, vol. 15, no. 3, pp. 37-51, 1995.
37. J. Boskovic, S. Liu and R. Mehra, "On-line failure detection and identification (FDI) and adaptive reconfiguration control (ARC) in aerospace applications," *Proceedings of the American Control Conference*, pp. 2625-2626, 2001.
38. D. Filev and T. Larsson, "Intelligent adaptive control using multiple models," *Proc. IEEE International Symposium on Intelligent Control*, pp. 314-319, 2001.
39. R. Murray-Smith and T. Johansen (Eds.), *Multiple Model Approaches to Modeling Control*, Taylor and Francis, Basingstoke, UK, 1997.
40. H. Pei and B. Krogh, "Stability regions for systems with mode transitions," *Proc. American Control Conference*, pp.4834-4839, 2001.
41. K. Narendra and S. Mukhopadhyay, "Adaptive control using neural networks and approximate models," *IEEE Trans. on Neural Networks*, vol. 8, no. 3, pp. 475-485, 1997.
42. J. Murray, C. Cox, R. Saeks and G. Lendaris, "Globally convergent approximate dynamic programming applied to an autolander," *Proc. American Control Conference*, pp. 2901-2906, 2001.
43. D. Prokhorov and D. Wunsch, "Adaptive critic designs," *IEEE Trans. on Neural Networks*, vol. 8, no. 5, pp. 997-1007, 1997.
44. G. Venayagamoorthy, R. Harley and D. Wunsch, "Comparison of a heuristic programming and a dual heuristic programming based adaptive critics neurocontroller for a turbogenerator," *Proc. International Joint Conference on Neural Networks*, vol. 3, pp. 233-238, 2000.

45. P. Werbos, “Stable adaptive control using new critic designs,” available at xxx.lanl.gov/abs/adap-org/9810001, 1998.
46. P. Werbos, “Backpropagation though time: what it does and how to do it,” *Proc. IEEE*, vol. 78, no. 10, pp. 1550-1560,1990.
47. O. Jesús and M. Hagan, “Backpropagation through time for a general class of recurrent network,” *Proc. International Joint Conference on Neural Networks*, pp. 2638-2643, 2001.
48. G. Lendaris and C. Paintz, “Training strategies for critic and action neural networks in dual heuristic programming method,” *Proc. International Joint Conference on Neural Networks*, pp. 712-717, 1997.
49. R. Williams, “Training recurrent networks using the extended Kalman filter,” *Proc. International Joint Conference on Neural Networks*, pp. 241-246, 1992.
50. M. Livstone, J. Farrell and W. Baker, “A computationally efficient algorithm for training recurrent connectionists networks,” *Proc. American Control Conference*, pp. 555-561, 1992.
51. G. Lendaris, C. Paintz and T. Shannon, “More on training strategies for critic and action neural networks in dual heuristic programming method,” *Proc. IEEE International Conference on Systems, Man and Cybernetics*, pp 3067-3072, 1997.
52. H. Qi and L. Qi, “Deriving sufficient conditions for global asymptotic stability of delayed neural networks via nonsmooth analysis,” *IEEE Transactions on Neural Networks*, vol. 15, no. 1, pp. 99-109, 2004.
53. Y. Xia and J. Wang, “A recurrent neural network for nonlinear convex optimization subject to nonlinear inequality constraints,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 7, pp. 1385-1394, 2004.
54. G. Stephanopoulos and C. Han, “Intelligent systems in process engineering: a review,” *Computers & Chemical Engineering*, vol. 20, no. 6-7, pp. 743-791, 1996.
55. E. Oshima, “Computer aided plant operation,” *Computers & Chemical Engineering*, vol. 7, no. 4, pp. 311-329, 1983.

56. C. Collewet, G. Rault, S. Quellec and P. Marchal, "Fuzzy adaptive controller design for the joint space control of an agricultural robot," *Fuzzy Sets and Systems*, vol. 99, no. 1, pp. 1-25, 1998.
57. K. Cheng and J. Chen, "A fuzzy-nets training scheme for controlling nonlinear systems," *Computers & Industrial Engineering*, vol. 31, no. 1-2, pp. 425-428, 1996.
58. R. Rhinehart and P. Murugan, "Improve process control using fuzzy logic," *Chemical Engineering Process*, vol. 91, no. 11, pp. 60-65, 1996.
59. C. Cimander, T. Bachinger and C. Mandenius, "Integration of distributed multi-analizer monitoring and control in bioprocessing based on a real-time expert system," *Journal of Biotechnology*, vol. 103, no. 3, pp. 327-248, 2003.
60. D. Fonseca and G. Knapp, "An expert system for reliability centered maintenance in the chemical industry," *Expert Systems with Applications*, vol. 19, no. 1, pp. 45-57, 2000.
61. A. Kordon, "Hybrid intelligent systems for industrial data analysis," Seminar given at MCEC meeting, Sep. 2002.
62. K. Astrom and T. Soderstrom, "Uniqueness of the maximum likelihood estimates of the parameters of an ARMA model," *IEEE Transactions on Automated Control*, vol. 19, no. 6, pp. 769 – 773, 1974.
63. C. Tsai and S. Wu, "A study for second-order modeling of fuzzy time series," *Proceedings of the IEEE International Fuzzy Systems Conference*, vol. 2, pp. 719-725, 1999.
64. N. Sisman and F. Alpaslan, "Temporal neurofuzzy MAR algorithm for time series data in rule-based systems," *Proceedings of the International Conference on Knowledge-Based Intelligent Electronic Systems*, vol.2, pp. 316-320, 1998.
65. M. Last, Y. Klein and A. Kandel, "Knowledge discovery in time series databases," *IEEE Transactions no System, Man and Cybernetics*, vol. 31, no. 1, pp. 160-169, 2001.
66. B. Carse and T. Fogarty, "Evolutionary learning of temporal behavior using discrete and fuzzy classifier systems," *Proceedings of the International Symposium on Intelligent Control*, pp. 183-188, 1995.

67. B. Carse, T. Fogarty and A. Munro, "Distributed adaptive routing control in communication networks using a temporal fuzzy classifier system," *Proceedings of the IEEE International Conference on Fuzzy Systems*, vol. 3, pp. 2201 – 2207, 1996.
68. N. Sharma, "Metrics for evaluation of the goodness of linguistic rules," MS Thesis, Oklahoma State University, School of Chemical Engineering, 2003.
69. G. Yen and H. Lu, "Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 253-274, 2003.
70. F. Chen, Z. Chen and Z. Jiao, "A novel processing for multiple gases detection," *Proceedings of the World Congress on Intelligent Control and Automation*, vol. 3, pp. 2186-2189, 2002.
71. M. South, C. Bancroft, M. Willis and M. Tham, "System identification via genetic programming," *Proceedings of the UKACC International Conference on Control*, vol. 2, pp. 912-917, 1996.
72. C. Fonseca and P. Fleming, "Genetic algorithms for multiobjective optimization: formulation, discussion and generalization," *Proceedings of the International Conference on Genetic Algorithms*, pp. 416-423, 1993.
73. E. Mosca and T. Agnoloni, "Switching supervisory control based on controller falsification and closed-loop performance inference," *Journal of Process Control*, vol. 12, no.4, pp. 457-466, 2002.
74. E. Wilson, C. Lages and R. Mah, "Gyro-based maximum-likelihood thruster fault detection and identification" *Proceedings of the American Control Conference*, vol. 6, pp. 4525 - 4530, 2002.
75. C. Bonivento, A. Paoli and L. Marconi, "Fault-tolerant control of the ship propulsion system benchmark," *Control Engineering Practice*, vol. 11, no.5, pp. 483-492, 2003.
76. W. Liu, "An on-line expert system-based fault-tolerant control system," *Expert Systems With Applications*, vol. 11, no. 1, pp. 59-64, 1996.

77. R. Izadi-Zamanabadi and M. Blanke, "A ship propulsion system as a benchmark for fault-tolerant control," *Control Engineering Practice*, vol. 7, no. 2, pp. 227-239, 1999.
78. X. Liu and A. Dexter, "Fault-tolerant supervisory control of VAV air-conditioning system," *Energy and Buildings*, vol. 3, no. 4, pp. 379-389, 2001.
79. S. J. Bradtke, B. E. Ydstie, and A. G. Barto, "Adaptive linear quadratic control using policy iteration," *Proceedings of the American Control Conference*, pp. 3475-3479, 1994.
80. X. Liu and S. Balakrishnan, "Convergence analysis of adaptive critic based optimal control," *Proceedings of the American Control Conference*, pp. 1929-1933, 2000.
81. F. C. Chen and C. H. Chang, "Practical stability issues in CMAC neural network control system," *IEEE Transactions on Control Systems Technology*, vol. 4, no.1, pp. 86-91, 1996.
82. G. Arslan and T. Basar, "Disturbance attenuation controller design for strict-feedback systems with structurally unknown dynamics," *Automatica*, vol. 37, no. 8, pp. 1175-1188, 2001.
83. F. Lewis and M. Abu-Khalaf, "A Hamilton-Jacobi setup for constrained neural network control," *Proceedings of the IEEE International Symposium on Intelligent Control*, pp. 1-15, 2003.
84. F. Lewis and M. Abu-Khalaf, "Nearly optimal state feedback control of constrained nonlinear systems using a neural network approach," *IFAC International Conference on Intelligent Control Systems and Signal Processing*, plenary paper, 2003.
85. F. L. Lewis, S. Jagannathan and A. Yesildirek, *Neural Network Control of Robot Manipulators and Nonlinear Systems*, Taylor & Francis, 1999.

VITA

Pedro Gerbase de Lima

Candidate for the Degree of

Doctor of Philosophy

Thesis: A SUPERVISED FAULT TOLERANT CONTROL ARCHITECTURE FOR NONLINEAR SYSTEMS

Major Field: Electrical and Computer Engineering

Biographical:

Education: Graduated from The University of Sao Paulo with the degree of Bachelor of Science in Electrical Engineering in December, 2000. Completed the Requirements for the Ph.D. degree at Oklahoma State University in December, 2005.

Experience: employed at the University of Sao Paulo in the Biomedical Engineering Laboratory as an undergraduate research intern from 1999 to 2000. Later, from 2001 to 2005, employed at the Intelligent Systems and Control Laboratory at the Department of Electrical and Computer Engineering, Oklahoma State University as a graduate research assistant. During the Fall of 2005, employed at Oklahoma State University as an instructor for an electrical engineering senior level course.

Professional Memberships: Student member of the Institute of Electrical and Electronics Engineers (IEEE) since 2001; member of the IEEE Computational Intelligence Society since 2005.

Name: Pedro Gerbase de Lima Date of Degree: December, 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma, USA

Title of Study: A SUPERVISED FAULT TOLERANT CONTROL ARCHITECTURE FOR NONLINEAR SYSTEMS

Pages in Study: 199 Candidate for the Degree of Doctor of Philosophy

Major Field: Electrical and Computer Engineering

Scope: The growing complexity of physical plants and control missions inevitably leads to increasing occurrence, diversity and severity of faults. Availability, defined as the probability that a system or equipment will operate satisfactory and effectively at any point of time, becomes a factor of increasing importance. Fault Tolerant Control (FTC) is a field of research that aims to increase availability and reduce the risk of safety hazards and other undesirable consequences by specifically designing control algorithms capable of maintaining stability and/or performance despite the occurrence of faults. This report presents a novel FTC solution based on a hierarchical architecture in which an adaptive critic controller is overseen by a supervisor managing a dynamic model bank of fault solutions.

Findings and Conclusions: The presented work has demonstrated that the implementation of a synergistic combination of a reconfigurable controller and a fault diagnosis and controller malfunction detection supervisor based on three distinct quality indexes generates an efficient and reliable FTC architecture. The application of adaptive critic designs as reconfigurable controllers is shown to give the hierarchical algorithm the degree of flexibility required to deal with both abrupt and incipient unknown changes in the plant dynamics due to faults. The proposed supervisor system is used to accelerate the convergence of the method by loading new initial conditions to the controller when the plant is affected by a known abrupt fault. Moreover, the developed fault diagnosis decision logic is capable of recognizing new fault scenarios and assimilating them on-line to the dynamic model bank, along with parameters for the corresponding controller. The introduction of the weight quality index has made possible to distinguish between faults in the plant and controller malfunctions caused by online training divergence or local minima convergence. In order to achieve application-specific key FTC specifications, a methodology for initializing and tuning twelve distinct parameters of the quality indexes was also developed. Finally, a series of key steps that form the basis for the fault development information extraction module capable of providing the probability of occurrence of future faults to the user, are also included in this report.

Advisor's Approval: Dr. Gary G. Yen