ABSTRACT

Development and Implementation of a Multi-Agent System for Intelligent Optimized Power Plant Control

Jason D. Head, M.S.E.C.E.

Mentor: Kwang Y. Lee, Ph.D.

As the demand for electric power grows and regulations on power plant operation become stricter, the size, and therefore complexity, of new power plant units is increasing while the intricacies of the multiple simultaneous processes that take place to generate electricity require tighter control. In order to provide a solution to some of the associated operational challenges arising from this situation, control techniques have been developed to allow optimized power plant control while considering non-fixed operating goals. Each of these techniques is computationally intensive, requiring a distributed, parallel control framework to implement each technique simultaneously in distributed subsystem environments. For these reasons, previous research has studied multi-agent systems as a means to implement such a control system. Therefore, the goal of this thesis is to fully develop a multi-agent system to coordinate and implement these techniques to control a third order fossil fuel power plant model.

Development and Implementation of a Multi-Agent System
for Intelligent Optimized Power Plant Control

by

Jason D. Head, B.S.E.C.E.

A Thesis

Approved by the Department of Electrical and Computer Engineering

____
Kwang Y. Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science in Electrical and Computer Engineering

Approved by the Thesis Committee

____
Kwang Y. Lee, Ph.D., Chairperson

____
Ian Gravagne, Ph. D.

____
Paul Grabow, Ph. D.

Accepted by the Graduate School
May 2012

____
J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACL	Agent Communication Language
ANN	Artificial Neural Network
CC	Coordinated Control
DRNN	Diagonal Recurrent Neural Network
FFPU	Fossil Fuel Power Unit
FFNN	Feedforward Neural Network
FIPA	Foundation for Intelligent Physical Agents
GA	Genetic Algorithm
GUI	Graphical User Interface
HNN	Human Neural Network
HPSO	Hybrid Particle Swarm Optimization
IEEE	The Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LRNN	Layer Recurrent Neural Network
MAS	Multi-Agent System
MACS	Multi-Agent Control System
MPC	Model Predictive Control
PI	Proportional-Integral
PSO	Particle Swarm Optimization
RGA	Relative Gain Array
RTS	Reliability Test System

UDP	User Datagram Protocol
ULDC	Unit Load Demand Curve

ACKNOWLEDGMENTS

I would like to thank Dr. Kwang Y. Lee, as my advisor, for his guidance and encouragement during my graduate studies at Baylor University, and for allowing me the opportunity to work under him. I would also like to thank Dr. Ian Gravagne for his contribution of expertise in control systems to the project and for serving as a graduate committee member, and Dr. Paul Grabow for serving as a graduate committee member and providing invaluable insite. In addition, I would like to thank the National Science Foundation, the Department of Electrical and Computer Engineering at Baylor University, my parents Ronald and Dovie Head, and my grandparents James Nelson and Laura Fae Head for providing financial support during my graduate studies. I would also like to express my deepest appreciation to my parents, Ronald and Dovie Head, my grandparents, James Nelson and Laura Fae Head, and my wonderful wife Nicole whose prayers, encouragement, and support helped me to endure and succeed during my studies at Baylor University. Lastly, I would like to thank my friends and colleagues John Miller, Jason Gomes, and Craig Williams for their contributions to this project.

CHAPTER ONE

Introduction

This chapter describes the problem statement for this thesis by introducing previous work that has led to this extension. The motivation for the methods proposed in previous work is described. This thesis implements, tests, and adds to the methods previously proposed.

1.1 Motivation

In today's electric utility markets, there are many worldwide challenges regarding the generation of electricity. The main challenge facing electric utility companies is to remain competitive in cost to end users while balancing many other financial, regulatory, and operational demands. Financial concerns stem from the complexity of planning for future costs in the production of energy that is driven by many diverse factors [1,2].

Constantly evolving government regulations that potentially increase the cost of producing electricity and change the way power plants are operated create other challenges. The operational challenges faced by utilities are due to the need to increase efficiency in existing plants and to effectively control large, complex plants that are being built in response to the rapidly increasing demand for electric power [3]. To describe these challenges and give motivation for this thesis, the following topics are briefly discussed:

- the financial impact of the varying cost of fuels based on world markets

- the decision to either replace aging plant equipment or to build new plants to maximize on capital investment
- the potential for increasingly stringent regulations to be passed concerning environmental impact and resource management, causing many existing plants to retrofit equipment with costly upgrades
- accommodating cyclic load operation without causing unnecessary stress to plant equipment to make efficient use of resources in power plants, many of which were designed for maximum load operation
- extending equipment life in aging plants to maximize capital investment
- improving energy efficiency, or heat rate, of power plants
- improving the efficiency of fuel combustion to avoid the emission of pollutants
- planning for increased generation capacity to meet the rapidly growing demand while considering the above mentioned issues.

Constantly evolving regulations on the electric power generation industry force electric utility companies to continually adapt the way they operate in order to comply. These considerations also cause increases in operational costs, such as those incurred because of higher taxes on emissions, expensive retrofits meant to reduce pollution, and requirements that mandate the use of more expensive, cleaner burning versions of fuel. Increases in operating costs bring challenges to utilities and have the potential to raise electricity prices to end users, making it more difficult for utility companies to compete in today’s market.

Furthermore, the future cost of fuel needed to operate existing power plants is affected by many global factors, making it hard to predict future prices. The instability in price makes it difficult for utilities to plan electricity generation in existing plants and makes deciding what types of new power plants to build risky. Consequently, this instability complicates financial planning for utilities.

Further complicating financial planning for electric utility companies is the issue of whether to retire and replace existing plants or to make costly upgrades in order to continue using them. This decision is affected by several factors, including the ones mentioned above. Possible regulatory legislation, if passed, could cause existing plants to become too expensive to operate, and other plants may not be able to meet the new requirements even if upgrades were made. Either of these cases would cause plants to be shut down.

Older plants, in particular, are vulnerable, because they were designed before more efficient equipment was available. Modifications required by regulations would be expensive initially, but would cause the plant's operating costs to decrease by reducing the taxes on emissions. The challenge is to decide what initial cost is worth investing to keep an existing plant in operation, while considering how quickly the modifications will pay for themselves.

To complicate matters further, power consumption is growing very rapidly, and is projected to continue growing at such a pace for the foreseeable future. Currently, the worldwide consumption of electric power is expected to increase at an average rate of 2.3 percent per year until the year 2035, making it the fastest growing end-use of energy consumption in the world [4]. To keep up, utilities will have to build new, larger power plants in addition to deciding what to do with the existing ones.

Besides financial concerns, there are many operational challenges that have arisen from a control perspective. Though these issues may be relevant to other plant types, fossil fuel power units (FFPUs) are the most widely used type of plant in the generation of electricity worldwide and are projected to remain as such for at least the next 25 years [4]. For this reason, they have been the focus of research leading to this thesis, and remain the focus herein.

Today, many FFPU's are operated cyclically in a wide-range, load-following manner, where load demands are sent from dispatch centers. This is done to make better use of resources and to reduce operating costs by not generating more power than is needed. Although newer equipment is better suited for this type of operation, many older plants were designed to be efficient while operating at the rated load capacity [5]. In both cases, careful considerations must be taken to avoid putting increased strain on plant equipment, shortening its overall life.

Maximizing the life of an FFPU is important, because the longer an FFPU lasts, the better the return will be on the initial capital investment that was made to build the plant. High stress operation is the main cause of shortened plant life, and is often a result of thermal stress caused by fluctuating steam temperature and pressure. The most severe strain occurs during startup and shutdown, as well as when sudden load variations occur.

These challenges call for the development of more robust control schemes than are currently employed today. Such control schemes should make use of advances in computer-based instrumentation, computational abilities, computer networking, and computational intelligence to allow more flexibility, robustness, and efficiency. Such a control scheme is the focus of this thesis.

1.2 Background

As previously mentioned, FFPUs generate the majority of electric power consumed around the world today and are projected to remain the predominant source for years to come. Therefore, this thesis focuses on solving control problems faced by FFPUs. This section discusses common control practices with respect to these units to provide context to give background for the control methods proposed here.

1.2.1 Current FFPU Situation

With the rising demand for electricity, in conjunction with the issues discussed in the previous section, more is being required of existing plants and new power plants are growing in size. Existing FFPUs must generate more electricity while increasing overall efficiency and reducing pollution. To make matters more difficult, the total combined generation capacity of FFPUs has increased very little compared to the increased demand, meaning that more power is required from FFPUs [1]. This scenario makes control optimization, discussed further in later sections, important for the continued operation of these units.

Furthermore, as new power plants get bigger, they become more complex in terms of the number of interconnected subsystems [6]. More interconnected subsystems make it more difficult to achieve efficient control because of the increased complexity of the interactions among processes. Also, conventional, centralized control approaches are no longer adequate, since the larger power plants have physically distributed subsystems. Furthermore, a failure in any part of a centralized control system can cause the entire control system to fail. For these reasons, a distributed control system is desirable.

1.2.2 Control Schemes

In general, there are three control strategies that have been used by FFPUs to match the unit load demand, or desired power output. These methods are boiler-following control, turbine-following control, and coordinated control. Each of these methods has its own advantages and disadvantages. The details of these methods are discussed to give background for the control techniques used in this thesis.

The boiler-following, or turbine-leading, control strategy matches the power output demand of an FFPU by varying steam flow to cause the turbine to generate more or less power by constraining or releasing the energy stored in the form of steam in the boiler. Releasing steam causes the steam pressure in the boiler to drop. To compensate, fuel burn is increased to vaporize more water in the boiler to maintain the steam pressure setpoint. The advantage of this method is that the plant can be made to change power output quickly, since the turbine responds quickly to changes in steam flow. The disadvantage is that this method causes the steam pressure to be less stable, because the boiler cannot produce steam as fast as the turbine can be made to change power output.

The turbine-following, or boiler-leading, control strategy matches the power output demand by varying fuel combustion, which generates more or less steam based on an increase or decrease in output demand. In response, steam flow to the turbine is varied to regulate the steam pressure in the boiler to match the setpoint value. The advantage of this method is that it produces a more stable steam pressure and temperature response to load changes. The disadvantage is that this method is slower in varying the output power in response to a load change because of the time it takes to generate steam.

Coordinated control (CC) methods attempt to make use of the advantages in the previous two methods, while minimizing their shortcomings. This type of approach uses control logic to simultaneously adjust the steam flow and fuel burn rates to quickly match the load demand while keeping up with the pressure demand in the boiler. To maintain the boiler pressure setpoint, the control logic must anticipate the pressure loss due to a load change so that it can preemptively adjust the fuel burn rate.

Historically, CC methods have made use of fixed, nonlinear functions to determine the setpoints that govern the boiler and turbine control. However, this method is inflexible, because it does not allow process optimization if the operating objectives must be changed. Therefore, to add more flexibility, a reference governor is needed to optimize the control setpoints during operation. The following section discusses a reference governor that has been designed to do this [3].

1.2.3 Reference Governor

The reference governor in [3] was designed to optimize control setpoints in real-time to implement a customizable CC strategy. To perform real-time setpoint optimization using such a reference governor, a model of the processes, for which setpoints are being optimized, is needed to evaluate candidate solutions. Many mathematical models of FFPUs are too computationally complex to be implemented in a real-time control scheme where the unit load demand is constantly changing. Therefore, a more advanced model must be used to speed up the simulation process for this strategy to be successful.

Artificial neural networks (ANNs) are a good choice as a replacement for computationally complex mathematical models because they can emulate these models of complex systems relatively quickly and are effective at modeling the nonlinear dynamics that are characteristic of FFPUs. Once trained, an ANN model can be used to predict the response of the modeled processes to control inputs to test whether the control result is a sufficient one. Using an ANN model in this way is considered model predictive control (MPC).

A number of search algorithms were explored in [3] to discover the best algorithm for producing optimized control setpoints in a reference governor designed for a 160 MW oil-fired drum-type boiler-turbine-generator FFPU. Reference [3] found that particle swarm optimization (PSO) algorithms give faster convergence with fewer iterations compared to a genetic algorithm (GA), and are sufficiently fast for online setpoint optimization. As will be explained in a later section, this FFPU is the model used in this research, and this reference governor design is used here for optimized multi-objective coordinated control of this model.

The setpoint optimization performed by the reference governor generates setpoints in a way that allows a FFPU to satisfy conflicting operating objectives, such as the need to meet the unit load demand while conserving fuel, reducing pollution, maximize equipment life, et cetera, to achieve greater overall efficiency. Priority values are used to give more weight to more important objectives to ensure they are met. For example, the need to meet the unit load demand would be given a higher priority than other objectives to ensure the setpoint optimization produces a result that generates the power output demanded of the plant.

1.2.4 Adaptive Feedback Gain Tuning

Adaptive feedback gain tuning is another control technique that can be used to increase efficiency in FFPUs. Adaptive gain tuning is used in this thesis to adjust the feedback gain values when the performance resulting from feedback control is determined to need improvement. One reason feedback gains need improvement is that the nonlinear dynamics of the FFPU cause the system to behave differently as the power level changes. As the behavior changes, feedback gains may become less effective for performing feedback control. However, adaptive gain tuning, in general, is used to preserve stability regarding feedback control during operation of a power unit.

Performing adaptive gain tuning online requires the ability to monitor the performance of the FFPU systems to determine when the response to feedback control is no longer adequate. When feedback gains need adjustment, the adaptive gain tuner will optimize new gains using the predicted response of the plant. Similar to the optimization method used in the reference governor, the feedback gain optimization will use an ANN model of the power plant processes to predict their response to candidate feedback gain values. Based on the performance shown in the predicted responses, gain values can be evaluated to find an optimized set using PSO.

1.2.5 Model Identification for MPC

To use MPC in a real-time control system, a model of the controlled system must be obtained. In this thesis, the processes of the FFPU are modeled using ANNs. Since the equations for these processes are available for the model used here, data can be obtained without collecting sensor data from an actual power plant unit. Instead, the equations are used to generate the data needed to train the ANNs to learn the process behavior.

There are two types of ANNs used in this thesis, feedforward-type and dynamic. The feedforward-type ANN can be sufficiently trained, offline, using the data generated from the FFPU model equations. The dynamic ANN requires online training in addition to offline training. The reason for this is that as the plant behavior changes with the change in power level, due to nonlinear system dynamics, the dynamic ANN needs to be updated to be able to model the change.

To update the dynamic ANN model online, the ability to collect data real-time system data is needed. With this data, the model can be incrementally trained to be accurate in simulating the dynamics at the current level of operation. This is what is needed with the dynamic ANN, because it is used to model the FFPU response to feedback control using candidate gains from the adaptive gain optimization process to improve feedback control at the current level of operation. This requires the ability to train ANNs online.

1.2.6 Need for a Multi-Agent System Approach

Each of the control techniques mentioned previous to this section are computationally intense. If they are to be implemented simultaneously for real-time control of the distributed subsystems of a large-scale FFPU, a control system framework that provides the means for decentralized control, parallel computation of control tasks, system monitoring, and large-scale coordination is needed. Furthermore, it is desirable that the framework be robust, flexible, and extensible. For these reasons, a multi-agent system (MAS) approach to implementing such a control scheme is attractive.

Though there are many ideas as to what constitutes an MAS among the computer science community, there are common features among the differing ideas [7]. All MASs consist of intelligent agents that perform separate tasks to perform a larger, coordinated goal. To solve the problems addressed in this thesis, agents would perform system monitoring and control tasks to achieve the coordinated goal of efficient, optimized, distributed control of an FFPU, with the ability to customize the operating goals that define efficiency and optimality. Furthermore, there is the potential to add robustness, flexibility, and extensibility because of the distributed, modular nature of MASs.

1.3 Problem Statement

There has been a considerable amount of research done to explore the use of a MAS framework to coordinate control and monitoring tasks which perform distributed multiobjective optimized intelligent control of FFPUs [6,8,9,10,11]. Numerous control techniques, including the ones mentioned here, have been developed to control FFPUs in this previous research. Also included in this research is the preliminary development of an MAS framework that organizes control and monitoring processes into agents for realizing control of a 600 MW FFPU and the 160 MW FFPU used here. However, a fully operational MAS has not been developed to test the online performance of control tasks executed by agents or the overall performance of a fully functional multi-agent control system (MACS).

1.4 Objective and Scope

It is the goal of this thesis to continue previous work done to develop an MAS method for implementing control and monitoring tasks of a FFPU by developing, implementing, and testing a fully functional MAS designed for the control of a 160 MW FFPU. Though the MAS control method is meant to address complex problems faced by large-scale FFPUs, a smaller, less complex 160 MW FFPU is the target of control in this thesis to simplify the development of the needed infrastructure. Since the MAS method is extensible and non-rigid, the infrastructure developed here can be used to design an MAS to control larger, more complex systems.

This thesis consists of five chapters. The first chapter gives motivation and background for this thesis, as well as a description of the problems addressed. The problems facing the electric utility industry and shortcomings of other control methods are discussed to motivate the development of a more advanced control system. This chapter also discusses the previous work that has led to the extension this thesis provides.

The second chapter describes the overall control architecture that is to be implemented by the MAS, the mathematical model of the FFPU used as the target of control to develop the MAS, and the control techniques to be implemented by the MAS. The third chapter gives a detailed description of MASs, the structure of an individual agent, and the structure of the proposed MACS. The proposed agents that perform the control techniques and the messaging system that allows the agents to communicate are also explained.

Chapter four details the software implementation and simulation of the MAS, the methods used to test the control system, as well as the methods used to confirm the successful operation of the individual agents. The fifth chapter gives a summary of this thesis, draws conclusions from chapter four, and suggests possible future research opportunities that could add to this research.

CHAPTER TWO

The Optimized Multiobjective Control System

This chapter discusses the individual control techniques of the optimized multiobjective control system and how they work together to perform control of a power plant system. To do so, an overview of the architecture of the optimized multiobjective control system is given. Then, the power plant model used to evaluate the operation of the control system is discussed and the reference governor that is used to calculate optimized control valve setpoints and output level references for control of the power plant model is described. Next, the feedback control system used to control the power plant model at the setpoints generated by the reference governor, and the gain optimizer used to optimize feedback gains for use in the feedback controller are discussed. Lastly, the function, type choice, structure and training of the artificial neural networks (ANNs) used to model the power plant system for model predictive control (MPC) in the reference governor and gain optimizer are described.

2.1 Optimized Multiobjective Control System Architecture

The goal of the optimized multiobjective control system, pictured in Fig. 2.1, is to allow optimized control of a power plant system in a customizable manner, such that an operator can choose operating goals for a specific plant system and implement them in real-time. Each goal is given a priority value based on ascribed importance to be used by the control system to produce optimized control. This method was proposed in [3] to help power plants respond to the constantly changing requirements on operation due to

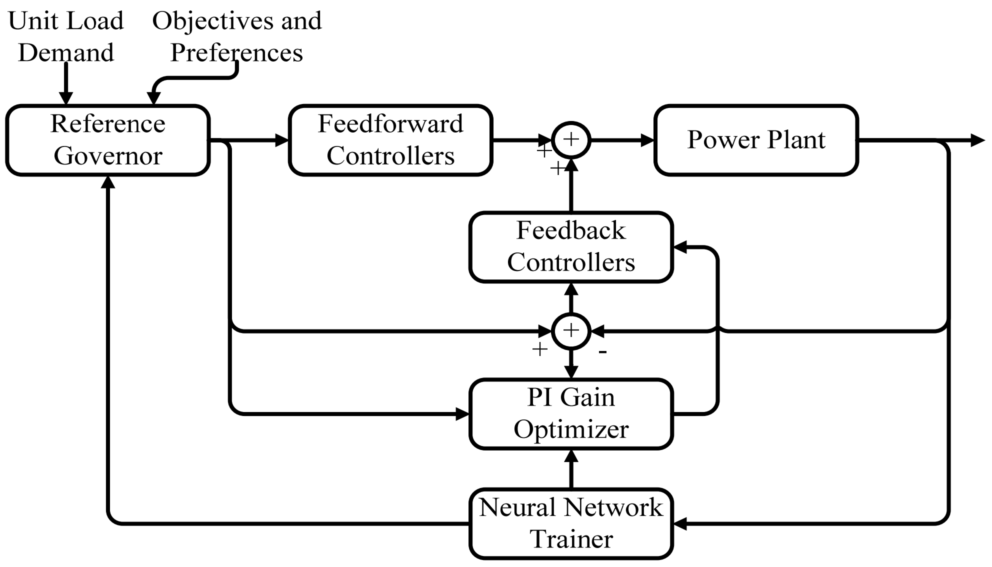


Fig 2.1. Block diagram of the overall control system.

government regulations as well as changes made to increase efficiency in order to become more competitive in today's utility market.

The inputs to the control system are the unit load demand, operating objectives, and preferences. The unit load demand is the power output demanded of the power plant unit, and is determined by economic dispatch, which dictates the operation of multiple power units to produce enough energy to reliably meet the demand from end users at the lowest cost to the provider. The operating objectives are strategic goals, with respect to operation of the power unit, that are meant to increase the efficiency of the unit, such as minimizing pollution, fuel consumption, stress on equipment, etc. Preferences are values that rank the operating objectives in order of importance, and, as will be discussed later in this chapter, these values determine how the operating objectives affect the outcome of the control setpoint optimization.

Based on these inputs, the control system uses a reference governor, feedforward control, feedback control, gain optimization, and real-time neural network training to produce optimized control of the target power plant unit. The reference governor uses the control system inputs, electric power demand level and the operating objectives and preferences, to optimize operating setpoints and output level references to be used in the feedback controller. The functionality of the feedforward controller is performed in the reference governor as a result of the optimization process.

The feedback controller uses the references from the reference governor and feedback gains produced by the gain optimizer to maintain tight control of the plant system with respect to the references. The gain optimizer optimizes the feedback gains whenever the control system detects that the error between the setpoints and plant output are above the predetermined threshold. The real-time neural network trainer continually collects data to train and adapt the artificial neural networks that are used in the control system to model plant processes in the optimization procedures performed in the reference governor and the gain optimizer.

2.2 Power Plant Model

The power plant model used to develop and test the optimized multiobjective control system discussed in this thesis is a mathematical model of a 160 MW oil-fired drum-type boiler-turbine-generator unit, a detailed formulation of which can be found in [12]. It is modeled as a third-order three-input three-output nonlinear model. The inputs to the system are positions of valve actuators that control the mass flow of fuel (represented as u_1 in per unit), steam to the turbine (u_2 in per unit), and feedwater to the drum (u_3 in per unit). The outputs are electric power generated by the plant (E in MW), drum steam pressure (P in kg/cm^2), and drum water-level deviation (L in meters). The resulting state variables are electric power (E), drum steam pressure (P), and steam-water density (ρ_f). The dynamic equations for the third-order model were developed by Bell and Åström in [12] and are as follows:

$$\frac{dE}{dt} = \left(\left(0.73u_2 - 0.16\right)P^{9/8} - E\right)/10 \tag{2.1a}$$

$$\frac{dP}{dt} = 0.9u_1 - 0.0018u_2P^{9/8} - 0.15u_3 \tag{2.1b}$$

$$\frac{d\rho_f}{dt} = \left(141u_3 - \left(1.1u_2 - 0.19\right)P\right)/85 \tag{2.1c}$$

The drum water level deviation from a fixed, drum-specific setpoint is calculated from the solution for ρ_f in equation (2.1c) in conjunction with the algebraic equations below:

$$q_e = 45.59166u_1 + \left(0.8537u_2 - 0.14746\right)P - 2.51431u_3 - 2.0958 \tag{2.2a}$$

$$\alpha_s = \frac{\left(1/\rho_f - 0.00154\right)}{\left(1/\left(0.8P - 25.6\right) - 0.00154\right)} \tag{2.2b}$$

$$L = 6.3565u_1 + 3000\alpha_s + 5.5556q_e - 3275 \tag{2.2c}$$

where q_e is the evaporation rate (kg/s) of water in the boiler and α_s is the steam quality.

Values for the control valve positions are represented by values on [0,1], 0 representing a completely closed valve and 1 representing a completely open valve, and have rates of change limited as shown below, as determined in [11]:

$$-0.007 \le du_1/dt \le 0.007 \tag{2.3a}$$

$$-2.0 \le du_2/dt \le 0.02 \tag{2.3b}$$

$$-0.05 \le du_3/dt \le 0.05 \tag{2.3c}$$

Steady-state equations for this power plant model are obtained by setting the dynamic equations in (2.1) to zero and solving for u_1, u_2, and u_3. This result gives an inverse steady-state model of the dynamic equations, shown below, consisting of only algebraic equations:

$$u_1 = \frac{0.0018u_2P^{9/8} + 0.15u_3}{0.9} \tag{2.4a}$$

$$u_2 = \frac{0.16P^{9/8} + E}{0.73P^{9/8}} \tag{2.4b}$$

$$u_3 = \frac{\left(1.1u_2 - 0.19\right)P}{141} \tag{2.4c}$$

Similarly, the steady-state electric power and drum steam pressure can be calculated from the control valve positions by solving (2.4) for E and P, whose result is shown below:

$$E = \frac{0.73u_2 - 0.16}{0.0018u_2}\left(0.9u_1 - 0.15u_3\right) \tag{2.5a}$$

$$P = \frac{141u_3}{1.1u_2 - 0.19} \tag{2.5b}$$

The above equations are used to model the power plant in software by calculating initial conditions for E, P, L and ρ_f, and continually solving the equations in (2.1) and (2.2) for a specified time-step. The equations in (2.1) are solved using an ordinary differential equation solver, followed by the straightforward calculation of the equations in (2.2). Control of the power plant is simulated by calculating the input values, u_1, u_2, and u_3 using the described control system implemented in software.

2.3 Reference Governor

The reference governor was designed to allow optimized control of a power plant unit with custom operating goals that can be adjusted real-time. The ability to customize operating goals would help operators conform to changing regulations and market situations without redesigning the control system. Therefore, with this reference governor design, the objective functions and preferences can be changed during the operation of the control system.

In operation, the reference governor generates optimized setpoints based on the unit load demand, E_{uld}, operating objective functions, J, and preference values, β, which are the inputs to the reference governor. It does this in three subprocesses, shown in Fig. 2.2. The first subprocess uses preformed tables of feasible operating bounds, listed in Appendix A, for each of the control valves to determine bounds on control operation for the current unit load demand. The second subprocess uses an optimization algorithm to find optimized control setpoint values for the current unit load demand, with respect to the operating objectives and preferences, within the control bounds specified by the first subprocess. The third subprocess translates the optimized control setpoints into output reference levels to be used in the feedback controller.

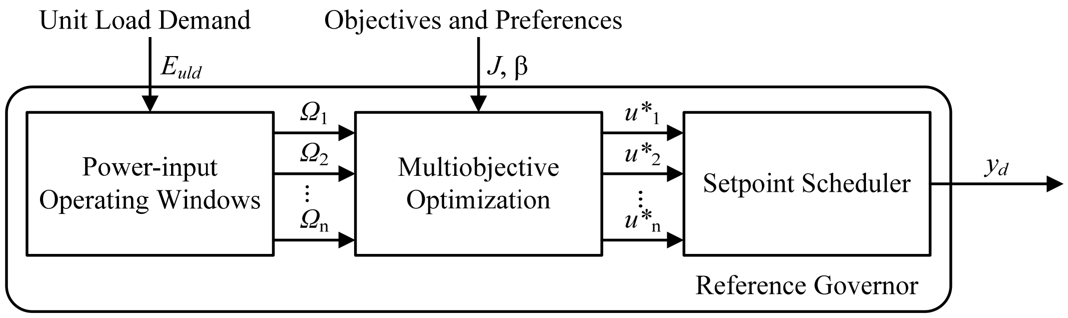


Fig 2.2. Diagram of the process flow in the reference governor.

2.3.1 Power-Input Operating Windows

The power-input operating windows in the first subprocess define the feasible range of operation for each control input with respect to a specific unit load demand. The power-input operating windows for the power plant used in this thesis are calculated from the power-pressure operating window, shown in Fig. 2.3. The power-pressure operating window gives a range of feasible operating drum steam pressures that can achieve a given unit load demand power in a power plant. Using the equations in (2.4), the minimum and maximum drum steam pressures for a specific unit load demand power are used to calculate a minimum and maximum value for each control input. Doing this for all possible unit load demand values results in the graphs in Fig. 2.4. A table of the pressure and control boundary values can be found in Appendix A.

The power-pressure operating window was determined by first calculating control values using the equations in (2.4) for power and drum steam pressure levels representing the full range of operation. Then, the power plant response to those inputs was simulated using the power plant model. The control values that are accepted as feasible are those that result in a steady-state convergence while meeting all constraints, such as the requirement for the steady-state drum water level deviation to be zero meters. The

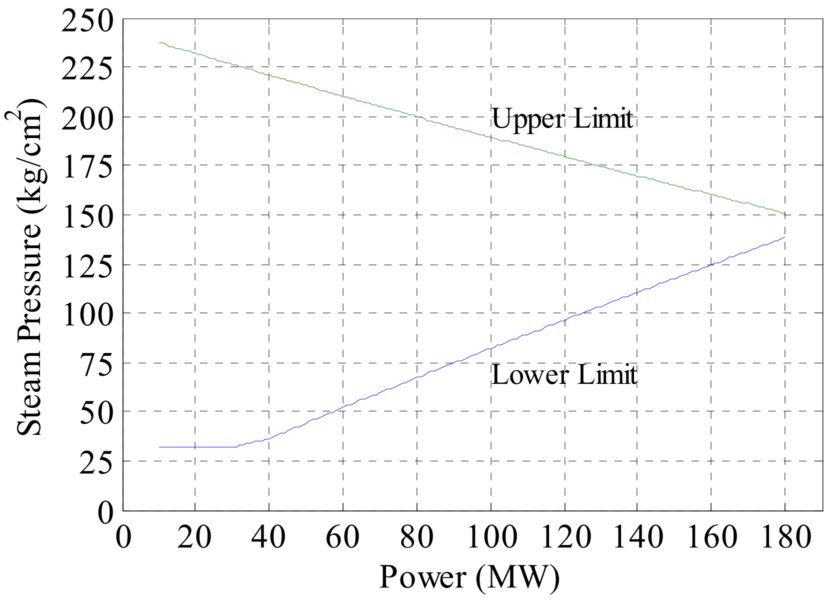
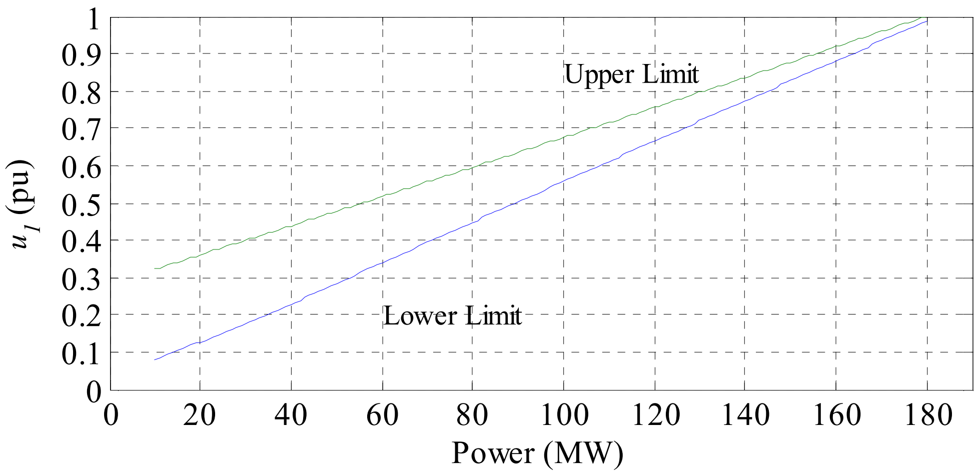
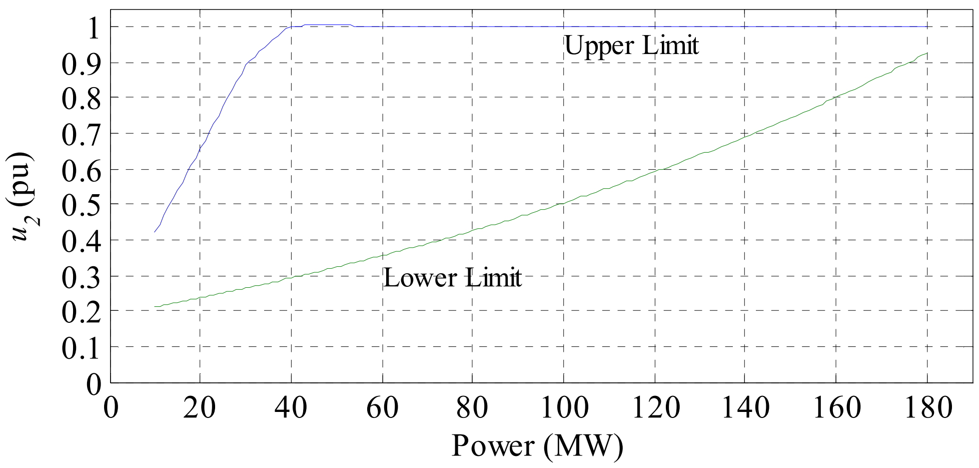


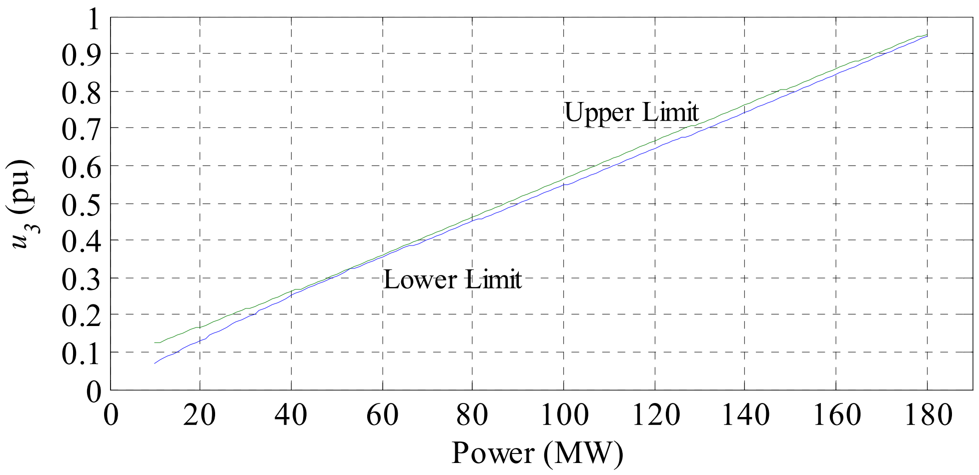
Fig 2.3. Graph of the power-pressure window over the range of possible unit load demands.



(a)



(b)



(c)

Fig 2.4. Plots of the power-input windows over the range of operation.

corresponding drum steam pressure resulting from the simulation to steady-state is also noted. The power-pressure window is made up of all the drum steam pressures resulting from this process matched with their corresponding power levels. The power-pressure window used here was determined in [11].

The optimized control values will, in some cases, be greater than the physical limits of the control valves in the power plant model. As discussed in [11], this is attributed to the fact that the positions of the valve actuators were collected manually while studying the power plant, introducing some involuntary error. This does not affect the formulation of the power plant model, as it is equivalent to replacing the control valves with a resizing procedure. Therefore, for the calculation of the power-input windows, the control values are scaled by the constants shown below to be between zero and one. The values are not scaled in the simulation of the power plant model:

$$kssu_1 = 0.7966 \tag{2.6a}$$

$$kssu_2 = 1.1814 \tag{2.6b}$$

$$kssu_3 = 1.1420 \tag{2.6c}$$

The scaling factor values were chosen such that the power plant model could be used to simulate control action for up to 110% of the rated maximum power of the plant, as is required in practice for power plant operation. This designates the operational range of the power plant model to be between 10 and 180 MW, with 180 MW being a peak maximum load. The maximum sustained load rating is 160 MW.

2.3.2 Multiobjective Optimization

The purpose of the multiobjective setpoint optimization is to solve the problem of finding an optimized combination of control inputs that meet operating objectives with respect to their assigned preferences. The defined operating goals could possibly conflict with each other, which is why preference values are needed. Using the preferences, the optimization algorithm will make sure that higher priority objectives are considered as such.

The operating objectives chosen for the power plant model used in this thesis are to minimize the following objective functions:

$$J_1(u) = |E_{uld} - E| \tag{2.7a}$$

$$J_2(u) = u_1 \tag{2.7b}$$

$$J_3(u) = -u_2 \tag{2.7c}$$

$$J_4(u) = -u_3 \tag{2.7d}$$

where objective function $J_1(u)$ represents the minimization of the power generation error, $J_2(u)$ represents the minimization of fuel consumption through u_1, $J_3(u)$ represents the minimization of the pressure drop across the steam valve u_2, and $J_4(u)$ represents the minimization of energy loss due to the pressure drop in the feedwater valve u_3. Each objective is given a preference value, ranging from 0 to 1. The preference values allow an operator to give priority to the different operating objectives, giving more important objectives higher values. If an objective function has a preference value of 1, it is considered to be most important in the optimization, whereas if it is given a value of 0, the objective is removed from consideration in the optimization process.

The algorithm used in this thesis to find optimized steady-state control values for the multiobjective optimization is hybrid particle swarm optimization (HPSO), as this method was found to outperform other methods used to obtain optimized solutions to this problem in [3]. The other methods tested in [3] include other variations of particle swarm and genetic algorithms, such as the constriction factor approach and evolutionary particle swarm optimization. The difference between particle swarm, which was proposed in [13], and the hybrid method is that the particles, or candidate solutions, with the worst performance are moved to the positions of those with the best performance, represented by step 6, below. An inertial weight, *w*, is also used. The HPSO is performed as follows:

1. Randomly initialize *n* particles within the search space, where *n* is a design parameter in the search algorithm.
2. Evaluate each particle with respect to the objective functions and preferences used to evaluate the fitness of candidate solutions.
3. Check to see if each particle's current fitness is better than its personal best, *pbest*. If so, store the new fitness as the personal best.
4. If there is a personal best that is better than the current global best, *gbest*, then replace the previous global best with that personal best.
5. Calculate a velocity for each particle to determine its next position in the search space and move them accordingly using the equations below, where v_i^k is the current velocity for particle *i* at iteration *k*, c_1 and c_2 are chosen weights, *rand*$_1$ and *rand*$_2$ are uniform random numbers on [0, 1], and s_i^k is the current position of particle *i* at iteration *k*:

$$v_i^{k+1} = wv_i^k + c_1 rand_1 * \left(pbest_i - s_i^k\right) + c_2 rand_2 * \left(gbest - s_i^k\right) \tag{2.8a}$$

$$w = w_{\max} - \left(\frac{w_{\max} - w_{\min}}{iter_{\max}} \right) * iter \tag{2.8b}$$

$$s_i^{k+1} = s_i^k + v_i^{k+1} \tag{2.8c}$$

6. Move the half of the particles with the worst current performance to the position of the half with the best performance, keeping the calculated velocities.
7. Re-evaluate each particle's performance as in step 2 and repeat steps 3-6 until either a sufficient solution is found, or the iteration limit is reached.

For the power plant model used in this thesis, the equations shown below are used to evaluate the fitness of the particles:

$$\delta_m = \max_{i=1,2,\ldots,k} \delta_i,\ \delta_i \geq 0 \tag{2.9a}$$

$$\delta_i = \beta_i \left| J_i(u) - J_i(u)^* \right|,\ i = 1,2 \ldots k,\ u \in \Omega \tag{2.9b}$$

$$J_i^* = \min\left\{ J_i(u); u \in \Omega \right\},\ i = 1,2 \ldots k \tag{2.9c}$$

This approach uses the largest value of the four objective functions as the overall fitness for a specific particle. The particle with the smallest calculated δ_m is considered to have the best performance with respect to the operating objectives and preferences. The objective function $J_1(u)$ is calculated using a steady-state model of the plant that takes control values as input and outputs the corresponding steady-state electric power and drum steam pressure, where the steady-state model is a feedforward-type ANN trained to model the equations in (2.5). The training and real-time adaptation of this model is discussed in Section 2.6.

The optimization method used in the reference governor has been designed to be general so that it can be used in other power plant units. All that is necessary to make the change is to design objective functions specific to the needs of the power plant to be controlled, and obtain a process model to evaluate the fitness using those objective functions as described here. It is expected that this method be used on power plant units that are much more complex in terms of number of inputs and operating goals. The simple model used here serves to simplify the development process of this control method as a starting point to controlling much larger, more complex systems.

2.3.3 Setpoint Scheduler and Feedforward Controller

The purpose of the setpoint scheduler is to calculate the generation setpoints from the feedforward control values produced by the optimization process. In the method used in this thesis, this action is carried out in the optimization process. As described in Section 2.3.2, electric power and drum steam pressure are calculated for each set of candidate control values. Once an optimized set of control values is found, the corresponding steady state electric power and drum steam pressure have already been determined and can be used as generation setpoints, using zero as the water level deviation demand.

The functionality of the feedforward controller is also performed in the optimization process. The function of the feedforward controller is to produce feedforward control values from the generation setpoints. Since the control optimization process produces a set of optimized control values and corresponding generation setpoints, there is no need for a separate feedforward controller apart from the reference governor.

2.4 Feedback Controller

The feedback control system uses proportional-integral (PI) control to track the setpoints generated by the reference governor, a general equation for which is shown below:

$$u_{fb}(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau \qquad (2.10)$$

Since the input and output variables are coupled, the resulting feedback gains, K_p and K_i, are three-by-three matrices, in this case, to allow for the interdependence of the variables. To simplify the control system, and therefore the calculation of feedback gains, the relative gain array (RGA) matrix was calculated to analytically determine which inputs have the most effect on the outputs. This calculation, shown in Appendix B, shows that the fuel flow, u_1, has the most effect on output power, the flow of steam to the turbine, u_2, has the most effect on drum steam pressure, and the flow of feedwater into the drum, u_3, has the most effect on the water level.

By this finding, the feedback controller was simplified into three feedback loops, with a PI controller in each loop. In this arrangement, each control loop controls one input variable based on one of the three output errors. Specifically, the feedback control loops for the mass flow of fuel, u_1, the mass flow of steam to the turbine, u_2, and the mass flow of feedwater to the drum, u_3, are controlled by the error between the electric power setpoint and the actual electric power output, the error between the drum steam pressure setpoint and the actual drum steam pressure, and the error between the water level deviation setpoint and the actual water level deviation, respectively. Diagrams for the control loops are shown in Fig. 2.5, where E_d is the desired unit load demand power, P_d is the desired drum steam pressure, L_d is the desired drum water level, K_{Pi} are the

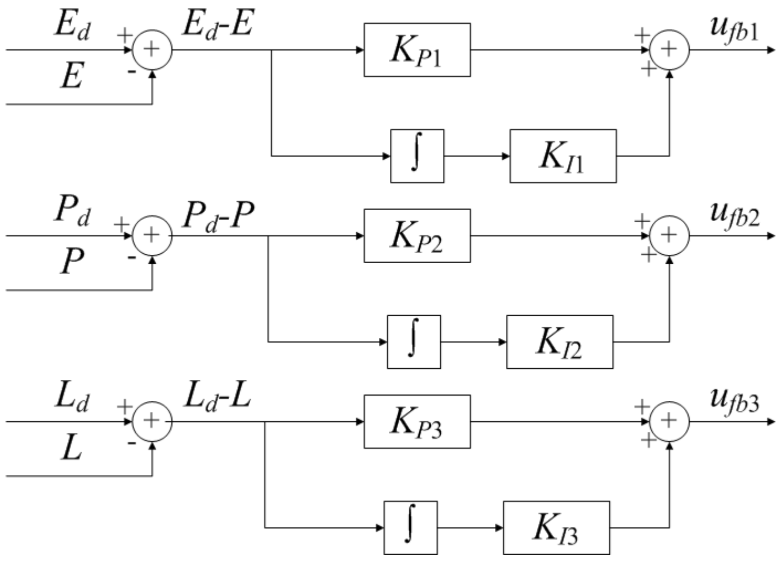


Fig 2.5. Block diagrams of the PI control loops for the decoupled feedback control approach.

proportional gains, K_{Ii} are the integral gains, and u_{fbi} are the respective feedback compensation values to be added to the steady-state feedforward control values, with i = 1, 2, 3. The gain values for the PI controllers are determined by the gain optimizer discussed in the next section.

2.5 Gain Optimizer

In this control system, gain optimization is used to optimize the PI feedback control gains used in the feedback controller. The gains are optimized once offline to obtain an initial set, and then in real-time whenever the performance of the feedback controller is found to need improvement. This way, it is possible to maintain good-as-possible control for all operating levels with respect to feedback control, increasing the overall efficiency of the power plant unit.

The need for gain improvement is determined using error thresholds. These thresholds are error levels set as a design parameter for a specific power plant unit and are compared against the output errors calculated by the feedback controller, E_d-E, P_d-P, and L_d-L. Once an error exceeds one of these thresholds, the gain optimizer will perform the optimization to improve control.

The algorithm chosen for the optimization is hybrid particle swarm optimization (HPSO), described in Section 2.3.2. It was chosen because of its success in optimizing the steady-state controls in the reference governor over other methods explored in [3]. However, though the author recognizes that this does not necessarily imply that it works better than all other algorithms for this optimization problem, finding the best optimization method was not a focus of this thesis.

The candidate gain sets are tested against a step in unit load demand, generated by the reference governor, over a sufficient amount of simulation time to analyze performance costs. To do this, a dynamic ANN, called a recurrent neural network, is used to simulate the power plant unit's response to feedback control using the candidate feedback gains to track the step input. The details of the training and real-time adaptation of the recurrent neural network are discussed in Section 2.6.

The objective functions used to optimize the feedback gains are the sum-squared errors for electric power, drum steam pressure, and water level deviation, shown below:

$$J_1 = \sum_{n=1}^{N} \left(E_d[n] - E[n] \right)^2 \tag{2.11a}$$

$$J_2 = \sum_{n=1}^{N} \left(P_d[n] - P[n] \right)^2 \tag{2.11b}$$

$$J_3 = \sum_{n=1}^{N} \left(L_d[n] - L[n] \right)^2 \tag{2.11c}$$

The costs for this optimization are evaluated similarly to those in Section 2.3.2, with the equations shown below:

$$\delta_m = \max_{i=1,2,\ldots,k} \delta_i,\ \delta_i \geq 0 \tag{2.12a}$$

$$\delta_i = \beta_i \left| J_i - J_i^* \right|,\ i = 1, 2 \ldots k \tag{2.12b}$$

$$J_i^* = \min\left\{ J_i \right\},\ i = 1, 2 \ldots k \tag{2.12c}$$

These equations find the objective function with the highest normalized error for each particle, or candidate solution, and use that value, δ_m, to evaluate fitness compared to the rest of the particles. During the optimization, if in any iteration a particle has a better performance than its current personal best, it will replace its personal best with the current gain set. If there is a particle that outperforms the current global best in a given iteration, it will replace it with its current gain values.

Another design consideration that must be made here is the maximum length of time to allow the gain optimization to run. Since gain optimization is only used when there is a perceived deficiency in the effectiveness of the current feedback gains, the faster an optimized solution can be found the better. As a general rule, the optimization must be run as long as it takes to consistently get an acceptable result over the entire range of the problem for which the optimization is to provide a solution. Factors affecting the time it takes to optimize are the number of particles used in the optimization, the length of simulation time used in the evaluation of candidate gain sets, and the maximum number of iterations. These parameters are design considerations that must be made based on the specific power plant unit to be controlled.

One way to shorten the time it takes to produce an optimized result, and also to make sure that the resulting gain set is not worse than the previous one, is to use the previous gain set as a starting point for the optimization. To do this, the previous gain set is used as one of the initial particle values, with the rest initialized randomly. This way, any improved gain sets that are found will be at least as good as the previous one. If for some reason the optimization does not find a better result, the current gain set will continue to be used, and the optimization can be run again until a better gain set is found.

2.6 Artificial Neural Network Models and the Neural Network Trainer

Artificial neural networks (ANNs) are used in this control system because the more complicated power plant systems for which this control method is intended either don't have equation models available or the available model is too computationally complex to be implemented due to constraints on calculation time. This necessitates a less computationally intense model suited for modeling nonlinear systems, such as an ANN. Therefore, though there is a simple set of equations for the power plant modeled in this thesis, ANNs are still used in the control system for the purpose of developing a method for implementing neural networks and a neural network trainer for use in larger, more complicated systems. Henceforth, the reader is assumed to have a minimal understanding of ANNs and is referred to Appendix C for a brief introduction.

There are two types of ANNs used to model the physical processes that take place in the power plant unit used in this thesis, a feedforward neural network (FFNN) and a dynamic ANN called a layer-recurrent neural network (LRNN). These ANNs are described in detail in the next two subsections.

2.6.1 Feedforward Neural Network

The FFNN is used to predict the steady state power plant output produced by a particular combination of control inputs. This prediction is meant to be equivalent to introducing a set of control inputs to the power plant unit and keeping them constant until steady state operation is achieved, noting the output at steady state. The FFNN is used in the reference governor to evaluate the electric power output and drum steam pressure corresponding to candidate control value sets as a cost in the control optimization process.

An FFNN is used in the control optimization process, because it is much faster, from a simulation standpoint, compared to running a dynamic simulation to steady state for each candidate control set to be evaluated. With multiple candidate sets of control values that are evaluated for each iteration, where there are many iterations, there is the potential for this evaluation to take place several hundreds or thousands of times for each optimization. Because of this, it is crucial that the evaluation be as quick as possible while giving an accurate result.

The FFNN is made up of multiple artificial neurons in three or more layers consisting of an input layer, a hidden layer or layers, and an output layer. The neurons in each layer are connected to all of the neurons in the preceding and proceeding layers with no recurring connections. The exceptions are the input layer, which has no preceding layer, and the output layer, which has no proceeding layer. A generalized diagram of this configuration is shown below in Fig. 2.7.

Each of these connections between artificial neurons contains a weight value that models the strength of a synaptic connection between neurons in the brain. These values can initially be set randomly and can then be trained using an appropriate training algorithm.

There are two types of training: parameter learning and structure learning. Parameter learning trains the weights between artificial neurons using a predetermined

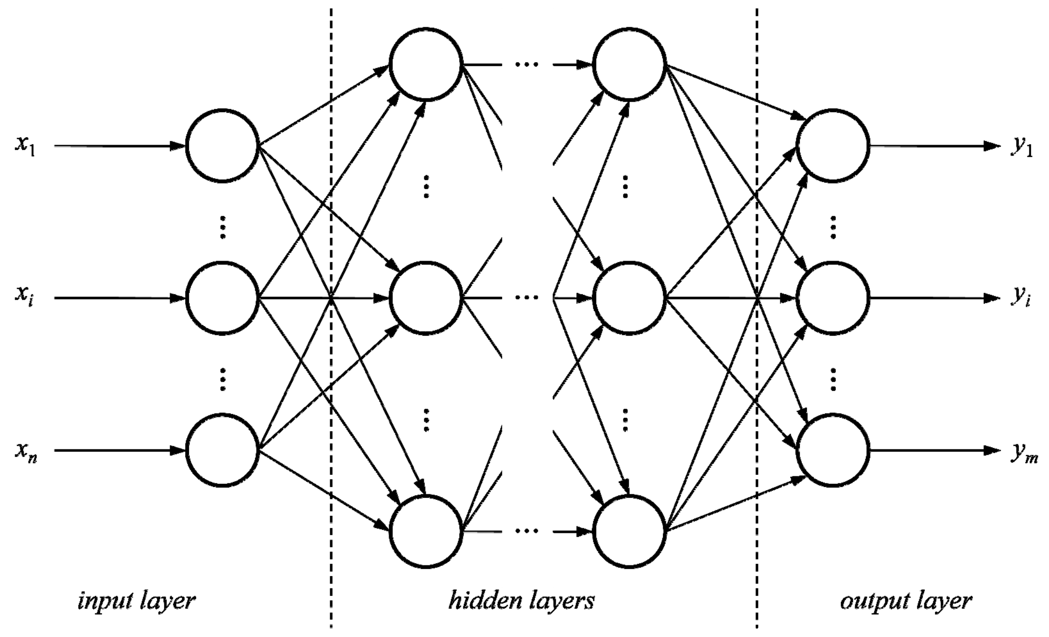


Fig. 2.6. General configuration of artificial neurons and weight interconnections for a multilayer feedforward neural network.

connection structure and number of neurons, where structure learning rearranges the structure of the ANN in order to learn a desired behavior. For the ANNs in this thesis, only parameter learning is used.

In parameter learning, there are three methods of training: supervised learning, reinforcement learning, and unsupervised learning. Supervised learning is the only method of training used in this thesis. With supervised learning, a set of normalized data is made available to the training algorithm consisting of inputs and outputs from the system to be modeled. The training algorithm uses the given inputs with the FFNN to simulate the corresponding output and calculate an error between the simulated output and the outputs provided in the training data. The error between these outputs is used to adjust the weights and biases between neurons using the method defined by the chosen training algorithm.

During the training process, the changing of weights and biases based on output error continues until the error has been decreased to a level deemed sufficient to accommodate the required accuracy of the application. Once the training process is finished, another set of data collected from the target system that wasn't used in training can be used to test the newly trained FFNN for accuracy. This gives a measure of the ANNs ability to accurately simulate the modeled system, where if only data used in the training process is used to test the ANNs accuracy, the tests may give a false sense of accuracy. This is because the training could resulted in the memorization of the training data rather than learning the overall behavior of the system.

2.6.2 Dynamic Neural Network

In this thesis, an LRNN is used to simulate the power plant dynamics resulting from inputs given by the addition of feedforward and feedback control values. Using the network in this way makes it possible to evaluate the cost, or fitness, of candidate proportional and integral gain combinations to be used in the feedback controller, which are determined in the gain optimization process performed in the Gain Optimizer. Using the LRNN allows the gains to be optimized while the control system is online and without using the physical system to test candidate gains, which makes optimized real-time, online gain-tuning possible.

The specific ANN used for this purpose is a LRNN. The LRNN is a generalized form of the Elman network. The LRNN generalizes the Elman network by allowing the use of multiple hidden layers and an arbitrary number of transfer functions in the artificial neurons, whereas the Elman network only uses one hidden layer and set transfer functions. However, the LRNN used in this thesis is constructed to have the original configuration used by the Elman network with the possibility of making needed configuration changes.

The structure of an LRNN is similar to that of an FFNN. The difference is that the LRNN contains internal time-delayed recurrent connections that store the time-delayed output of each artificial neuron in a hidden layer to be input into each neuron in that layer, as shown in Fig. 2.8. This configuration applies for all hidden layers in an LRNN. These time-delayed recurrent connections serve as a temporal memory through which the network can simulate system dynamics.

The training process for the LRNN is similar to that of the FFNN, only sequences

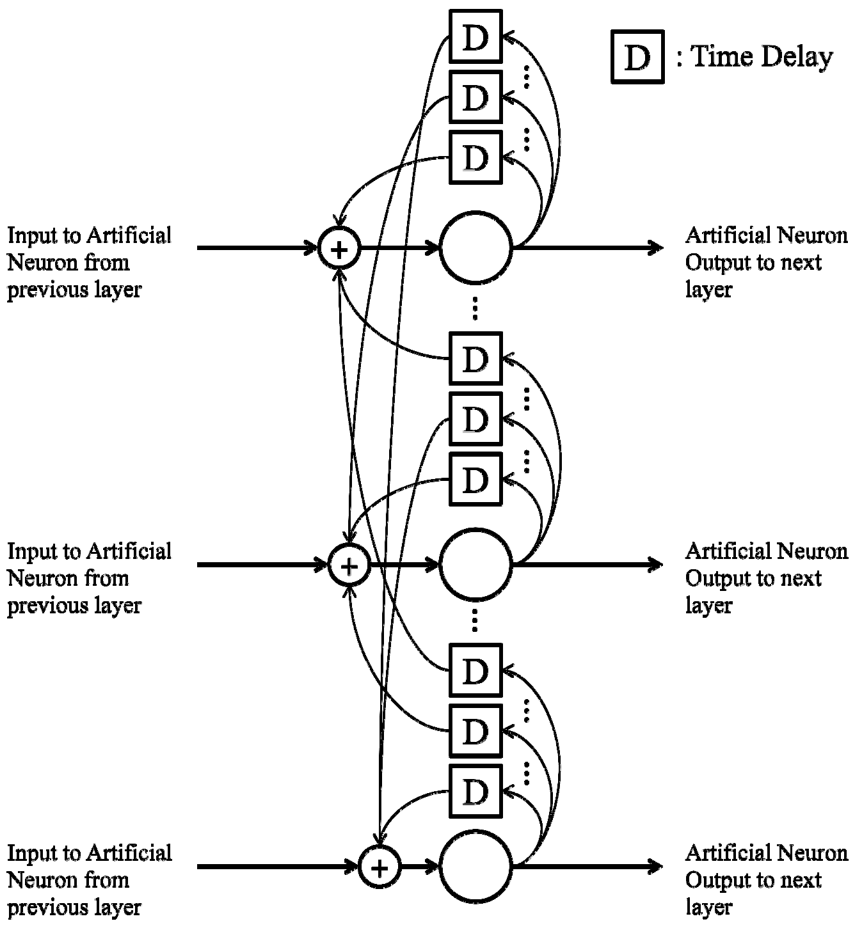

D : Time Delay
Input to Artificial Neuron from previous layer
Artificial Neuron Output to next layer
Input to Artificial Neuron from previous layer
Artificial Neuron Output to next layer
Input to Artificial Neuron from previous layer
Artificial Neuron Output to next layer

Fig. 2.7. General configuration of artificial neurons and weight interconnections for a single hidden layer in a layer recurrent neural network.

of data are used to train this ANN instead of multiple independent input-output pairs. Sequences of data are needed in this case, because in order to accurately train the network weights, especially the recurrent connection weights, the training algorithm must be have access to time-delayed output values for each input-output pair to be trained. Just as with the FFNN, the LRNN is trained in this thesis using supervised parameter learning. Supervised parameter learning uses the error between simulated outputs and provided outputs to change the network weights according to the rules of the training algorithm.

The recurrent neural network structure can be further simplified by the Diagonal Recurrent Neural Network (DRNN). The DRNN is a simplified LRNN where the time-delayed output of each artificial neuron in the hidden layer is input only into the same neuron and not to any other neurons in the hidden layer [15]. This greatly simplifies the network structure and reduces computation time since only one recurrent weight will be needed for each neuron in the hidden layer, avoiding cross talk between neurons.

2.6.3 Neural Network Trainer

The ANNs mentioned in the last two section are initially trained offline to be able to model power plant operation at operating points throughout the power-pressure window shown in Fig. 2.3 equally well. This training is referred to as global training. The global training for the FFNN results in a model that is very accurate at modeling the power plant in all regions of the power-pressure operating window.

The global training for the LRNN, however, produces a model that is equally accurate at representing all operating points, but is not accurate enough in any one region to be useful. This means that after global training, the LRNN is not accurate enough to be used to evaluate the performance of gain values in the gain optimization procedure.

Therefore, the LRNN must be fine-tuned with data specific to the current level of operation to give a useful result.

To correct this problem, a neural network trainer has been designed to fine-tune the LRNN online, as real-time power plant data is collected, to be more accurate in modeling the current level of operation. The neural network trainer incrementally trains the LRNN with the new data as it is collected. This allows the gain optimizer to have access to a LRNN that is most accurate at simulating the current operating level, which is what is needed.

Though fine-tuning the LRNN improves simulation accuracy for the operating level, it decreases the simulation accuracy in the other operating levels. Therefore, special consideration must be made so as not to degrade the simulation accuracy of the LRNN in other regions to the point where they cannot be retrained online to give an accurate result.

CHAPTER THREE

Multi-Agent System

This chapter gives definitions for multi-agent systems (MAS) and discusses the details of the multi-agent control system (MACS), which is a multi-agent implementation of the control method described in Chapter Two. The sections herein describe the common architecture shared by agents, the architecture of the MACS, the agents proposed to implement the control system and their functionality, and the agent communication protocol. Considerations for handling agent failures are also discussed.

3.1 Overview of Multi-Agent Systems

Within the computer science community, there are many different ideas regarding what constitutes a multi-agent system [16-20]. However, all of these descriptions seem to agree on a few central ideas which include the concept of an agent, an agent's environment, and an agent's autonomous nature [7]. In [20], an agent is described simply to be, "a software (or hardware) entity that is situated in some *environment* and is able to automatically react to changes in that *environment*."

Furthermore, an intelligent agent is defined to be an agent that is autonomous, proactive, reactive, social, and flexible within its defined environment [7]. An agent is defined as autonomous if it is able to operate on its own to perform its designed functionality once it is activated. An agent is proactive if it has been designed with a goal, or goals, and is actively pursuing those goals autonomously. Reactivity in an agent indicates that it has the ability to respond to perceived changes in its environment in a way that serves to meet its programmed goal. Furthermore, an agent is considered to be flexible if it has the ability to react appropriately to unexpected situations in its environment. An agent is described as social if it has the ability to communicate with other agents, and possibly resources, available to it. A multi-agent system (MAS) is defined to be two or more agents working together to achieve a coordinated goal.

Within the context of this thesis, agents are defined to be software running on computers connected through intranet, for the purpose of communicating via UDP, that align with the above mentioned characteristics. More than one agent can run on a single computer. The environments in which they are situated are specific systems, physical and cyber, within a power plant unit. Perception of an environment is carried out through the use of sensors and communication received from other agents, system resources, or an operator. For the purposes of this thesis, the coordinated goal of the MAS is to implement and maintain intelligent control of a power plant unit to allow optimized operation as defined by the optimized multiobjective control scheme discussed in Chapter Two. The specifics of how this MAS achieves this goal are discussed in the sections to follow.

3.2 Single Agent Architecture

In this thesis, each agent has a common architecture independent of its task, depicted in Fig. 3.1. Within this architecture, each agent consists of a *task thread* and a *messenger thread* [21], where a thread is a computational task being performed in parallel with other threads to perform one process [x5]. The task thread performs an agent's task functionality by executing the necessary algorithms. The messenger thread receives and processes all incoming communications from other agents and resources in the MAS. The agents were structured in this way so that the task thread is not forced to regularly halt, or

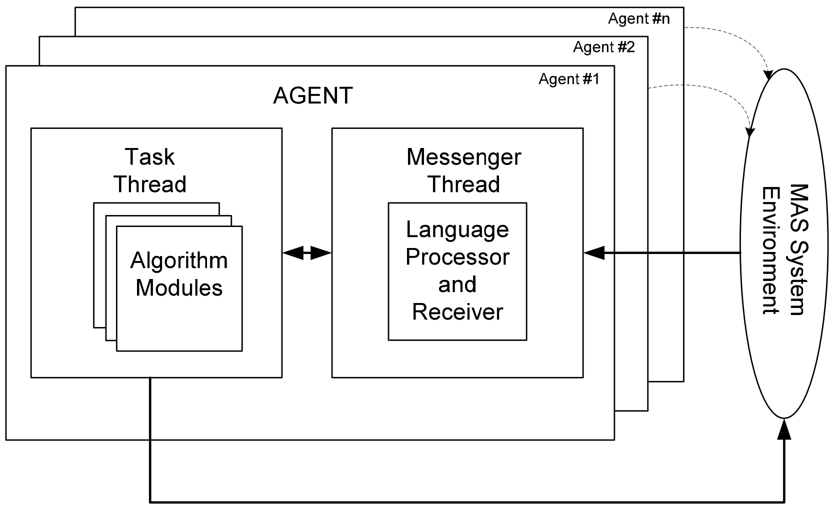


Fig. 3.1. Diagram of the common agent architecture.

block, its operation to listen for and manage incoming communication. However, the task thread sends its own messages, because it is more efficient than having to synchronize the message data with the message thread.

For the MAS proposed in this thesis, the task thread for each agent contains all of the code necessary to perform the duties of any agent. This allows the MAS to reassign tasks to maintain control of the power plant unit in the case of an agent failure. The ability of the MAS to restructure itself gives flexibility and robustness to the control system, as it is able to automatically adapt to potentially devastating situations, such as when agents that are critical to the operation of the control system fail.

The messenger thread contains an interpreter to decipher incoming MAS messages, the format of which will be described in a later section. After interpreting an incoming message, the messenger thread is used to preprocess the data contained in messages as much as possible to save computation time in the task thread. Once the data has been processed and is ready to be relayed to the task thread, it is sent during a synchronization period that is invoked in the task thread only when a message has been received and processed. The synchronization period is designed to make sure that data is not updated in the task thread until the appropriate time. Otherwise, data being used in calculations in the task thread may be overwritten and the results made inaccurate due to the mixing of current and past data in the same calculation.

3.3 Multi-Agent System Architecture and Proposed Agents

The agents in the MAS have been divided into a three-tier hierarchy based on their functionality type. This hierarchy comprises the MAS architecture. The three tiers are designated as high, middle, and low level. High level agents, or interface agents, allow human operators to interact with the multi-agent system to specify control parameters, and monitor operation and performance in the system. Middle level agents, or managing agents, delegate tasks, monitor agent and system performance, and provide means to acquire, store, and distribute data throughout the system. Low level agents, or control agents, execute the functionality of the control system. Low level agents include those that interface with the physical systems of the power plant.

The agents proposed to control the power plant unit described in Section 2.2 are the Interface agent, Free agent, Delegation agent, Monitoring agent, Database agent, Feedforward agent, Feedback agent, Gain Optimizer agent, and Neural Network agent, categorized in the three-tier hierarchy as shown in Fig. 3.2. Since the power plant model used in this thesis is relatively simple in terms of complexity, at least compared to much larger plants that exist today, there is only need for one of each of the agents mentioned. For larger power plant systems, such as the 600 MW plant used in [8], there would likely be multiples of some of the agent types. Also, once the managing infrastructure of the MAS is in place, any number of additional agents can be added to increase the

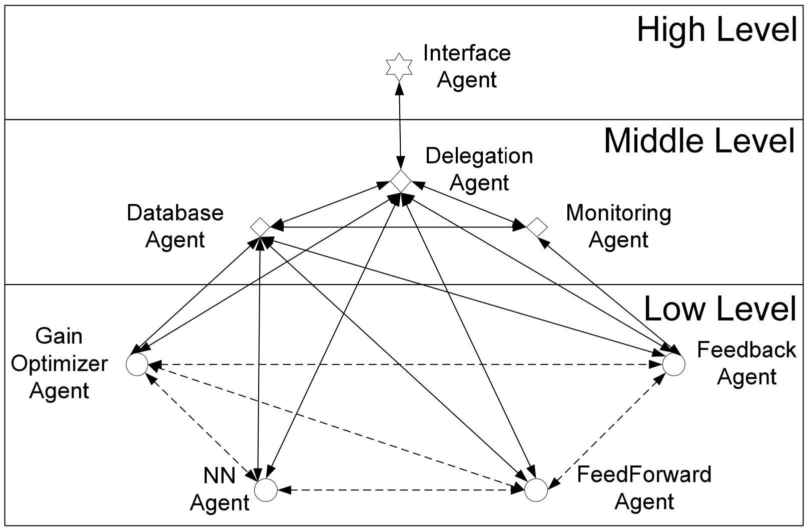


Fig. 3.2. Proposed agents divided into the MAS hierarchy.

functionality of the control system through the addition of more agent types. Regardless of how many agents the MAS control system may be designed to have, there should be enough additional agents in operation as Free agents to serve as backups for any of the agents in the case of an agent failure.

3.3.1 Interface Agent

The Interface agent is meant to allow an operator the ability to interface with the MAS. Using this interface, an operator will be able to monitor the MAS, the agents, and the power plant unit for performance, as well as configure control parameters in the control system, such as preferences and objective functions (see Section 2.3). Though this agent is mostly controlled by an operator, it is intelligent in that it has to decide how to carry out the commands given by the operator. For example, when requesting data, the Interface agent must determine the location of the appropriate agent, request the data, and make sure that it receives and processes the requested data to be sent to the operator. When relaying commands to the system, the Interface agent must again determine the location of the specific agent needed, relay the data, and make sure the command is carried out. If something goes wrong with an operation, it must retry the action, try to resolve the issue itself, alert the Delegation agent of a possible agent failure, or alert the operator that the action failed.

To interface with the MAS, the operator would use the specially designed graphical user interface (GUI) to communicate with the Interface agent which would intelligently carry out requested actions. The GUI translates the user's requests into MAS messages and sends them to the Interface agent. The Interface agent receives these messages the same way it would receive any other messages from the MAS. A flowchart showing the basic operation of the Interface agent is shown in Fig. 3.3.

3.3.2 Free Agent

When an agent other than the Interface agent is initiated, it will begin as a Free agent. When a Free agent starts, it begins a discovery process that explores the MAS network to determine what other agents and resources exist. It does this by polling the communication network and, upon finding an agent or resource, will determine what task it is performing or a description of its function as a resource. In doing this, the Free agent creates a local address table of the MAS, called an *agent directory*. If no Delegation agent is discovered during the exploration process, which is the agent that manages and assigns agent tasks, then the Free agent with the lowest Internet Protocol (IP) address will become the Delegation agent. This action is taken since the Delegation agent is essential for the organization and operation of the MAS. Upon becoming the Delegation agent, the agent will use its agent directory to manage and assign agents.

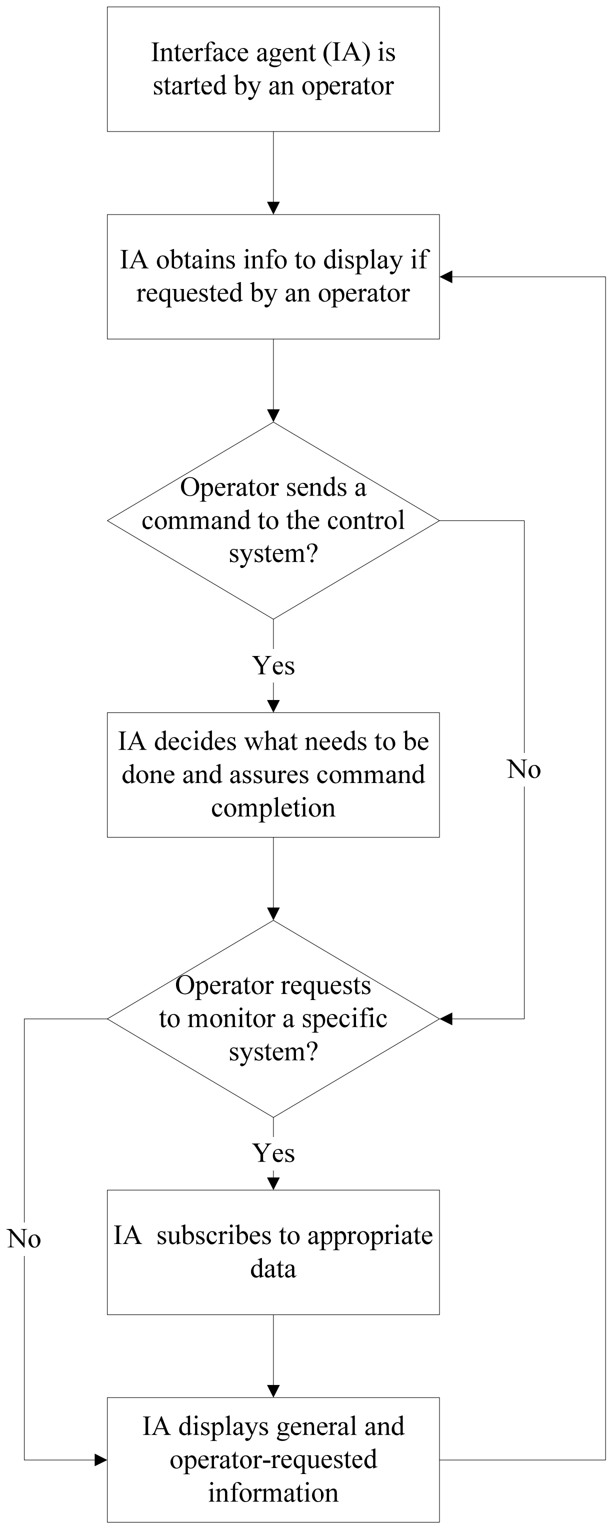

Interface agent (IA) is started by an operator
IA obtains info to display if requested by an operator
Operator sends a command to the control system?
Yes
No
IA decides what needs to be done and assures command completion
Operator requests to monitor a specific system?
Yes
No
IA subscribes to appropriate data
IA displays general and operator-requested information

Fig. 3.3. Flow diagram of the basic operation of the Interface agent.

In the case that the Free agent detects that a Delegation agent already exists, it will switch into stand-by mode, waiting to be assigned a task to perform. In stand-by mode, a Free agent will regularly poll the Delegation agent to ensure that it is still functioning properly. This way, if the Delegation agent fails, the first Free agent to detect this will assume the Delegation agent role, minimizing its downtime and the possibility that something will go wrong because of the failure.

After all needed agents are assigned, the remaining Free agents act not only as a backup for the Delegation agent, but as a form of redundancy for any of the other tasks should there be an agent failure. This helps to minimize downtime should an agent fail for any reason. However, instead of assuming the role of a failed agent as with the Delegation agent, a Free agent will wait to be assigned the missing task by the Delegation agent. Flowcharts of the Free agent's operation can be found in Figs. 3.4 and 3.5.

3.3.3 Delegation Agent

When a Free agent becomes the Delegation agent, it is responsible for assigning the tasks needed for the control system, as well as monitoring the MAS to ensure that all needed tasks are being performed. When the Delegation agent emerges, it will contain an up-to-date agent directory from which it will begin assigning tasks to the other Free agents. To do this, it will use a *prioritized agent assignment list* to make sure that the most critical tasks are assigned first, and work its way down to the least critical.

For the agents proposed in this thesis, the prioritized agent assignment list begins with the Feedback and Feedforward agents, and proceeds with the Neural Network agent, Gain Optimizer agent, Monitoring agent, and Database agent. This list is set such that if agent failures occur, the agents that are most important to the continued, stable function

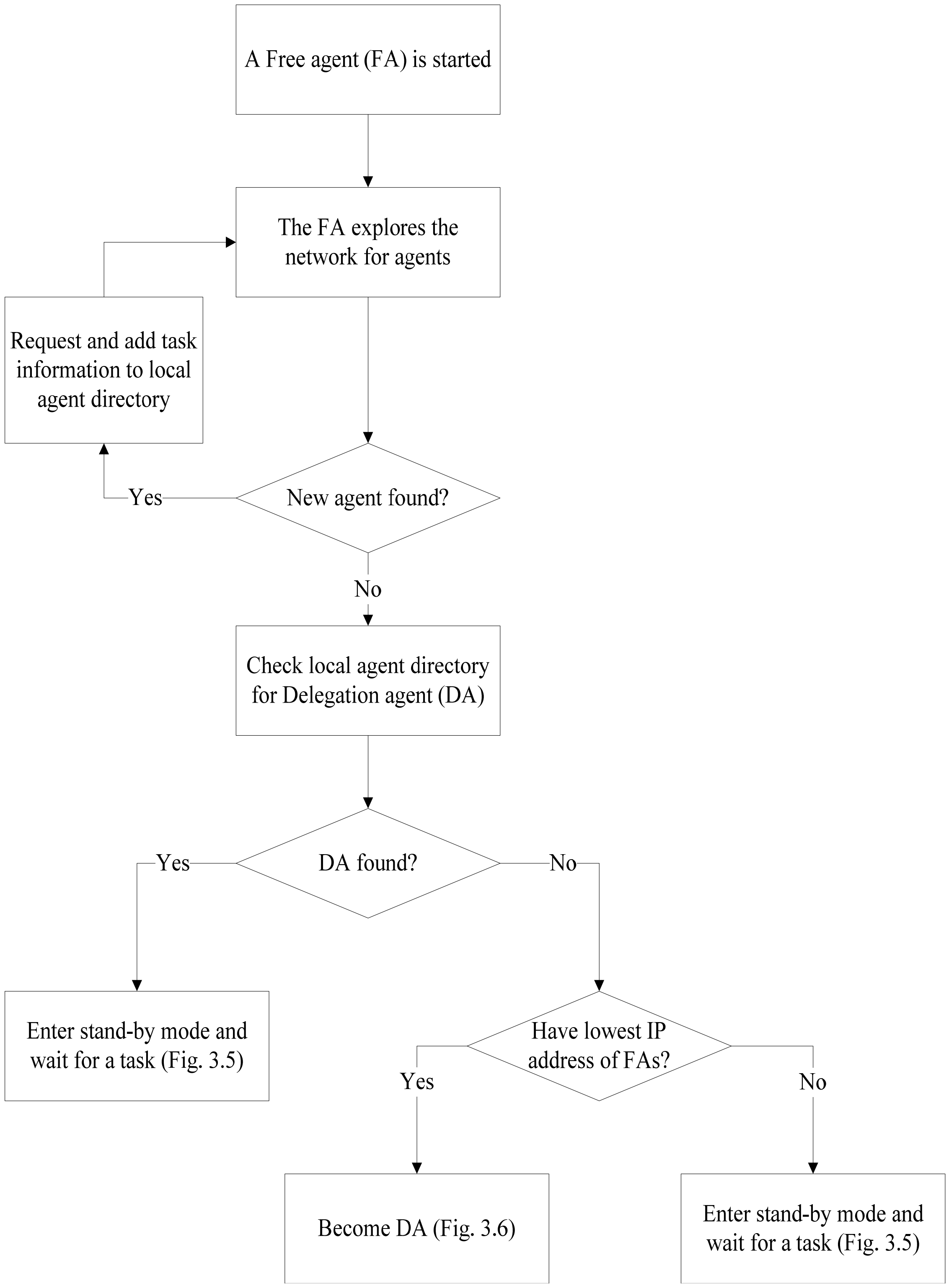

A Free agent (FA) is started
The FA explores the network for agents
Request and add task information to local agent directory
Yes
New agent found?
No
Check local agent directory for Delegation agent (DA)
Yes
DA found?
No
Enter stand-by mode and wait for a task (Fig. 3.5)
Have lowest IP address of FAs?
Yes
No
Become DA (Fig. 3.6)
Enter stand-by mode and wait for a task (Fig. 3.5)

Fig. 3.4. Flow diagram of the basic operation of the Free agent.

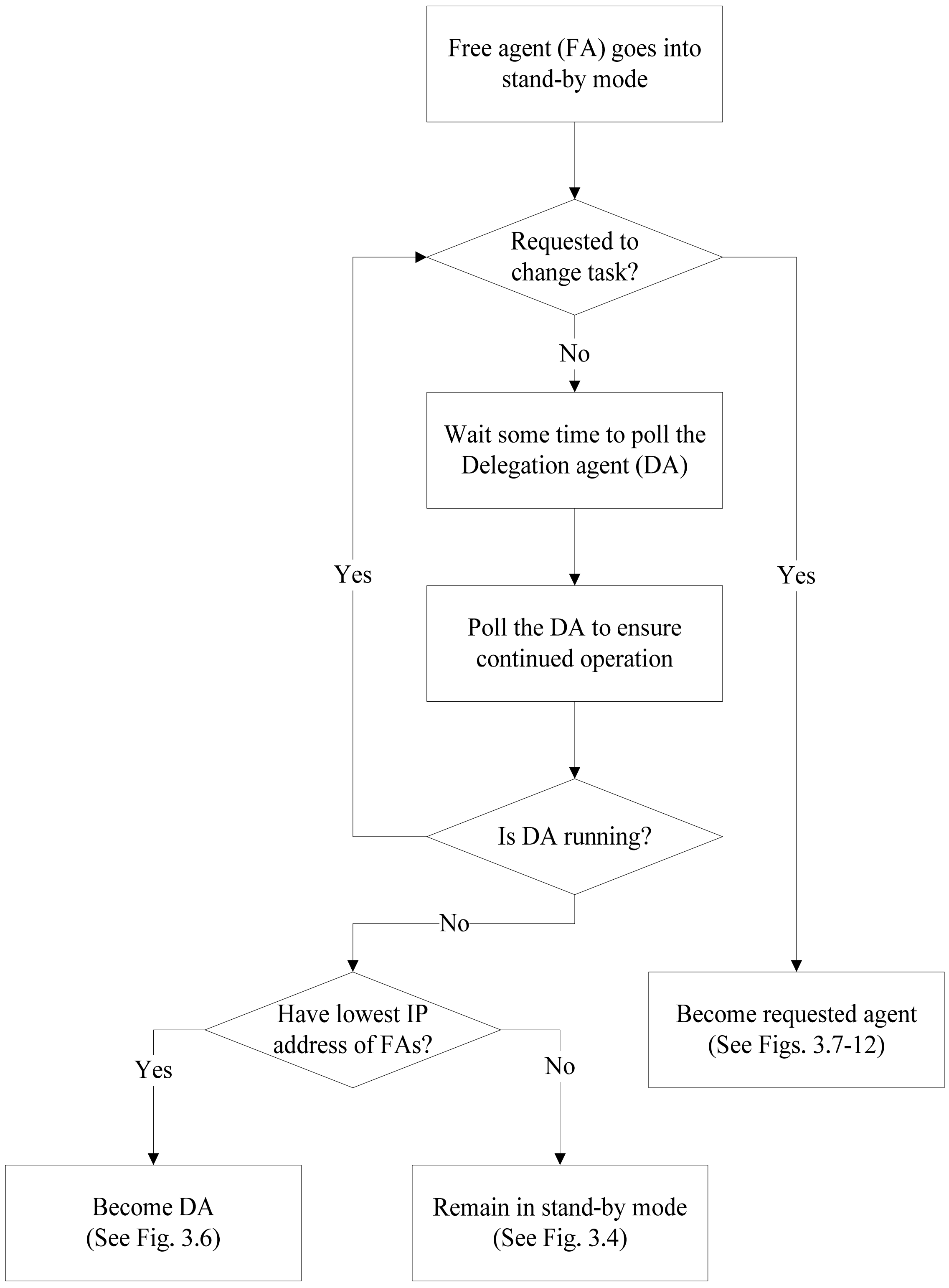

Free agent (FA) goes into stand-by mode
Requested to change task?
No
Wait some time to poll the Delegation agent (DA)
Yes
Yes
Poll the DA to ensure continued operation
Is DA running?
No
Have lowest IP address of FAs?
Yes
No
Become requested agent (See Figs. 3.7-12)
Become DA (See Fig. 3.6)
Remain in stand-by mode (See Fig. 3.4)

Fig. 3.5. Flow diagram of the basic operation of the Free agent in stand-by mode.

of the power plant are reassigned first. The reasoning for this order is discussed in the section on agent failures.

Once all of the agents have been assigned, the Delegation agent's task is to monitor the agents to ensure they are operating as expected. To do this, the Delegation agent will regularly send a *confirm operation* message to each of the agents, requesting a response that guarantees the agent is functioning. The Delegation agent will try a predetermined number of times to ascertain an agent's operational state without a response before it will assume the agent has failed and proceed with finding its replacement. Once it has been determined that an agent may have failed, the Delegation agent will quickly reassign the failed agent's task to a Free agent to prevent unintended operation of the control system due to the failure.

The Delegation agent also serves as a directory which can be used as a reference by other agents, because it already contains up-to-date information about the agents. This information includes IP address, task, as well as other agent specific information. A condensed form of the directory information will be stored in the other agents, referred to as the *agent directory*. The agent directory contains only agent address and communication information intended to be used for sending messages between agents. This information includes the task, network address, and incoming message port. In order to maintain a current version of the agent directory in each of the agents, the Delegation agent will send an updated version of this table as a part of the confirm operation message.

In the case that the Delegation agent fails, the Free agents will detect the failure, because they are polling the Delegation agent to make sure it is still functioning. When the Free agents realize that the Delegation agent has failed, they will use their agent directories to determine which Free agent has the lowest IP address and that Free agent will assume the Delegation agent role. The general operation of the Delegation agent is shown in flowchart form in Fig. 3.6.

3.3.4 Feedforward and Feedback Agents

The first two agents that are assigned by the Delegation agent are the Feedforward and Feedback agents. These two agents comprise the smallest subset of the proposed agents with which it would be possible to establish optimized control of the power plant. This minimal control system is depicted in Fig. 3.7.

The Feedforward agent performs the functionality of the reference governor and the feedforward controller as they are described in Section 2.3. In doing so, the Feedforward agent generates optimized control setpoints and output level references, according to the current operating objectives and preferences set by the operator using the Interface agent, corresponding to the demand power given by the unit load demand from economic dispatch. As previously described in Section 2.3, the reference governor uses a HPSO algorithm to find the optimized control setpoints. In the process of finding the optimized control setpoints, the corresponding drum steam pressure setpoint is also found using the FFNN and used as an output level reference. Therefore, both the setpoint optimization and feedforward control are performed by the Feedforward agent.

At any time in the operation of the MAS control system, an operator can change the operating objectives and preferences that govern the optimization process. The preference values are easily changed by using the Interface agent to send the Feedforward agent a message requesting the change. Upon receiving this request, the Feedforward

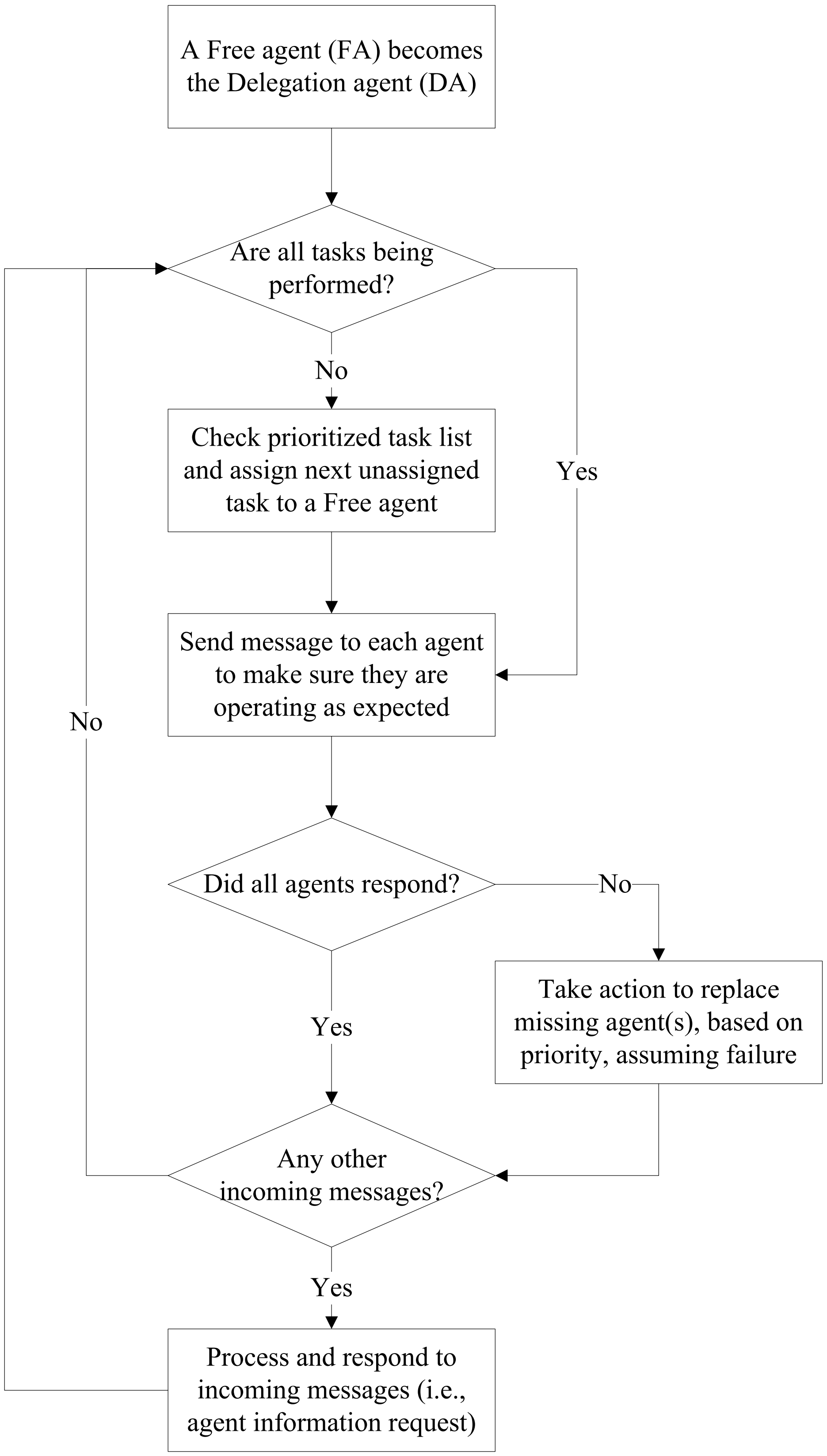

A Free agent (FA) becomes the Delegation agent (DA)
Are all tasks being performed?
No
Yes
Check prioritized task list and assign next unassigned task to a Free agent
Send message to each agent to make sure they are operating as expected
No
Did all agents respond?
No
Take action to replace missing agent(s), based on priority, assuming failure
Yes
Any other incoming messages?
Yes
Process and respond to incoming messages (i.e., agent information request)

Fig. 3.6. Flow diagram of the basic operation of the Delegation agent.

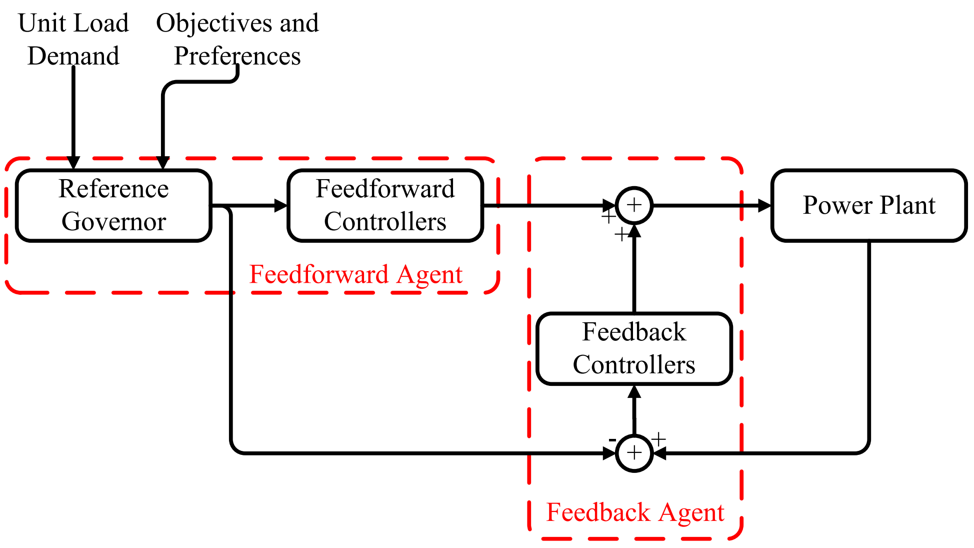


Fig. 3.7. Block diagram of the minimal control structure formed by the Feedforward and Feedback agents.

agent will update its memory with the new values and use them the next time it performs the control optimization.

Operating objectives can also be changed real-time using the Interface agent. By changing the preference values the operating objectives can be activated by assigning a nonzero value as its preference and deactivated by assigning it a preference value of zero. However, to add new operating objectives, the agents' code must be reprogrammed with the new objectives and the agents restarted so that the code is updated. This is necessary, because the operating objectives are lines of code that define the cost functions in the control optimization.

The Feedback agent performs the functionality of the feedback controllers, which are described in Section 2.4. The input to the Feedback agent from the MAS is the output level references and control setpoints calculated by the Feedforward agent, the output recorded from the sensors connected to the power plant, and the PI gains optimized by the Gain Optimizer agent. The output level references and the actual output recorded from the power plant are used to calculate the errors that drive the feedback controllers. Using these error calculations and the gains from the Gain Optimizer agent, the feedback compensation values are calculated, added to the control setpoint values, and applied to the power plant by sending the resulting values to the valve actuators.

To provide consistency in the timing of the application of feedback compensation, the Feedback agent was designed to send control values to the power plant valve actuators every t_s seconds. The value of t_s is a design parameter and also depends on the performance capabilities of the hardware implementing the Feedback agent, in that the hardware must be able to receive incoming messages, process them, and perform the feedback calculations in less than t_s seconds. Another design consideration in determining the length of t_s is how quickly the outputs of the power plant can be sampled. For example, there is no benefit from t_s being shorter than the possible sampling time of the sensors that are sampled for power plant output values, since the feedback compensation results will not change until new power plant output data is received.

Another function of the Feedback agent is to monitor the error in plant output for the Gain Optimizer agent, since the gain optimization process in the Gain Optimizer agent is triggered when the error between the output level references produced by the Feedforward agent and the actual output recorded from the power plant exceeds the preset thresholds. The Feedback agent is used for this purpose, because it already calculates the errors, and it is only a simple step to send these values to the Gain Optimizer agent when a violation occurs. Therefore, the Gain Optimizer agent doesn't have to continuously request data from the Feedforward agent and the power plant

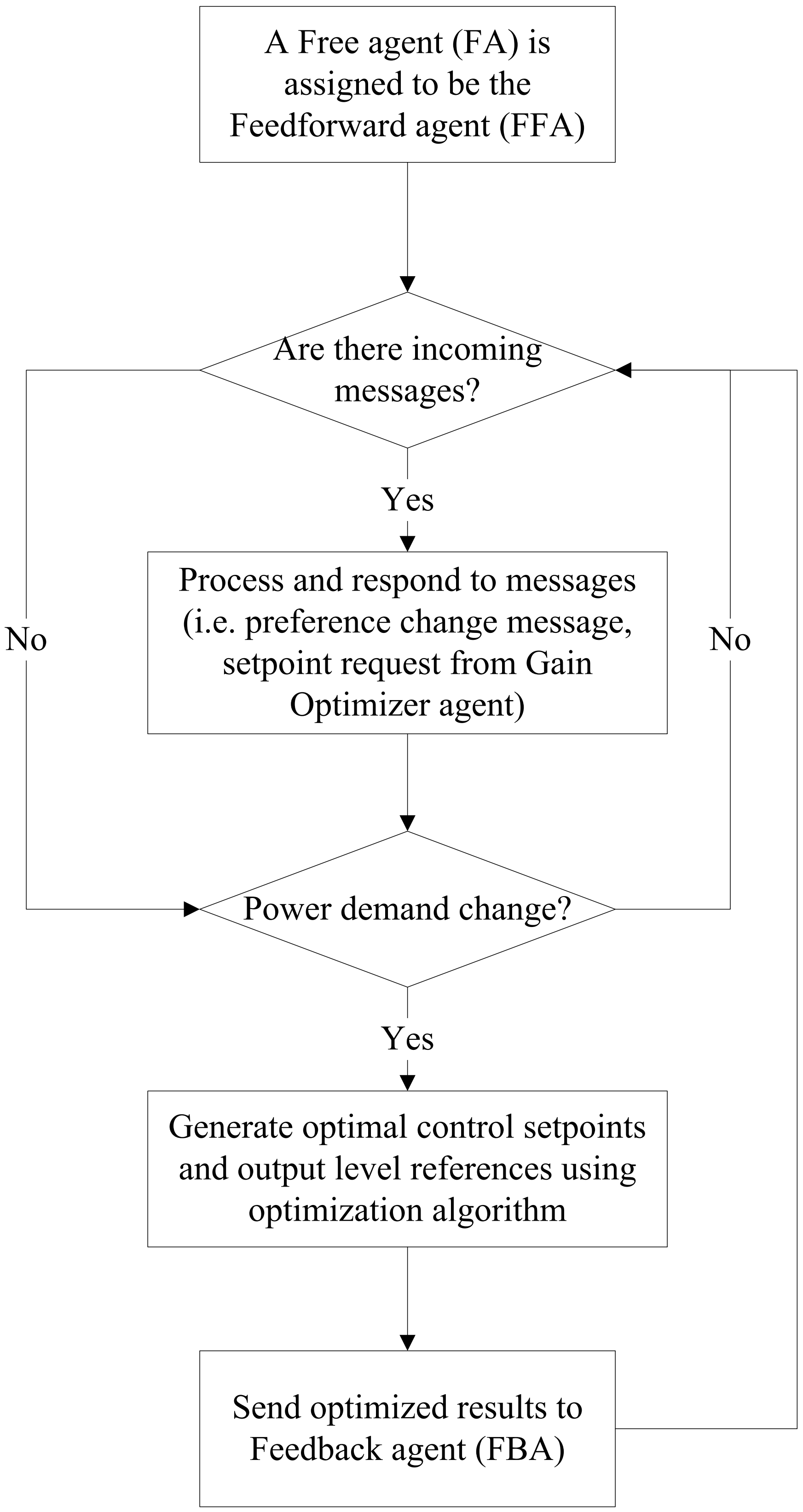

A Free agent (FA) is assigned to be the Feedforward agent (FFA)
Are there incoming messages?
Yes
No
No
Process and respond to messages (i.e. preference change message, setpoint request from Gain Optimizer agent)
Power demand change?
Yes
Generate optimal control setpoints and output level references using optimization algorithm
Send optimized results to Feedback agent (FBA)

Fig. 3.8. Flow diagram of the basic operation of the Feedforward agent.

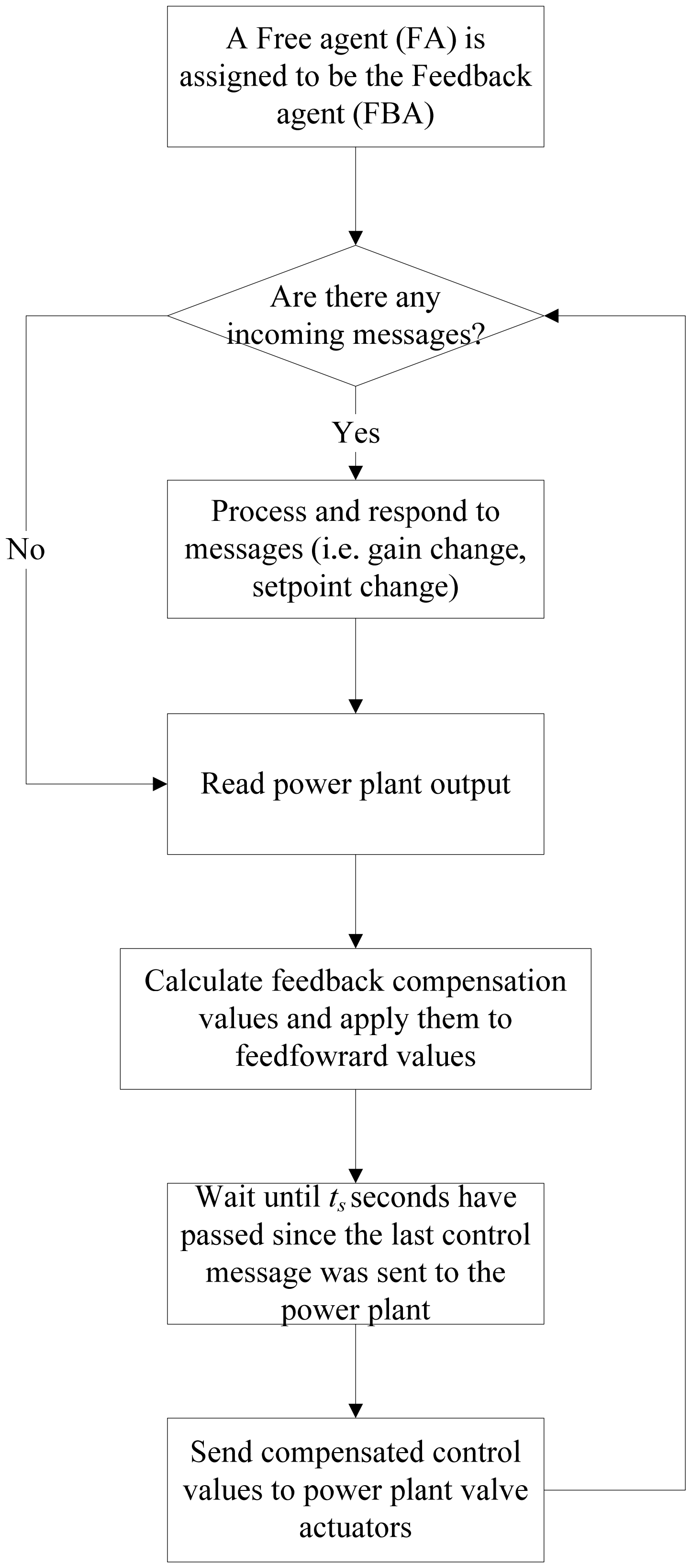

A Free agent (FA) is assigned to be the Feedback agent (FBA)
Are there any incoming messages?
Yes
No
Process and respond to messages (i.e. gain change, setpoint change)
Read power plant output
Calculate feedback compensation values and apply them to feedfowrard values
Wait until t_s seconds have passed since the last control message was sent to the power plant
Send compensated control values to power plant valve actuators

Fig. 3.9. Flow diagram of the basic operation of the Feedback agent.

sensors unnecessarily to calculate the errors itself. Flowcharts describing the operation of these agents can be found in Figs. 3.8 and 3.9.

3.3.5 Neural Network Agent

The Neural Network agent is assigned next, as it is needed by the Gain Optimizer agent to maintain the LRNN model. The task of the Neural Network agent is to fine-tune the dynamic ANN models, in this case one LRNN, used in the control system to simulate the current operating region more accurately. As mentioned in Section 2.6, this is needed because the global training of the LRNN produces a network that is equally good at simulating power plant operation in the entire stable operating region, but is not accurate enough in a specific region of operation to give an acceptable result.

Therefore, the Neural Network agent continually collects power plant input-output data from the power plant to fine-tune the LRNN. Since the Neural Network agent is constantly updating the LRNN, it will always be most accurate at simulating the current level of operation. This is important, because, as mentioned in the last section, the Gain Optimizer agent will request the current version of the LRNN as it starts the optimization process in order to have the most accurate model to simulate the power plant response. This gives the Gain Optimizer agent an LRNN that has been tuned to simulate the level of operation for which gain values will be tested. The Neural Network agent will continue to update the LRNN in preparation for future gain optimizations, sending the current version of the LRNN, when requested, as long as the Neural Network agent task is assigned. A flowchart describing the Neural Network agent's operation is shown in Fig. 3.10.

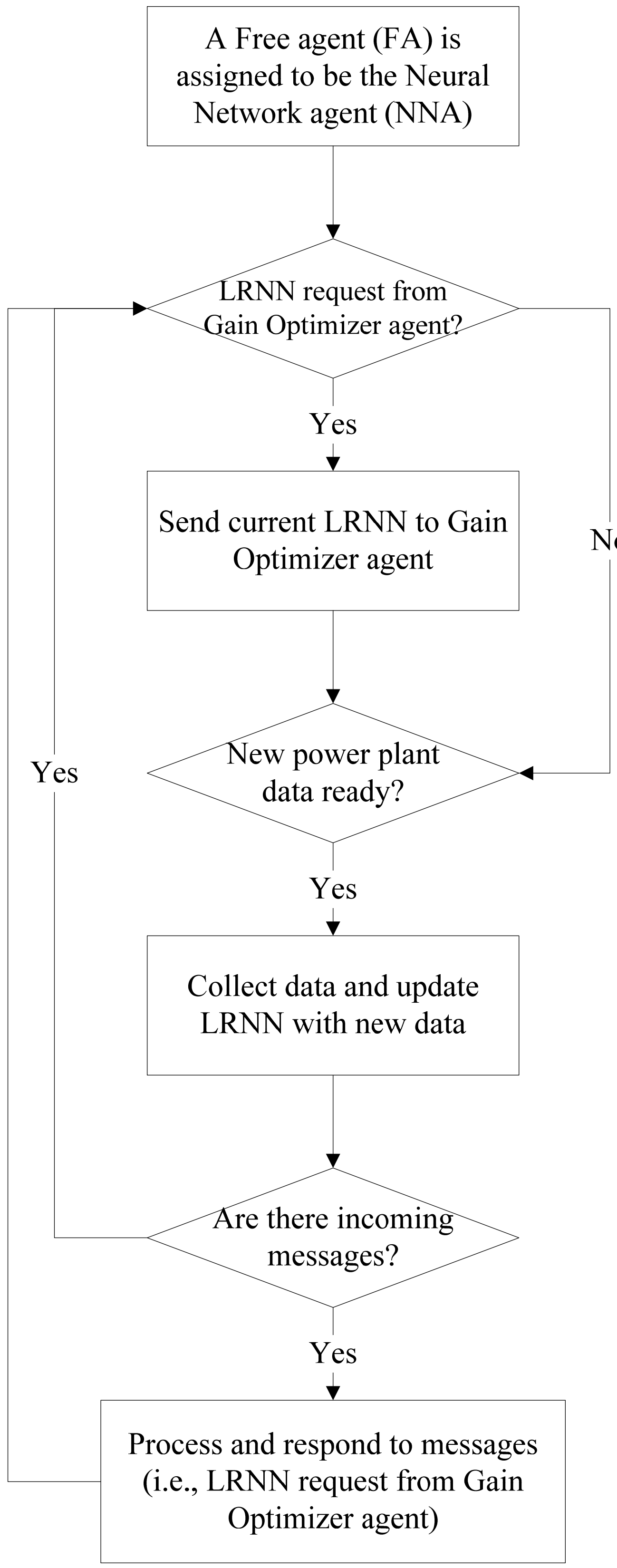

A Free agent (FA) is assigned to be the Neural Network agent (NNA)
LRNN request from Gain Optimizer agent?
Yes
No
Send current LRNN to Gain Optimizer agent
New power plant data ready?
Yes
Yes
Collect data and update LRNN with new data
Are there incoming messages?
Yes
Process and respond to messages (i.e., LRNN request from Gain Optimizer agent)

Fig. 3.10. Flow diagram of the basic operation of the Neural Network agent.

3.3.6 Gain Optimizer Agent

The Gain Optimizer agent is the last of the proposed low level agents to be assigned. The Gain Optimizer agent performs the functionality of the gain optimizer, which is described in Section 2.5. This agent waits until the error between the output level references produced by the Feedforward agent and the actual output levels produced by the power plant are reported to be above the predetermined threshold values by the Feedback agent, signifying that better feedback gains are needed to achieve tighter control of the power plant.

Once one or more of the error level thresholds are reached, the Gain Optimizer agent will begin the process of optimizing new feedback gain values. The first step in this process is to request control setpoints and output level references for the current power level and a power level that is a predetermined step-size higher from the Feedforward agent. These setpoints and references are then used by the Gain Optimizer agent to form a test vector that can be used by the HPSO to evaluate candidate gain sets by simulating the power plant response to feedback control using the candidate gain sets. The next step in the optimization process is to request the current LRNN from the Neural Network agent. The LRNN is the model used to determine the cost of a gain set by comparing the simulation data to the desired output response defined by the output level references in the test vector. As described in Section 2.5, the optimization will choose the gain set with best performance, or lowest cost, as optimized.

Once an optimized gain set is found, it is sent to the Feedback agent for immediate implementation. Once the gains are sent, the optimization process is complete and the Gain Optimizer agent will stand-by until an error threshold is violated to perform the optimization process again. The Gain Optimizer agent will optimize feedback gains, as needed, as long as the task is assigned. A flowchart describing the operation of this agent can be found in Fig. 3.11.

3.3.7 Database and Monitoring Agents

The Database and Monitoring agents are the last agents to be assigned. Assigning these agents completes the establishment of the middle level in the hierarchy of the MAS. These agents are lower in priority than the low level agents, because the low level agents perform the tasks that control the power plant, whereas these two agents provide ancillary services.

The Database agent handles data files needing to be archived or distributed for use by other agents. In essence, the Database agent is an intelligent fileserver that keeps track of where data is stored on the network, and can deliver old or archive new data upon request. Whenever an agent needs to archive data, it sends it to the Database agent, and whenever an agent needs data that has been archived, it will request it from the Database agent. Data handled by the Database agent includes power plant input-output data, artificial neural networks (ANNs), and control system performance data used for analysis of control system performance.

The Monitoring agent is intended to monitor different parts of the physical plant system in order to detect situations demanding immediate attention and action from the MAS. This functionality was not developed in this thesis, but possible scenarios in which the Monitoring agent would be desirable are those such as detecting and reacting to electrical faults in the grid system and shutting down or switching equipment in the case of an imminent failure in part of the physical plant system.

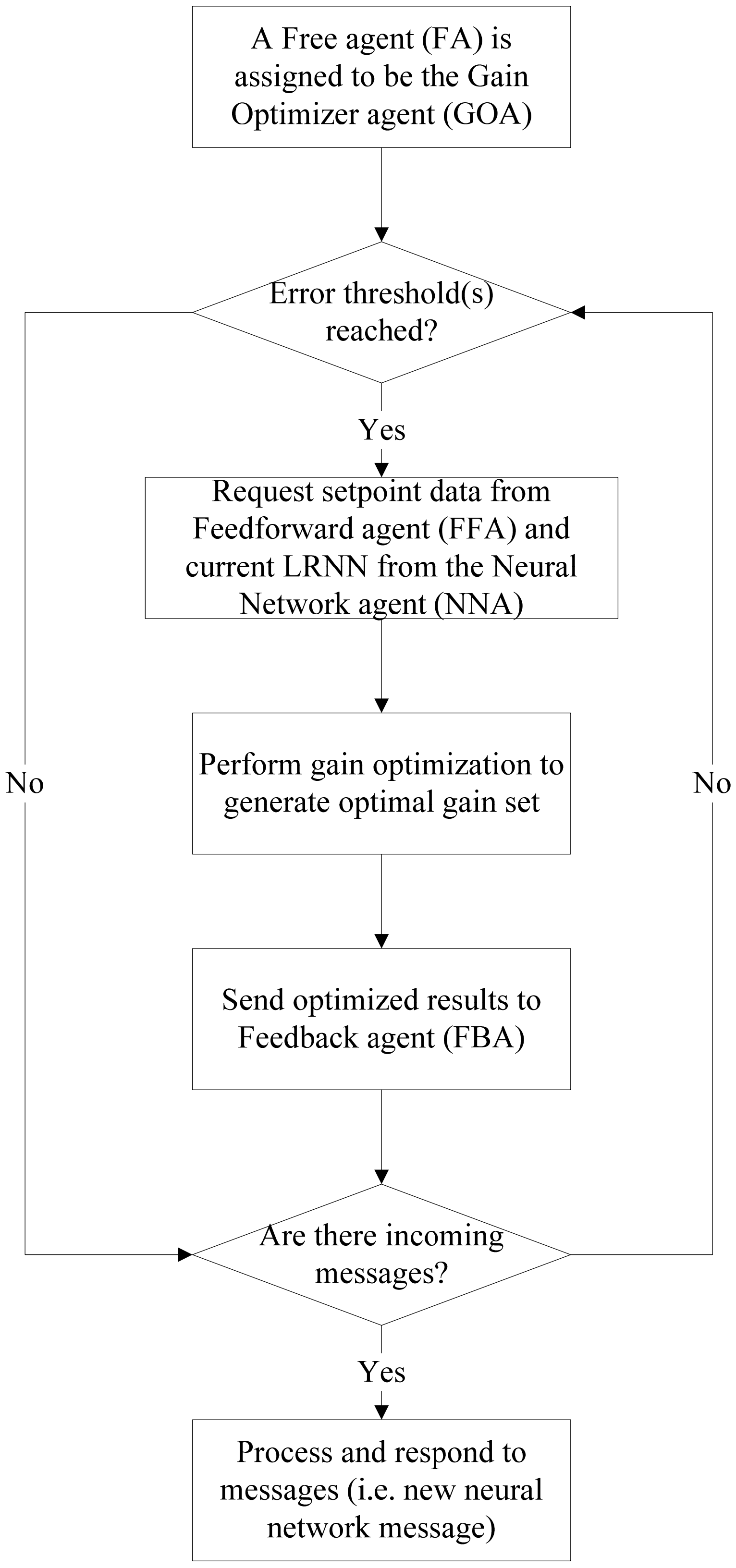

A Free agent (FA) is assigned to be the Gain Optimizer agent (GOA)
Error threshold(s) reached?
Yes
No
No
Request setpoint data from Feedforward agent (FFA) and current LRNN from the Neural Network agent (NNA)
Perform gain optimization to generate optimal gain set
Send optimized results to Feedback agent (FBA)
Are there incoming messages?
Yes
Process and respond to messages (i.e. new neural network message)

Fig. 3.11. Flow diagram of the basic operation of the Gain Optimizer agent.

Since this aspect of control has not been focused on here, the Monitoring agent monitors and records data from the power plant system for the purpose of analyzing the performance of the control system and overall power plant operation. This is done by collecting power plant data and storing the data in memory until the data reaches the predetermined file size limit. Once the limit is reached, the data in memory is sent as a file to the Database agent for storage. After sending the data file, the Monitoring agent will start a new file and repeat this process for as long as it is functioning. Flowcharts describing the operation of these agents in diagram form can be found in Figs. 3.12 and 3.13.

3.4 Agent Communication

The agents defined in this thesis use an Agent Communication Language (ACL) to communicate using the User Datagram Protocol (UDP) over a computer network. This ACL is a subset of the one developed by the Foundation for Intelligent Physical Agents (FIPA), where only the needed functionality was included [22,23]. The custom version of the FIPA-ACL will be referred to as the MAS-ACL in this thesis. The MAS-ACL defines the structure of the language used in messages sent between agents. The MAS-ACL has four required fields with an optional fifth field. As shown in Fig. 3.14 the message fields are the *performative*, *sender*, *receiver*, *content*, and *replyby* fields.

The performative field contains a keyword that informs the receiving agent of the purpose for which the message was sent. The performative keywords are *request*, *inform*, *subscribe*, *agree*, *refuse*, and *not-understood*. The request performative informs a receiving agent that the message it is receiving is to be processed as a request to do something, such as send information or change task. The inform performative informs the

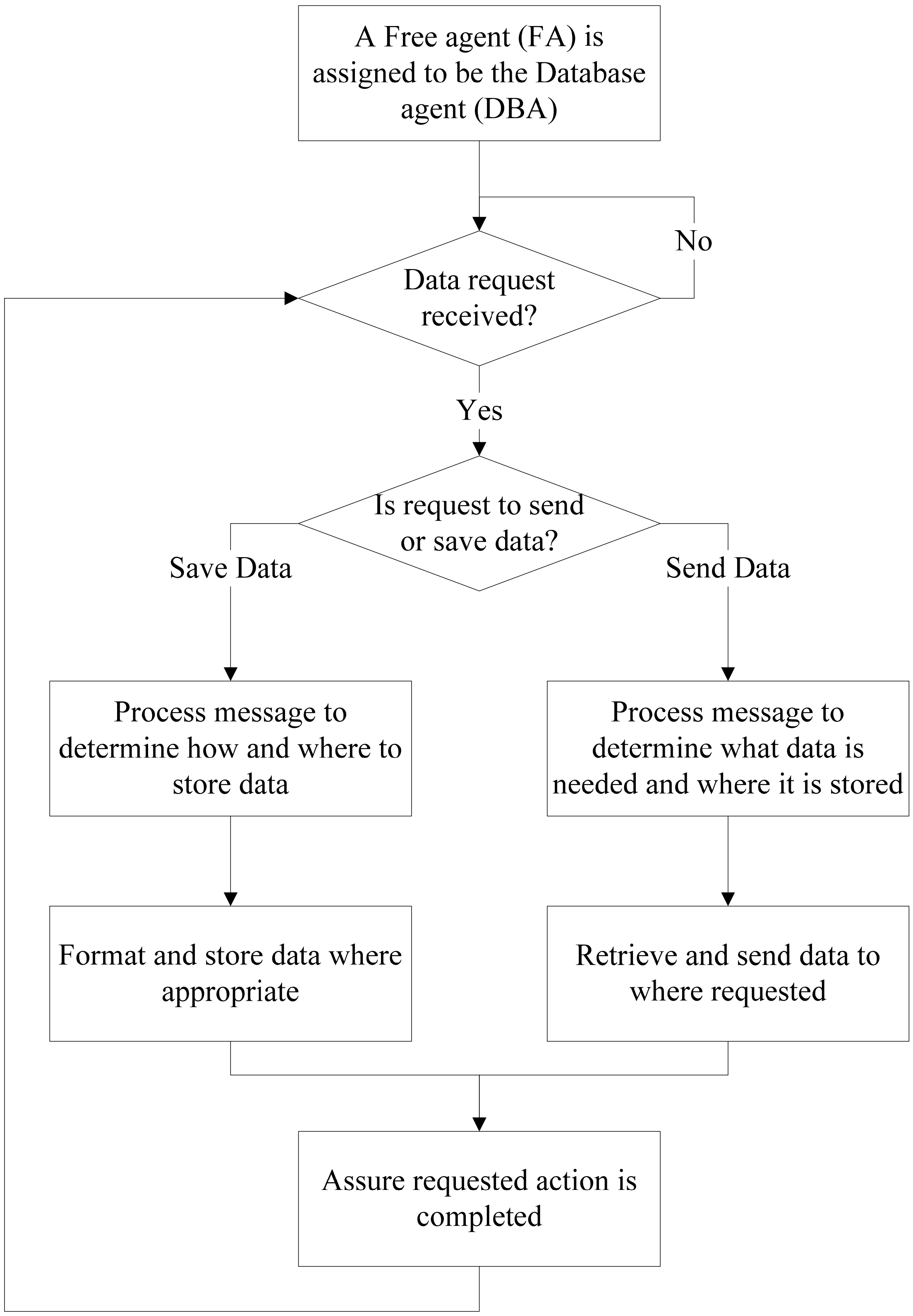

A Free agent (FA) is assigned to be the Database agent (DBA)
No
Data request received?
Yes
Is request to send or save data?
Save Data
Send Data
Process message to determine how and where to store data
Process message to determine what data is needed and where it is stored
Format and store data where appropriate
Retrieve and send data to where requested
Assure requested action is completed

Fig. 3.12. Flow diagram of the basic operation of the Database agent.

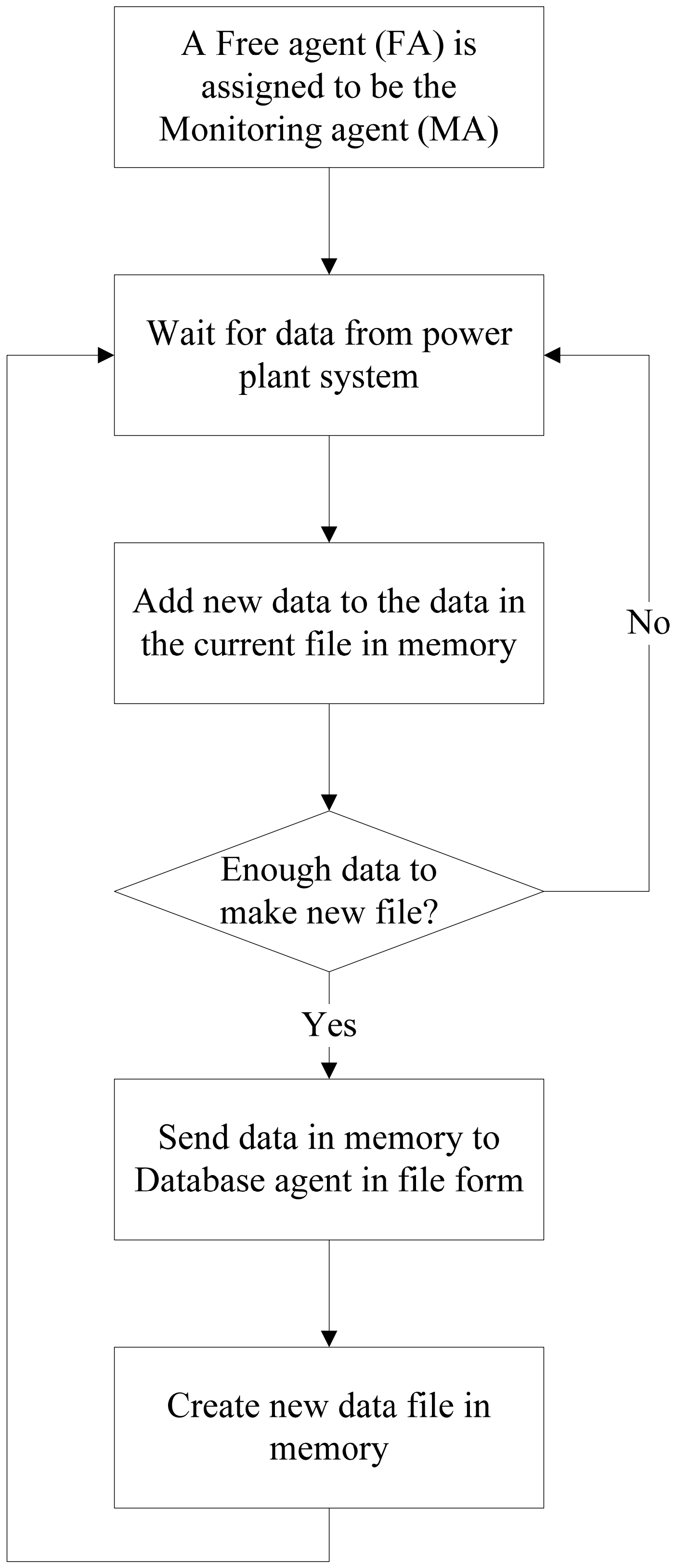

A Free agent (FA) is assigned to be the Monitoring agent (MA)
Wait for data from power plant system
Add new data to the data in the current file in memory
Enough data to make new file?
No
Yes
Send data in memory to Database agent in file form
Create new data file in memory

Fig. 3.13. Flow diagram of the basic operation of the Monitoring agent.

Performative	Sender	Receiver	Content	Replyby

Fig. 3.14. Message structure of the MAS ACL.

receiving agent that the received message contains information that should be extracted for processing, such as power plant data. The subscribe performative informs the receiving agent that the sending agent wants to be informed any time a certain piece of data changes, such as when the Monitoring agent has new input-output data for the Neural Network agent. The agree performative informs the receiving agent that the sending agent is agreeing to a request message, such as when an agent is requested to change task. The refuse performative is used by a sending agent to inform the receiving agent that it is refusing a requested action, because it is either unable to perform the specified function, or it is involved with something which takes precedent over the requested task. The not-understood performative is used by a sending agent to notify the receiving agent that a previous message sent by the receiving agent was not understood, possibly because the message was corrupted in transmission.

The sender field contains information about the agent sending a message. This information is used by the receiving agent to send a reply message if a response is needed for the type of message received. The information consists of an IP address on the computer network, the port number it is listening on for communication, and the sending agent's current task.

The receiver field contains information about the agent the message is intended for. This information is used by the receiving agent to make sure the message was sent to the right agent. This helps in the case where an agent task has changed location on the computer network unbeknownst to the sending agent. The identification information is of the same type and format as in the sender field, including IP address, port number, and current task.

The content field, depending on the type of message, will contain things like data, the location of data, an instruction, or possibly nothing at all. Data is sent when the sending agent is informing the receiving agent of data, and it is concise enough to send in the actual message. Location information for data is sent when the data is too large in size to efficiently send in the message. This information is used to request data from the Database agent. An instruction may be sent when the sending agent is requesting something of the receiving agent, such as to change its task. The content field is left blank when the performative is sufficient for communicating what needs to be done, such as when an agent uses the agree performative when agreeing to a request. However, when an agent uses the refuse performative to refuse a request, a reason for refusal may be sent in the content field.

The replyby field is an optional field used by a sending agent to signify that the sent message is of high importance, and should be responded to in the time specified. This field is used when the sent request is of a time-sensitive nature. For example, if the Delegation agent needs to verify that an agent is still functioning properly, it can request that the agent respond by a certain time. If the agent does not respond in the given amount of time after a specified number of tries, the Delegation agent will assume the agent is in a failed state and assign its current duties to another agent.

To add messaging capability to the MAS-ACL, more of the FIPA-ACL can be added to the message processor. The message processor is the code that is common to each agent that processes and interprets incoming messages. As the messages are interpreted, a receiving agent can then respond appropriately.

3.5 Agent Failure

One of the appealing characteristics of a MAS framework is the potential to recover quickly when agents fail so that stable power plant operation is maintained. Though agent failures were not tested in this thesis, a discussion of some considerations for handling them is given to provide some suggestions for future work and for completeness.

3.5.1 Agent Operating State

In order to successfully recover from agent failures, data defining the operating state of the agents would need to be saved regularly, in a central location, so that when an agent fails, its last operating state can be restored in the new agent assigned to perform its task. For example, the Feedback agent would save data such as the current feedback gains so that they would not be lost in the event that this agent failed. The Feedforward agent would save data such as the current unit load demand and the preference values being used. In general, the operating data saved by the agents would include data that either could not otherwise be recovered in the event of a failure, or would be difficult to infer.

This concept is implemented, to some degree, in the MAS developed in this thesis. The Delegation agent sends the agent table to the agents as a part of the confirm operation message, so that all agents have a copy of this data in case the Delegation agent were to fail. Therefore, the Free agent that assumed the role of the Delegation agent after the failure would already have this information and could begin performing its new task immediately.

Similar to this procedure, one possible way to save operating state data with the MAS architecture developed in this thesis would be for the agents to respond to the Delegation agent's confirm operation message with operating state data so it can stored locally by the Delegation agent and sent to the Database agent for backup storage. This way, if an agent failed, the Delegation agent can reassign the task and initialize the new agent to begin where the previous agent left off. Also, by sending a copy of the data to the Database agent to be archived, it can be recalled if the Delegation agent where to fail and lose its local copy.

3.5.2 Prioritized Agent Assignment List

Another important aspect of dealing with agent failures is deciding the order in which agents are reassigned if multiple agent failures occur simultaneously. This section briefly discusses some considerations for choosing the order for agent reassignment with respect to the agents proposed in this thesis. As part of this discussion, possible behaviors of the MACS due to failures of each agent type are described.

One way to implement the reassignment of agents in case of agent failures is to provide the Delegation agent with a prioritized agent assignment list to allow it to reassign the most critical agents first. The order of the agents on this list should be such that if all agents with assigned tasks other than the Delegation agent failed, restoring the agent tasks in this order would have the least potential to allow the controlled system to become unstable. Furthermore, it is desirable to assign the agents in such a way that normal operation of the power plant unit is restored quickly. For example, in this thesis, this list starts with the Feedback and Feedforward agents and continues with the Neural Network agent, Gain Optimization agent, Monitoring agent, and Database agent.

The list starts with the Feedback agent, because if this agent fails, it would need to be replaced quickly, as it would leave the power plant unit uncontrolled and operating at the last control values sent before failure. If the power level was being changed at the time of failure, it is uncertain what steady-state output would result from the last feedback compensated control values sent to the power plant. Therefore, this agent is given the highest priority for reassignment.

The Feedforward agent is listed next as it is used frequently to generate control setpoints, but does not have the same potential to cause instability in the operation of the power plant unit as the Feedback agent if it failed. If the Feedforward agent failed, the Feedback agent would continue to control the power plant at the last setpoint sent by the Feedforward agent. The most trouble that would seem to be caused by the failure of this agent is a possible delay in changing the power level until a new Feedforward agent could be assigned.

The Neural Network agent follows the Feedforward agent on the list, preceeding the Gain Optimizer agent, because the Gain Optimizer agent depends on the Neural Network agent to tune the ANN model it uses to optimize gains. If the Neural Network agent failed, the worst case scenario would include the Gain Optimization using a less accurate ANN to optimize feedback gains, possibly producing an unstable result. However, it seems unlikely that the ANN model would become inaccurate enough to produce such a result unless the power level changed significantly while the Neural Network agent was down.

Next on the list is the Gain Optimizer agent, which is the last control task of the agents proposed in this thesis. If this agent failed, the worst case would include the MACS calling for gain optimization while the agent was down. As long as the feedback gains are stable, this would not cause a stability problem. The gains produced by the Gain Optimizer should be stable, since a model of the controlled system is used to evaluate the gains for effectiveness and stability before they are used. However, this assumes the model is accurate. Another reason this agent is the last control agent to be assigned is that the Gain Optimizer agent is used less frequently, making the probability of this agent failing during a time where its function is needed less than the others.

Last on the list are the Monitoring and Database agents. These agents are last, because they are not likely to cause unstable behavior if they were to fail. The consequences caused by a failure of the Monitoring agent would depend on its intended function. In the case where the Monitoring agent would perform fault diagnosis, it would not have an effect on the control system unless the monitored equipment was to fail with little warning while the agent was down. This is unlikely because plant equipment does not generally fail frequently or without warning.

If the Database agent were to fail, the consequences could include delays in the retrieval and storage of control system data. The effect this would have depends on the types of data that are archived, what the data are used for, and how often the data are retrieved. For the MAS described in this thesis, the data stored by the Database agent are not vital to the operation of the control system. Therefore, this agent is the last to be assigned.

CHAPTER FOUR
Simulation and Results

This chapter discusses the simulation and testing of the multi-agent control system (MACS). In explaining the simulation of the MACS, the software implementation of the agents and power plant model are discussed. To explain how the MACS was tested, the methods for testing the individual agents, as well as the entire control system, are discussed. Finally, the results of the MACS tests are discussed to explain how the agents and the control system as a whole was shown to perform as intended.

4.1 Simulation of the Multi-Agent Control System

This section explains how the MACS is simulated. This explanation includes a description of how the MAS and the power plant model are implemented in software. Also, development tools used to create these software are mentioned.

4.1.1 MAS Simulation

The agents comprising the MAS developed in this thesis were programmed in Matlab version 2010a and implemented on computers connected by a computer network. To enable multiple agents to run on one computer, the Matlab Parallel Computing Toolbox (PCT) was used. One component of the PCT is the Interactive Parallel Command Window (IPCW), which allows the user to instantiate a specified number of separate Matlab instances, called labs. Regardless of how many labs are started, the IPCW uses one command line to interface with the labs.

To enable each of the labs to run separate code, a single script is run from the IPCW command line that uses a switch statement to run code based on each lab's *lab index*. A lab index is a unique, integer identifier assigned by Matlab. For example, if four labs are used, each lab is assigned an integer, 1 through 4, as its lab index. By default, the maximum number of labs that Matlab allows to run at once is equal to the number of processor cores available.

Since Matlab is not capable of multi-threading, each agent requires two labs, one for its task thread and one for its messenger thread. As previously described, the task thread runs the code that performs the agent's task, and the messenger thread runs the code that receives and preprocesses messages sent to the agent. Since the labs do not share memory, memory synchronization is initiated by the messenger thread after every message received to update the task thread memory with the data processed from the message. The synchronization uses commands available in the PCT that allows the labs to be programmed to check for and send data between themselves.

The computers used to run the MAS have eight processors, meaning they can run eight labs each, which is enough for four agents. To start the agents needed for a simulation of the MACS, the IPCW is started on each computer with eight labs. Then, a script is manually run in the IPCW command line that starts the Free agent code running on odd numbered labs and corresponding messenger threads on the even numbered labs.

The Free agent code will search the computer network for other agents and system resources by polling a predefined range of IP addresses and port numbers. When the Free agents finish exploring the network and discover there is no Delegation agent, the Free agent with the lowest IP will assume the role of that agent. Once the Delegation agent is established, it will begin assigning the needed tasks to the other Free agents according to the prioritized list mentioned in Section 3.3.3.

Once all other agents have been assigned by the Delegation agent, they wait for an initialization signal that propagates through the MAS when the Interface agent GUI, discussed in a later section, is used to send the unit load demand curve (ULDC) to the Feedforward agent through the Interface agent. The ULDC describes how the unit load demand should vary with time for the current simulation. After receiving this information from the Interface agent, the Feedforward agent begins generating setpoints at a predetermined frequency, in real-time, according the ULDC. As the setpoints are generated, they are sent to the Feedback agent to calculate feedback control. The first setpoint sent starts the Feedback agent.

Once the Feedforward and Feedback agents are functioning, the power plant simulator (PPS) is started choosing an initial power level equal to the first unit load demand in the ULDC. Choosing the initial power level this way ensures the simulation will not become unstable because of a large initial error between the output level of the plant and the unit load demand. If this error gets too large, the feedback controller may overcompensate, potentially driving the system unstable.

The PPS, described in more detail in the next section, models the behavior of the 160 MW power plant in real-time. The PPS is connected to the computer network used by the MAS and can send and receive MAS messages. This allows the MAS to change the input values of the power plant model and allows the PPS to send calculated model output values to the MAS.

When the PPS is started, it will begin broadcasting the output values to a specific port number using the user datagram protocol (UDP), so that any agent that needs this information can use it. This behavior simulates the real world situation where the MAS would have access to sensors connected to an actual power plant unit. The Feedback, Monitoring and Neural Network agents are listening for messages sent on the PPS output port, and when the Monitoring and Neural Network agents receive their first message, they will begin performing their task. This behavior also allows the Feedback agent to collect the power plant output to calculate feedback values.

The Database and Gain Optimizer agents will continue to standby until they are needed to store or retrieve data or optimize gains, respectively. When the MAS is done with a simulation, the data stored by the Monitoring agent can be used to analyze the performance of individual agents, as well as the MACS as a whole. It is in this manner that the agents will be tested and shown to behave as intended in the following sections.

4.1.2 The Power Plant Simulator

The Power Plant Simulator (PPS) is a real-time software model of the 160 MW oil-fired drum-type boiler-turbine-generator unit, described in Section 2.2. The PPS is the dynamic system controlled by the MACS. The simulator was programmed in Matlab and communicates with the MACS over a computer network using UDP to send and receive MAS messages.

Running the PPS software will cause a GUI to appear that allows a user to start the simulator after specifying an initial power level. The power level is used to generate initial conditions, consisting of drum steam pressure, steam quality, and initial control values, for the ordinary differential equations (ODEs) that describe the power plant model. The initial conditions are generated using the HPSO algorithm used in the reference governor to find stable control values and drum steam pressure. Using the specified power level and the drum steam pressure level found in the previous step, another search is performed to find an initial steam quality that corresponds to an initial water level deviation of zero.

Once the initial conditions are found, the PPS will begin an endless loop, solving the ODEs using a time-step set by the user every for iteration of the loop. At the beginning of every iteration, the PPS checks for incoming communication from the MACS that would contain new control variables to be used in calculating the solution to the ODEs. This is how the PPS simulates changes in the control valve positions.

After checking for new control values, the PPS calculates the power, drum steam pressure, and water level deviation that would result from the specified time-step of power plant operation at the current input and state variable values. The third step in the loop is to record the final conditions of the ODE solution as initial conditions for use in the next iteration and send the calculated output values to the MACS by sending an MAS message. This message is broadcast to all computers connected to the MAS network that are listening on a specific port. The agents needing this information, such as the Feedback and Monitoring agents, would simply listen on this port to receive the data whenever it is sent. A flowchart of the PPS behavior is shown in Fig. 4.1.

The PPS simulates the power plant behavior in real-time by keeping track of the time the current loop took to execute and pauses execution for the difference of that amount of time and the specified time-step. By doing this, the simulation of one time-step of power plant dynamics will take that amount real time to execute. For the PPS to work

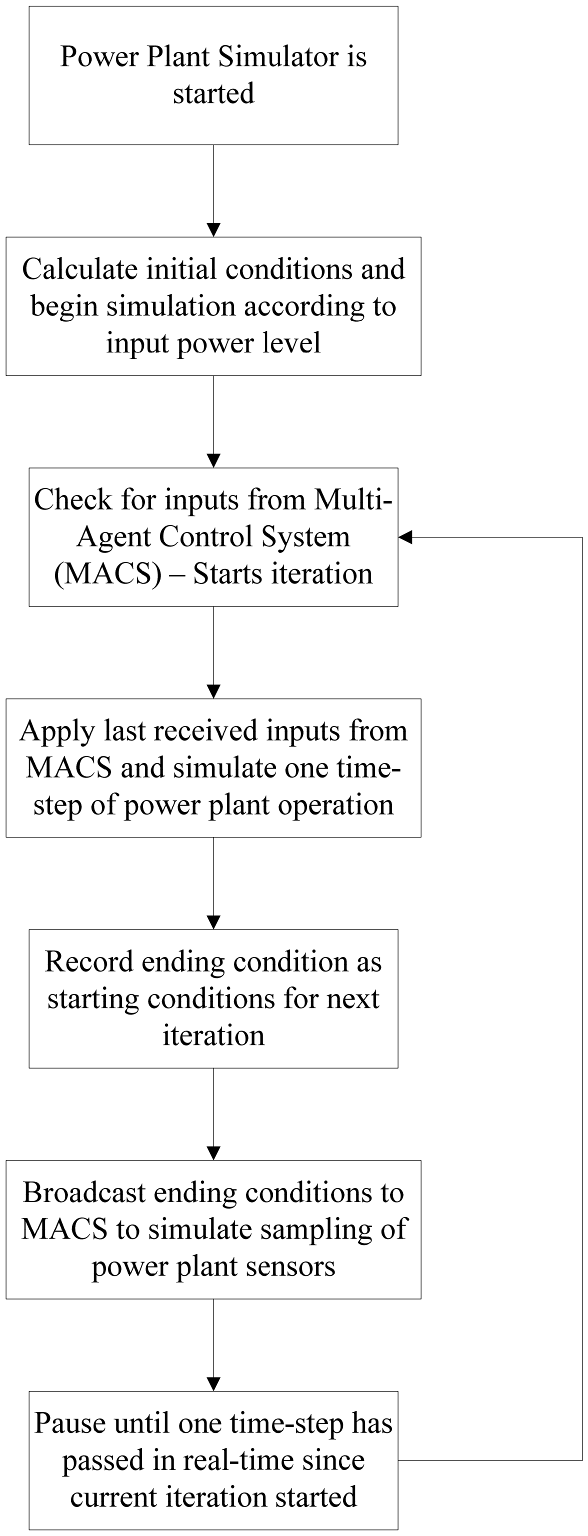

Power Plant Simulator is started
Calculate initial conditions and begin simulation according to input power level
Check for inputs from Multi-Agent Control System (MACS) – Starts iteration
Apply last received inputs from MACS and simulate one time-step of power plant operation
Record ending condition as starting conditions for next iteration
Broadcast ending conditions to MACS to simulate sampling of power plant sensors
Pause until one time-step has passed in real-time since current iteration started

Fig. 4.1. Flow diagram of the basic operation of the Power Plant Simulator.

properly, the computer used to run the software must be able to execute one loop in an amount of time shorter than the time-step. As the PPS is running, the current power plant input and output values are displayed on the GUI. The GUI was created using the Matlab Graphical User Interface Development Environment (GUIDE).

4.1.3 Interface Agent GUI

The Interface agent GUI allows a user to communicate with the MACS through the Interface agent. The Interface agent GUI is mainly used to load ULDC data, stored in Matlab data files, and send that data to the Feedforward agent as the first step in initializing a MACS simulation. Other functionality that has been added for experimentation with the MACS is the ability to manually change agents' tasks, feedback gains in the Feedback agent, and optimization preference values in the Feedforward agent.

The Interface agent and GUI are initialized by the same script in this thesis. When the script is executed, the Interface agent is started and waits for instruction from the GUI. Once an operation is requested, the Interface agent decides how it should be executed, as explained in Section 3.3.1. The Interface agent can also be used to retrieve MAS or power plant data to display for the user in the GUI. Like the PPS GUI, the Interface agent GUI was created using GUIDE.

4.2 Simulation and Testing Results

This section explains how the MACS was tested and discusses the results. The explanation includes a discussion of the methods for testing the individual agents, as well as the entire control system. The results of the MACS tests are discussed to explain how the agents and the control system as a whole were shown to perform as designed.

4.2.1 Feedforward Agent

Since the Feedforward agent functions as the reference governor in the MACS, the operation of the setpoint optimization algorithm and the accuracy of the FFNN model used by the optimization are tested to show that the Feedforward agent can perform as described in Section 2.3. The setpoint optimization is tested in four stages. The first stage considers only one objective function and the subsequent stages add an objective function to show the effect each of them has on the optimization. Each stage was performed at power levels across the full range of operation and, from the results, is shown to perform as designed.

The accuracy of the FFNN is tested by comparing the result of performing setpoint optimizations for power levels across the full range of operation using the equation model and the FFNN model to evaluate the cost of candidate solutions. If the FFNN has successfully been trained to model the equations, there will be little or no error between the setpoints generated using each model for a given power level. Furthermore, the Feedforward agent is shown to perform as designed in later sections by the successful overall performance of the MACS.

4.2.1.1 Testing the setpoint optimization. As previously mentioned, the setpoint optimization procedure uses a hybrid particle swarm optimization (HPSO) algorithm. The number of particles and the maximum number of iterations were chosen such that the variation in the resulting control setpoints and boiler pressure reference level produced by the setpoint optimization was consistently less than 10^{-4} for 100 trials at power levels starting at 10 MW and increasing by 10 MW to 160 MW. The number of particles chosen was 120, where each particle represents a candidate solution evaluated each iteration. The lowest maximum iteration number that met the mentioned criteria using 120 particles was 170. Since the number of iterations directly effects the time it takes to perform an optimization, it is desirable to choose the maximum iteration number as small as possible. The number of particles has very little effect on the time it takes to perform the optimization compared to the maximum iteration number, but it is still desirable to use only as many as needed.

The remaining optimization parameters, listed in Section 2.3.2, were chosen to be as those in [3], where $c_1 = c_2 = 2$, $w_{\min} = 0.3$ and $w_{\max} = 0.8$, and preference values are $\beta_1 = 1$, $\beta_2 = 0.5$, $\beta_3 = 1$, and $\beta_4 = 0$. The operating objectives are, as previously stated, the minimization of the objective functions $J_i(u)$, for $i = 1, 2, 3, 4$, which represent the minimization of load-tracking error, fuel consumption through the fuel valve, u_1, pressure drop across the steam valve, u_2, and pressure drop across the feedwater valve, u_3, respectively. The setpoint optimization method is tested by performing multiple setpoint optimizations and changing which objectives are used in order to observe the effect on the setpoints generated. This is intended to show that using each objective function has the desired effect on the results. The results of this test are shown in four cases for power levels of 10 MW, 60 MW, 110 MW, and 160 MW to provide results representing the full range of operation.

In order to compare the results of setpoint optimization, the optimal values of each of the control inputs, with respect to the operating objectives, are listed in Table 4.1 for the power output levels of interest. These values are taken from the tables in Appendix A, which contain the minimum and maximum stable valve positions for various power output levels over the possible range or operation. According to the objective functions, it is desired that u1 be minimized and u2 and u3 be maximized.

The first case performs the setpoint optimizations using only the load-tracking error, $J_1(u)$, as an objective function. To use only this objective function, the preference values are set to $\beta_1 = 1$, $\beta_2 = 0$, $\beta_3 = 0$, $\beta_4 = 0$, or $\beta = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$. As shown in Table 4.2 under Case 1, the load-tracking errors resulting from the setpoint optimizations performed in this manner are smaller than double precision floating-point format can represent and is shown as zero. Comparing the resulting control setpoints, u_1, u_2, and u_3, to the optimal ones in Table 4.1, it can be seen that these values are not optimized. This is the expected result, since the control values were not used as cost criterion for the setpoint optimizations.

The second case performs setpoint optimizations using the load-tracking error and the minimization of fuel consumption, $J_1(u)$ and $J_2(u)$, as objective functions. To do this, the preference values are set to $\beta = \begin{bmatrix} 1 & 0.5 & 0 & 0 \end{bmatrix}$. As shown in Table 4.2 under Case 2, the load-tracking errors resulting from the setpoint optimizations are again zero. However, this time the fuel valve positions, u_1, match the optimal ones in Table 4.1. This represents the successful minimization of fuel consumption, because minimizing the fuel valve position will result in the least amount of fuel consumption for particular power level. Comparing the resulting control setpoints for u_2 and u_3 will show that, as expected, these values are not optimized. Therefore, these setpoint optimizations were successful according to the operating objectives used.

The third case performs setpoint optimizations using the load-tracking error, the minimization of fuel consumption, and the minimization of energy loss across the steam valve, $J_1(u)$, $J_2(u)$, and $J_3(u)$, as objective functions, setting the preference values to $\beta = [1 \quad 0.5 \quad 1 \quad 0]$. As shown in Table 4.2 under Case 3, the load-tracking errors resulting from the setpoint optimizations are again zero and the fuel valve positions have been minimized. This time the steam valve positions, u_2, were maximized, matching the optimal values shown in Table 4.1. This represents the successful minimization of energy loss across the steam valve, because more energy is lost the more the steam valve is closed due to the increased pressure from the restriction in flow. Therefore, the setpoint optimizations in this case were also successful according to the operating objections used.

The forth case shows there is no significant difference in the setpoint optimization result when the pressure drop across the feedwater valve, $J_4(u)$, is included as a cost function in the setpoint optimization. This objective was included in the Case 4 by setting the preference values to $\beta = [1 \quad 0.5 \quad 1 \quad 1]$, where setting $\beta_4 = 1$ causes $J_4(u)$ to have the most effect on the result of the optimization. Case 4 in Table 4.2 shows that including the minimization of the pressure drop across the feedwater valve as an operating objective only causes a change in results from Case 3 at low power levels. This is caused by the small variability in feedwater valve position, u_3, for a given power level, where the feasible range for the feedwater valve is the largest at very low power levels (see Fig. 2.4c). Since the power plant does not operate at a power level low enough for operating objective $J_4(u)$ to have an effect during normal operation, this operating objective is not used in simulation in this thesis.

4.2.1.2 Testing the accuracy of the FFNN model. The FFNN used to evaluate the performance of the control setpoint optimization predicts the steady-state output levels corresponding to a combination of control input values. The data used to train this ANN

Table 4.1: Optimal values for u_1, u_2, and u_3 according to $J_2(u)$, $J_3(u)$, and $J_4(u)$ for the power demand levels tested.

E_{uld}	$J_2(u) = \mathrm{Min}(u_1)$	$J_3(u) = \mathrm{Max}(u_2)$	$J_4(u) = \mathrm{Max}(u_3)$
10 MW	0.0625	0.4968	0.1400
60 MW	0.2702	1.1816	0.4101
110 MW	0.4871	1.1816	0.7028
160 MW	0.7016	1.1815	0.9806

Table 4.2: Results of control optimization verification tests.

Case 1	Resulting control values and generation error with $\beta = [1\ 0\ 0\ 0]$.			
E_{uld}	$\lvert E\text{-}E_{uld}\rvert$	u_1	u_2	u_3
10 MW	0	0.1226	0.2940	.0897
60 MW	0	0.2857	0.8926	0.4072
110 MW	0	0.5165	0.8851	0.6955
160 MW	0	0.7154	1.0559	0.9731

Case 2	Resulting control values and generation error with $\beta = [1\ 0.5\ 0\ 0]$.			
E_{uld}	$\lvert E\text{-}E_{uld}\rvert$	u_1	u_2	u_3
10 MW	0	0.0625	0.4968	0.0809
60 MW	0	0.2702	1.620	0.4054
110 MW	0	0.4871	1.1477	0.6878
160 MW	0	0.7016	1.1642	0.9696

Case 3	Resulting control values and generation error with $\beta = [1\ 0.5\ 1\ 0]$.			
E_{uld}	$\lvert E\text{-}E_{uld}\rvert$	u_1	u_2	u_3
10 MW	0	0.0625	0.4968	0.0809
60 MW	0	0.2702	1.1816	0.4101
110 MW	0	0.4871	1.1816	0.7028
160 MW	0	0.7016	1.1815	0.9806

Case 4	Resulting control values and generation error with $\beta = [1\ 0.5\ 1\ 1]$.			
E_{uld}	$\lvert E\text{-}E_{uld}\rvert$	u_1	u_2	u_3
10 MW	4.5342E-03	0.0716	0.4968	0.1354
60 MW	0	0.2702	1.1815	0.4101
110 MW	0	0.4871	1.1816	0.7028
160 MW	0	0.7016	1.1815	0.9806

was generated by calculating the control values, u_1, u_2, and u_3, using the equations in (2.4) for combinations of electric power, E, and drum steam pressure, P, where E varies from 10 MW to 180 MW in steps of 1 MW and P varies from the minimum stable pressure for each E to the maximum, shown in Fig. 2.3, in steps of 5 kg/cm^2.

A test was done to determine the number of hidden neurons that would allow a trained FFNN to most accurately model the training data. The test involved using the training data generated from the steady-state equations to train FFNNs initialized with different numbers of hidden neurons. The number of hidden neurons ranged from 7 to 30.

The number seven is chosen as the minimum based on the Hecht-Nielson Theorem, which states that any continuous function $f : I^n \rightarrow R^m$ can be approximated by a feedforward network with $2n + 1$ hidden nodes, where n is the number of inputs and m is the number of outputs [24,25]. The maximum number of nodes was chosen as 30 as a higher number of hidden neurons means longer training and simulation times. For each trial, a FFNN was initialized and trained with the same data for a maximum 50,000 iterations using the Levenberg-Marquardt backpropagation training algorithm. The number of hidden neurons producing the lowest mean-squared error was chosen as the best. Therefore, the number of hidden nodes used is 30.

To test whether or not the FFNN is capable of accurately modeling the equations in (2.4), and therefore able to produce an accurate optimization result when using a FFNN in place of the equations, both the equation model and the FFNN model were implemented in setpoint optimizations performed for power demand levels from 10 MW to 160 MW in increments of 5 MW. Each test performed used the optimization parameters noted in the previous section. In Fig. 4.2, which shows the results of the test optimizations performed, it can be seen that using the FFNN is equivalent to using the equation model, because the control setpoints, u_1, u_2, and u_3, resulting from the use of both models for optimization are approximately equal. This implies that the FFNN was successfully trained to model the equations in (2.4).

To give a quantifiable result, the mean-squared and maximum error of the difference in setpoints generated by the two methods, shown in Fig. 4.2, are given for each control variable in Table 4.3, where u_{eq} is the result of optimization using the equations and u_{nn} is the result of optimization using the neural network. Using the steady-state model equations in (2.5), the difference in steady-state power and pressure caused by changing each control variable by the maximum error is shown in Table 4.4. For

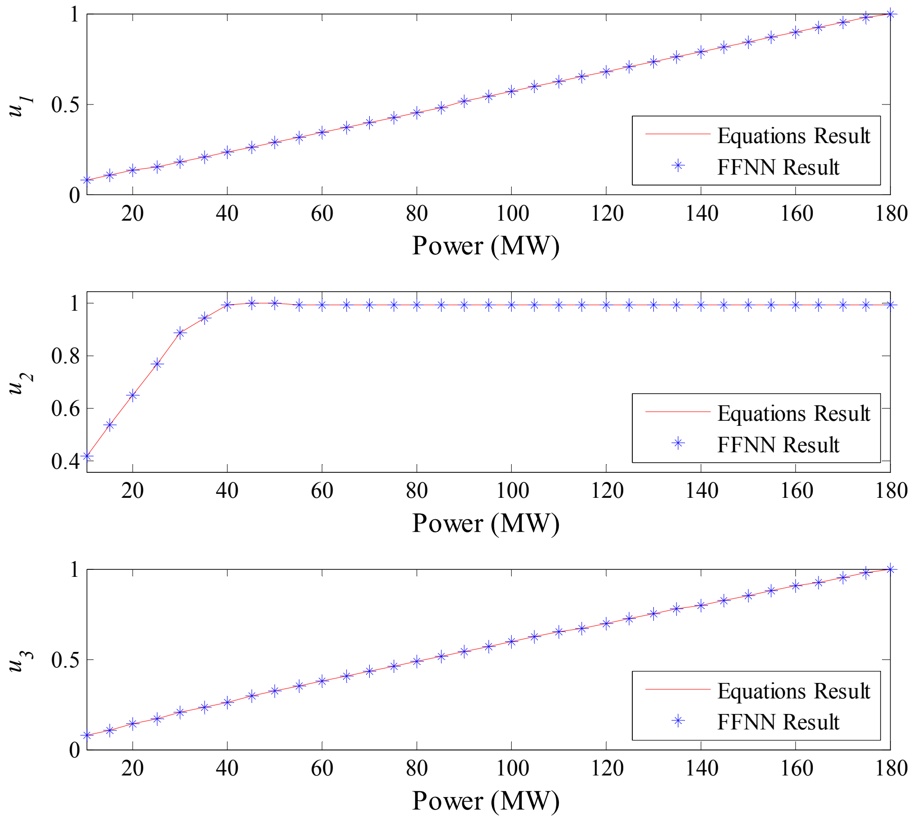


Fig. 4.2. The result of setpoint optimization performed comparing the equation model with the FFNN model.

example, in Table 4.4, the u_1 column shows the difference in steady-state power and pressure caused by changing only the fuel valve position by the maximum difference in the setpoints produced for u_1 using the two methods, shown in Table 4.3. The u_2 and u_3 columns in Table 4.4 show the results for the same changes in u_2 and u_3, respectively. Large values in Table 4.4 would indicate that the FFNN was not trained well and that the Feedforward agent will likely produce setpoints that are not optimized with respect to the objective functions.

Table 4.3: The mean-squared and maximum error of the difference in setpoints generated by the setpoint optimization using the equation and FFNN models.

	u_1	u_2	u_3
MSE(u_{eq}-u_{nn})	2.6013e-16	1.0594e-12	2.3694e-14
max(|u_{eq}-u_{nn} |)	1.1315e-11	4.3260e-6	1.1159e-7

Table 4.4: The difference in power and pressure setpoints caused by changing each control variable by the maximum error shown in Table 4.3.

	u_1	u_2	u_3
E(max|u_{eq}-u_{nn} |) (MW)	0	1.6727e-5	9.1931e-6
P(max|u_{eq}-u_{nn} |) (kg/cm^2)	0	7.0581e-5	5.1246e-5

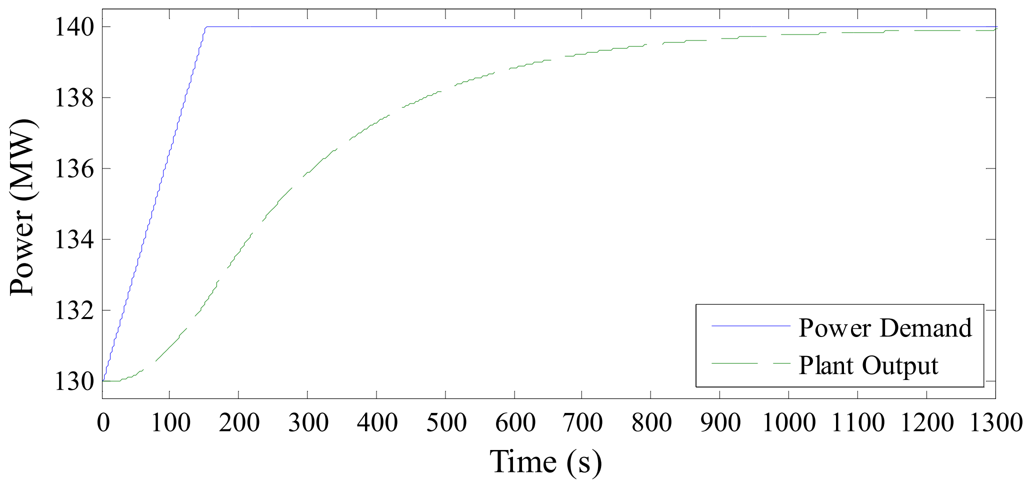
4.2.2 Feedback Agent

To test the performance of the Feedback agent, which was designed to perform the feedback control function in the MACS, the PPS simulator was used to simulate the result of MAS control with and without the Feedback agent. The unit load demand for these simulations is a ramp in power level from 130 MW to 140 MW at a rate of 5%/min, which is the maximum acceptable rate in practice. The Feedforward agent was used in both cases to provide the control setpoints and output reference levels.

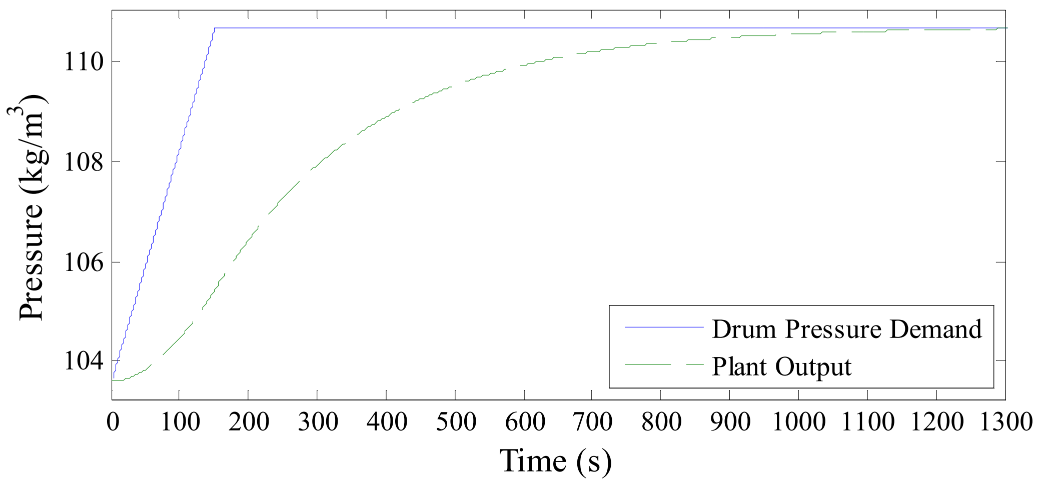
The result of the described simulation without the Feedback agent is shown in Figs. 4.3 and 4.4. In Fig. 4.3a, the power output response to the increase in power level without feedback is very slow, taking approximately 1300 seconds to meet the final setpoint. Fig. 4.3b shows the pressure response to be similarly slow, taking approximately the same time to reach the final setpoint. The water level response, shown in Fig. 4.3c, shows the water level deviation rising to a level of approximately 170 mm above the level setpoint, which would not be acceptable in a real-world scenario. These results show a base case for evaluating the Feedback agent performance, and also that feedback control is necessary for acceptable operation of this plant.

The result of the simulation with the Feedback agent active is shown in Figs. 4.5 and 4.6. The gains used in the Feedback agent for this simulation were the result of an offline gain optimization performed a step input in power level from 130 MW to 131 MW. These gains are shown in Table 4.5, where the columns contain the proportional and integral gain values corresponding to the control loops that control u_1, u_2, and u_3, respectively.

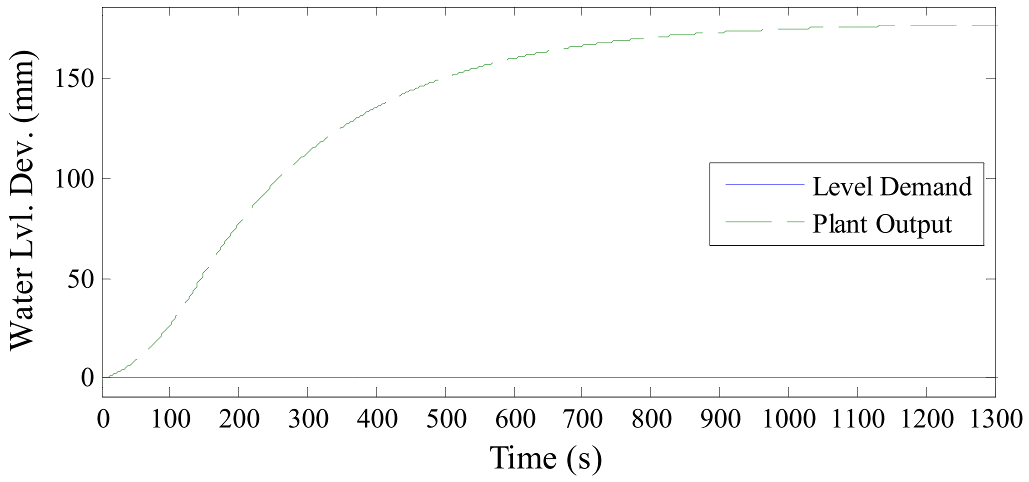
In Fig. 4.5a, the power output response to Feedback agent control matches the demand much more closely than in the case without feedback control, taking less than 200 seconds to make the 10 MW change in power level. The pressure response to Feedback agent control, shown in Fig. 4.5b, is also much better than in the case without feedback control, closely matching the demand and meeting the final setpoint in approximately 200 seconds. The water level response, shown in Fig. 4.5c, is much better as well, never deviating more than 10mm from the setpoint level. These results show that



(a)

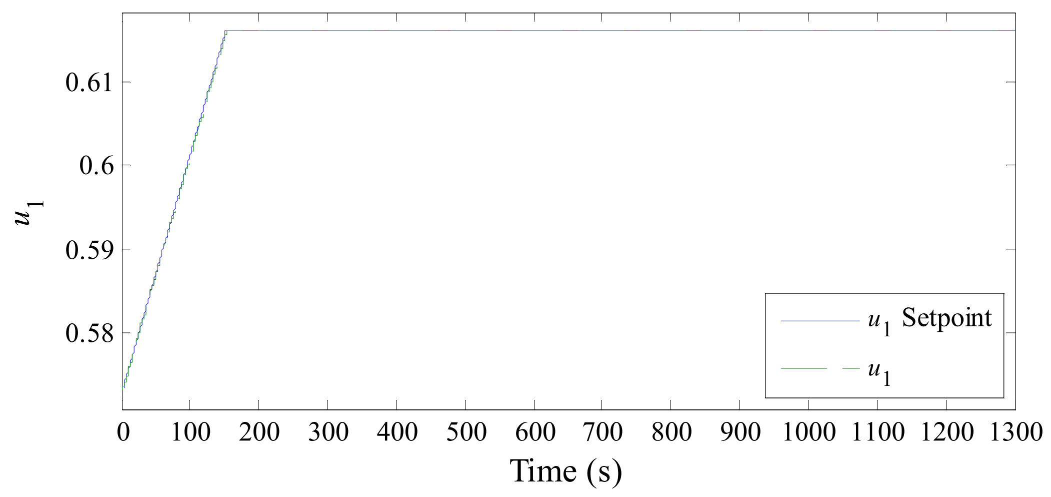


(b)

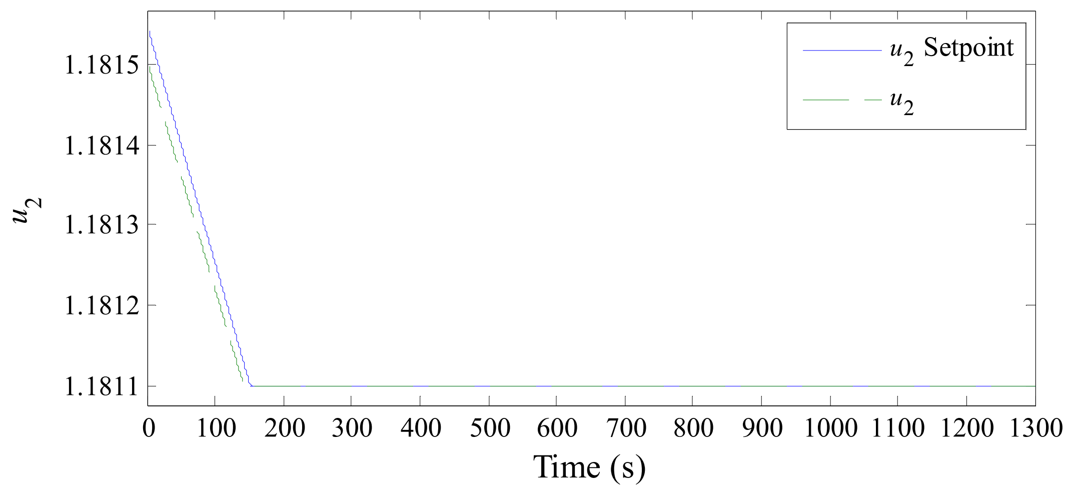


(c)

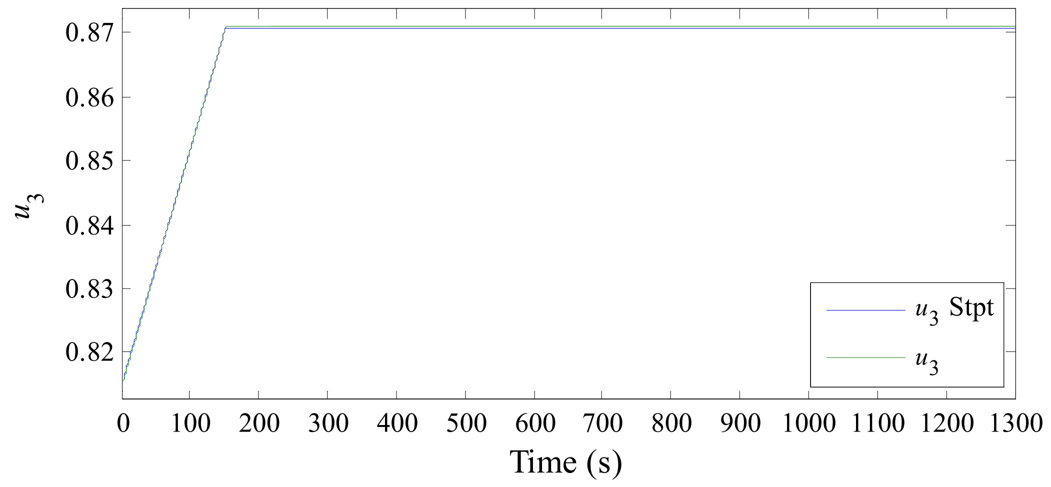
Fig. 4.3. The PPS response to a ramp in power level without feedback control.



(a)

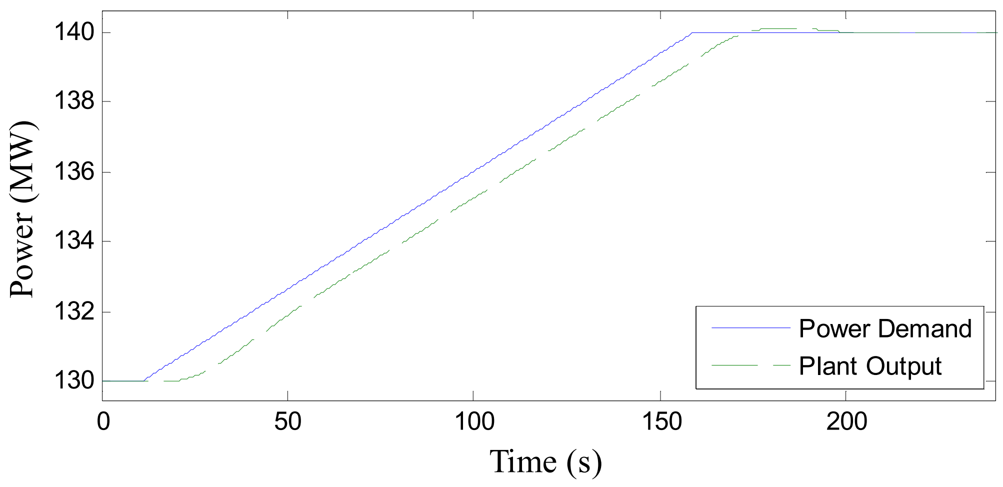


(b)

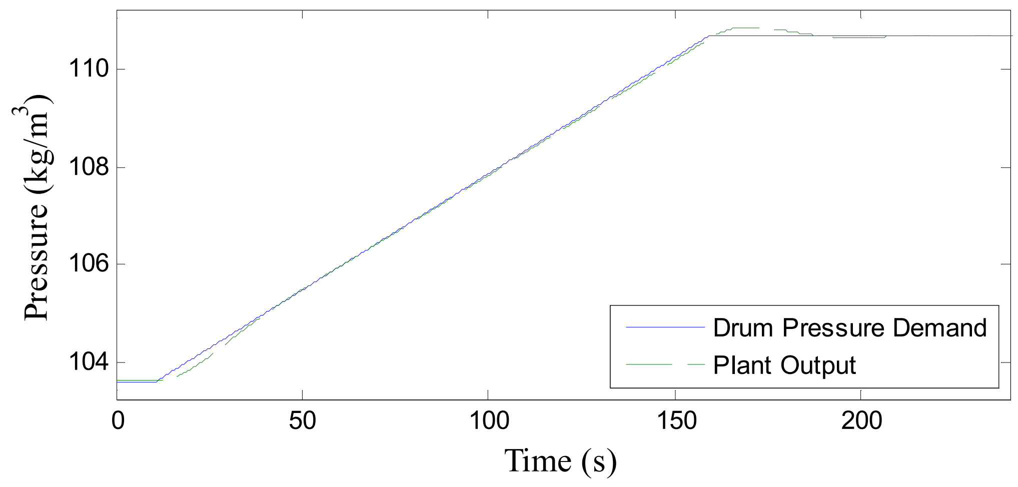


(c)

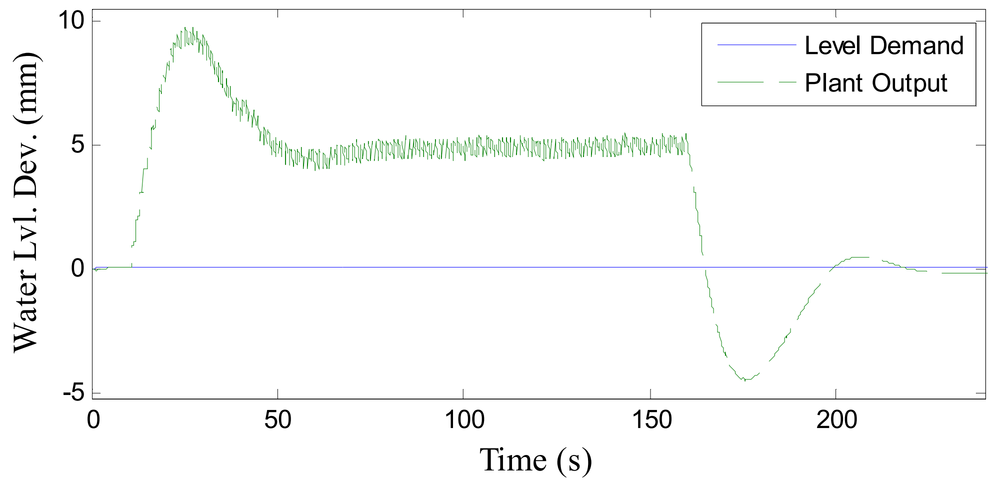
Fig. 4.4. The change in control valve positions corresponding to the ramp in power level in Fig 4.3 without feedback control.



(a)

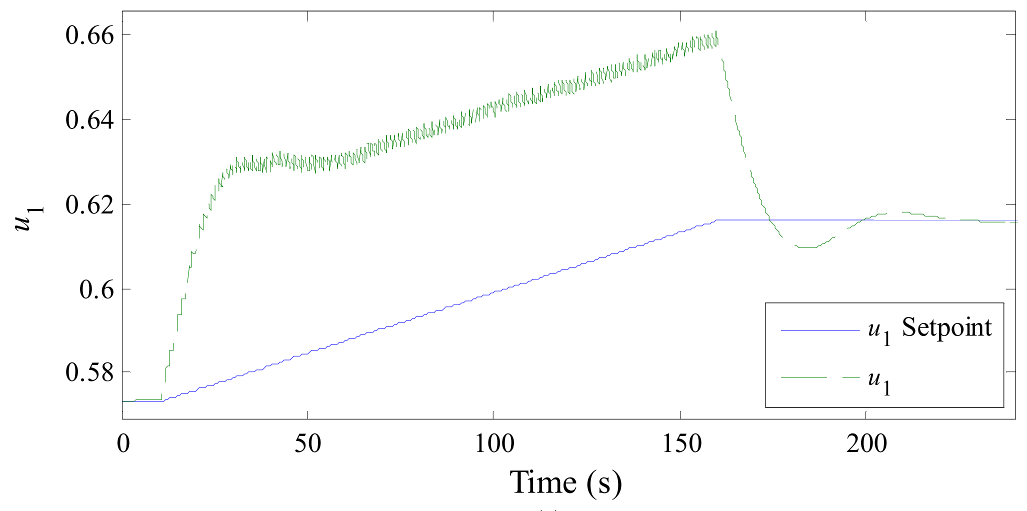


(b)

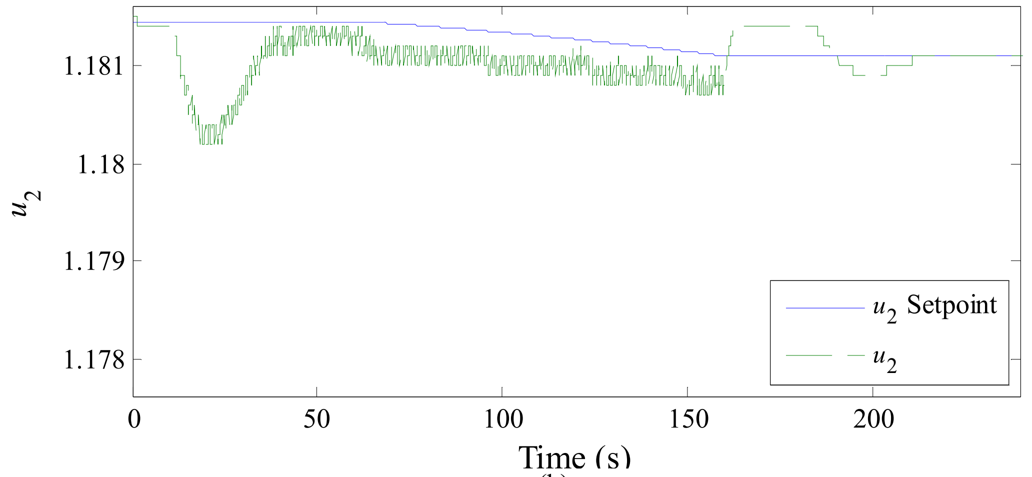


(c)

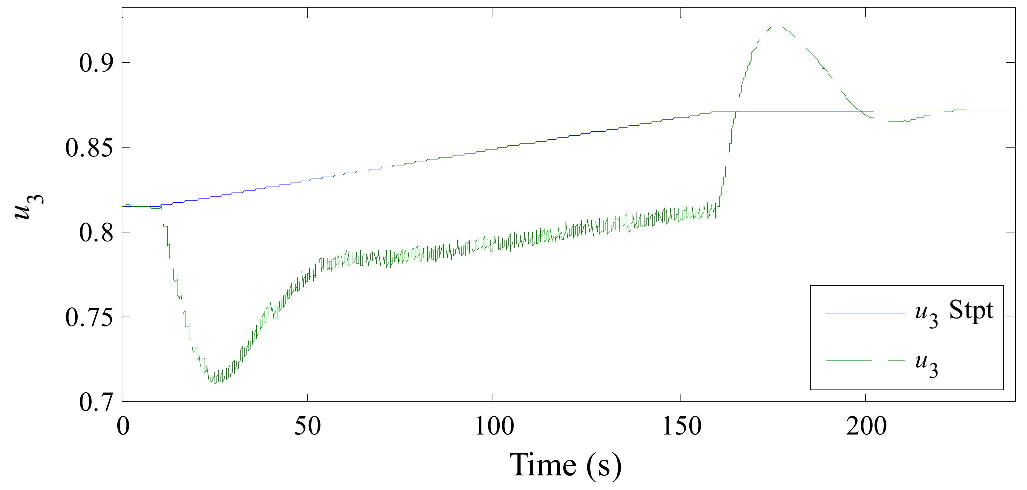
Fig.4.5. The PPS response to a ramp in power level with feedback control from the Feedback agent.



(a)



(b)



(c)

Fig.4.6. The change in control valve positions corresponding to the ramp in power level in Fig 4.5 with feedback control.

the Feedback agent can successfully perform feedback control of the power plant model as part of the MAS designed in this thesis.

Table 4.5: Optimized feedback gain values used in the Feedback agent simulation.

	u_{fb1}	u_{fb2}	u_{fb3}
K_p	0.0550	-0.0055	0.0114
K_i	9.7480e-06	-3.1380e-06	8.2410e-06

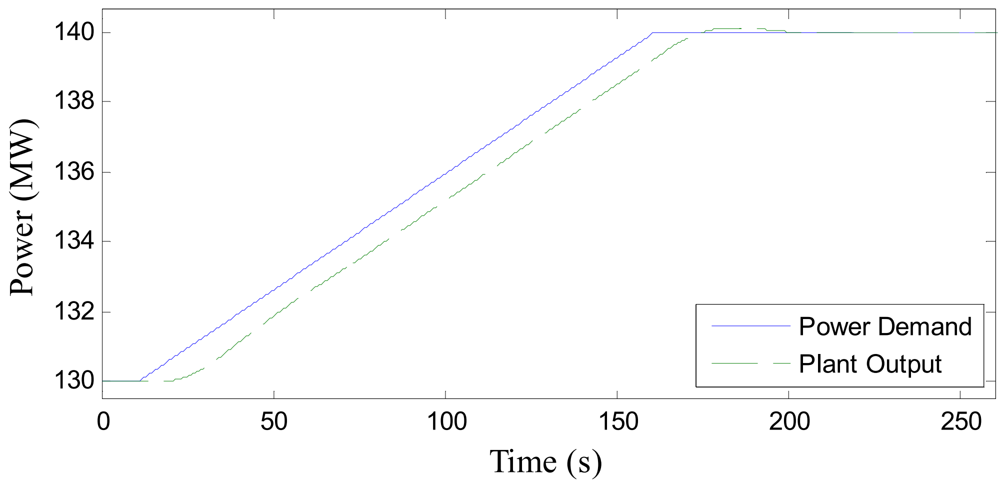
One thing to notice in the results in Fig. 4.5 is the consistent difference in the power demand and the power response during the ramp, appearing to be a time-delay in the response compared to the demand. This is caused by the fact that the demand is changing frequently during the ramp, and control actions are changing to meet the new demands as they are given. This does not give the power plant enough time to catch up to the demand before a new one is given, and the only way it could catch up during a ramp is to generate control actions in anticipation of future demand changes.

However, the pressure level in Fig. 4.5b does catch up to the demand initially, but lags behind toward the end of the ramp. It does so because the feedback controller causes it to overshoot the current demand at the beginning, but settles to lag behind the demand toward the end. The water level in Fig. 4.5c increases in response to the increase in power demand and, similar to the power output response, does not catch up until the demand stops changing.

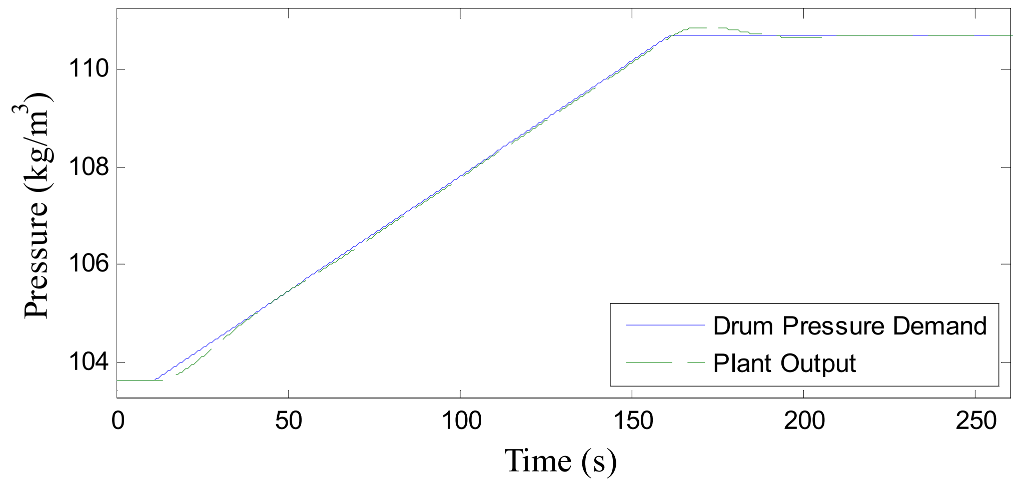
Looking at Fig. 4.5c, there appears to be some oscillation in the water level response. This is caused by the fact that the setpoints are changed once every second and the feedback changes the control variables ten times per second. When the setpoints are changed to represent an increase in power level, the control valves are changed in response to the new setpoints, causing the water level to rise. During the remaining time before a new setpoint is given, the Feedback agent is trying to decrease the water level back to the setpoint. This happens every time the demand is changed until the ramp in power level is complete. This process causes an oscillation in the water level that has a frequency of 1 Hz.

To solve this problem, another simulation was performed for the same ramp in power, but the Feedforward agent was made to generate setpoints ten times faster. The result of this simulation is shown in Fig2. 4.7 and 4.8. The power output and pressure responses, shown in Fig.4.7a and Fig. 4.7b, respectively, very much resemble the corresponding responses in Fig. 4.5, having slightly improved by decreasing the error between the demands and the output levels. The water level response has also improved, showing a slight decrease in the error between the response and the demand. However, the main difference is that there is much less oscillation due to the changes in setpoints. Therefore, a higher frequency in setpoint generation is better in terms of minimizing the water level oscillation, and increases the effectiveness of feedback control in terms of following the changes in setpoints.

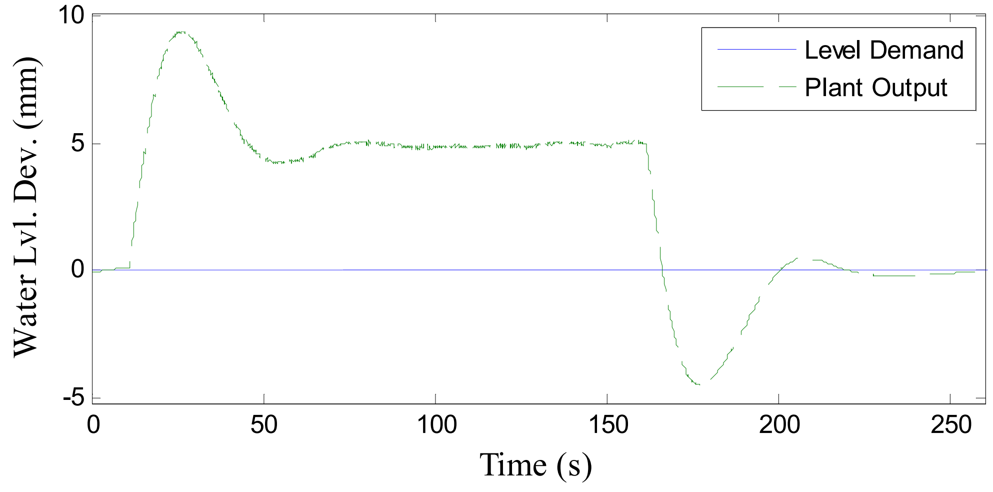
Although the increase in setpoint generation frequency decreased the oscillatory response in the water level to a more acceptable level, the Feedforward agent was not able to generate the setpoints at the new frequency in real time. Instead, the setpoints were generated before starting the simulation and were applied by the Feedforward agent at the appropriate time. The Feedforward agent was designed to generate new setpoints once a second to allow the optimization sufficient time to execute. However, this problem is due to a lack of performance capability of the computers used to perform the



(a)

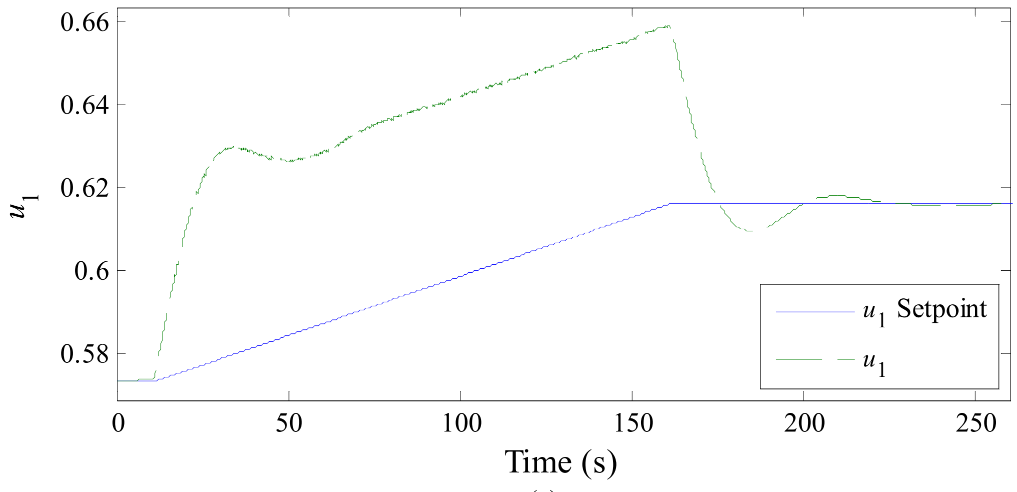


(b)

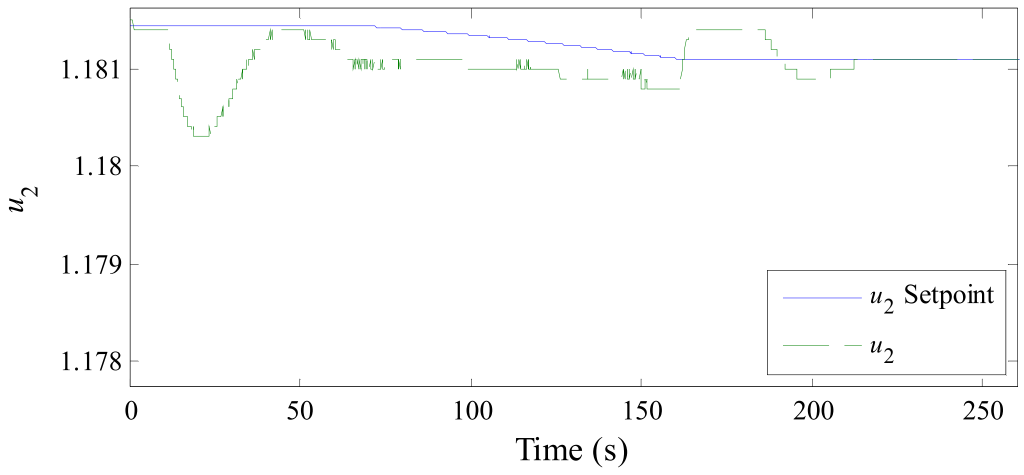


(c)

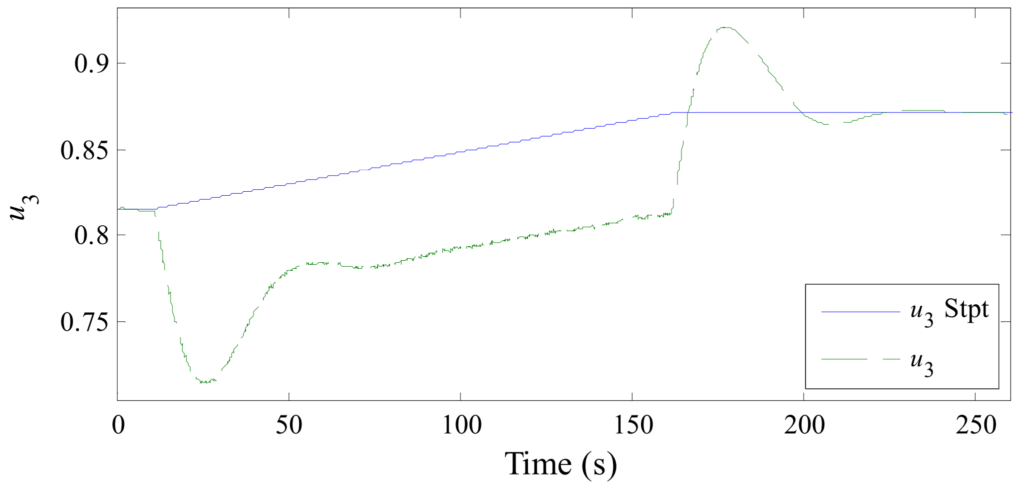
Fig.4.7. The PPS response to a ramp in power level with feedback control implemented by the Feedback agent.



(a)



(b)



(c)

Fig.4.8. The change in control valve positions corresponding to the ramp in power level in Fig 4.7 with feedback control implemented by the Feedback agent.

simulation and may be solved by implementing this control system on higher performance equipment.

4.2.3 Gain Optimizer Agent

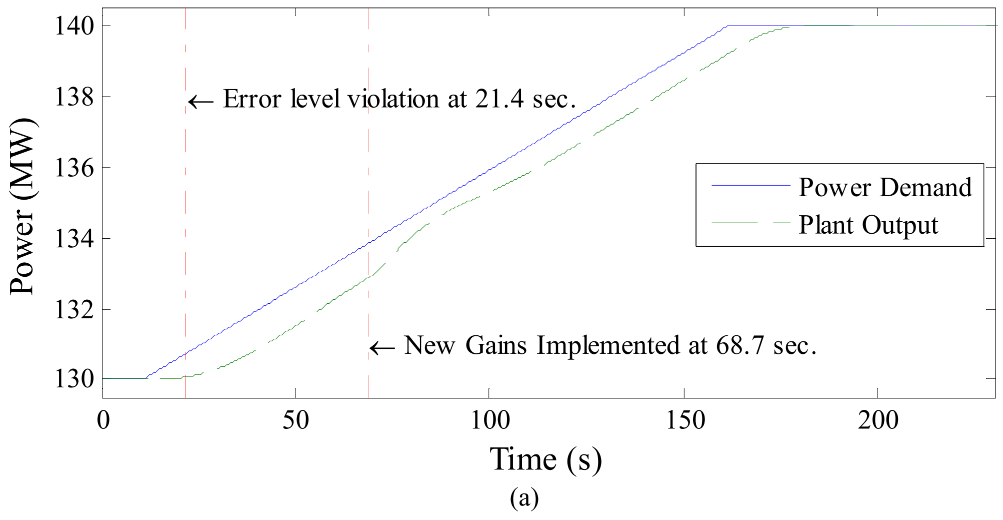
To test the performance of the Gain Optimizer agent, a MAS simulation was performed using gains in the Feedback agent that would allow the error between the power plant output and the demand to be greater than the error thresholds that are used to trigger the gain optimization in the Gain Optimizer agent. Once the Gain Optimizer agent is triggered, it has been programmed to provide new feedback gains in 45 seconds or less so that they are implemented in 50 seconds or less.

The simulation is performed for a ramp in power level from 130 MW to 140 MW, changing at a rate of 5%/min. The HPSO algorithm uses 5 particles, which is significantly less than the control optimization, because of the time it takes to simulate each particle, or candidate gain set. The increased time in evaluating candidate solutions is caused by the need to simulate the dynamic response of every particle to obtain a 25-second simulation of the control response caused by each candidate gain set. Since one of the candidate solutions is initialized to the PI gains used in the Feedback agent when the optimization begins, the resulting gains cannot be worse than the gains currently used. This is due to the fact that if a better solution is not found, the currently used gains will perform the best and will be reused.

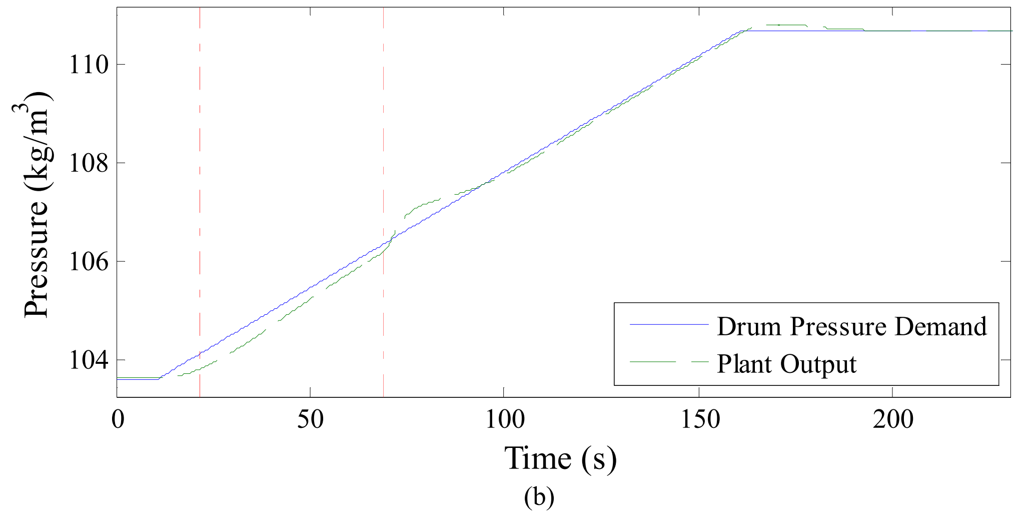
The other optimization parameters, outlined in Section 2.3.2, are $c_1 = c_2 = 2$, $w_{min} = 0.5$ and $w_{max} = 1.0$, and preference values are $\beta_1 = 1$, $\beta_2 = 0.25$, and $\beta_3 = 0.5$. The preference values, β_i, correspond to the cost functions, J_i for $i = 1,2,3$, as mentioned in (2.11), and represent the sum-squared error of the difference between the power, pressure, and water level deviation demand and the simulation output, respectively. The error level thresholds which activate the gain optimization procedure are ±1 MW, ±2 kg/cm^2, and ±10 mm for power, drum pressure, and water level deviation, respectively. The results of the simulation are shown in Figs. 4.9 and 4.10.

Fig. 4.9 shows that an error threshold was exceeded at 21.4 seconds when the water level output rose to above 10 mm in deviation from the setpoint. This violation caused the Gain Optimizer to begin optimizing new feedback gains. The new gains were sent to the Feedback agent and implemented at 68.7 seconds in the simulation, which was 47.3 seconds from the time the violation occurred. This means that the Gain Optimizer agent used the 45 seconds allotted for the gain optimization, and it took less than the remaining 5 seconds for necessary communication and data transfer to take place.

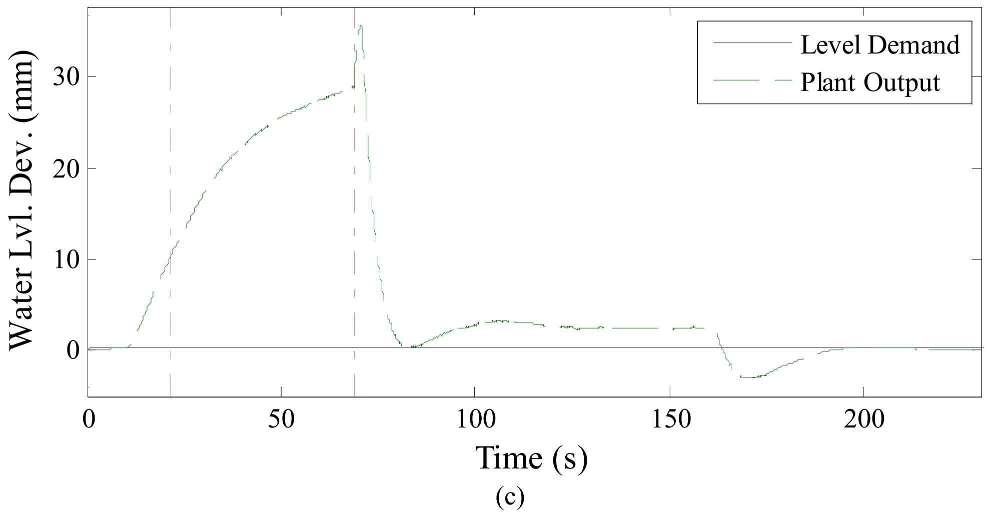
The power output and pressure responses, shown in Figs. 4.9a and 4.9b, do not show much change from the implementation of the new feedback gains, showing only a brief fluctuation in output at the time the change occurred. The water level output, shown in Fig. 4.9c, climbs to almost 30 mm above setpoint before the optimized gains are implemented, at which point the water level spikes quickly to approximately 35 mm and falls sharply to below 5 mm above setpoint. Once the new gains are implemented, the water level remains at a more acceptable level between ±5 mm. This simulation shows that the Gain Optimizer agent can be used to optimize gains, when needed, and implement them in a timely manner. Therefore, the Gain Optimizer agent is shown to perform as designed. The initial PI gains, optimized PI gains, and the bounds set for the gains in the optimization procedure are shown in Tables 4.6-4.8. The position of the control valves during the simulation are shown in Fig. 4.10.



(a)

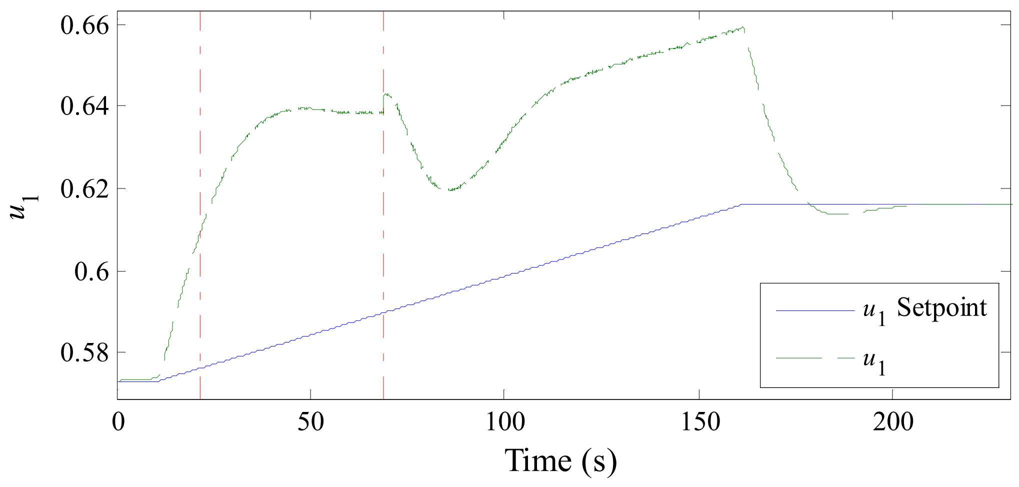


(b)

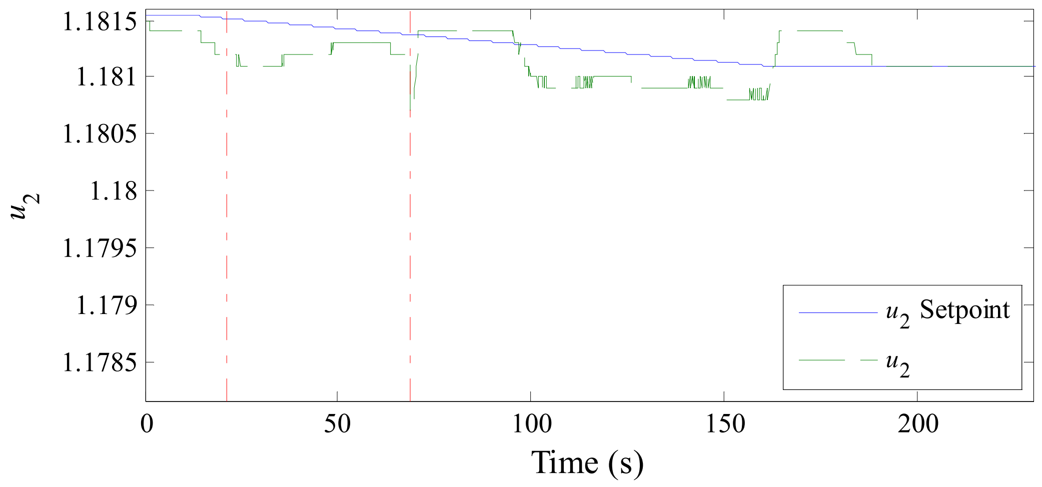


(c)

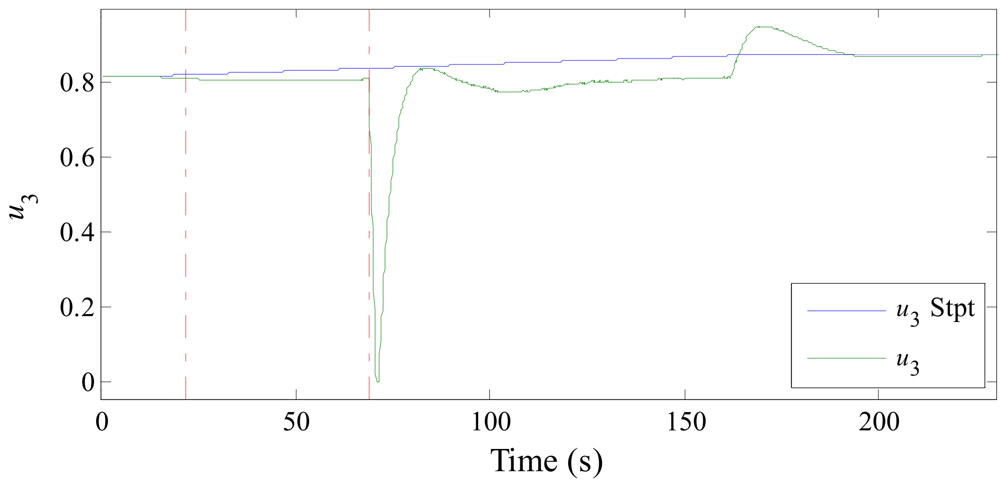
Fig.4.9. The PPS response to a ramp in power level with feedback control implemented by the Feedback agent before and after replacing the PI gains with optimized ones.



(a)



(b)



(c)

Fig.4.10. The change in control valve positions corresponding to the ramp in power level in Fig 4.9 with feedback control implemented by the Feedback agent before and after replacing the PI gains with optimized ones.

Table 4.6: Initial feedback gain values.

	u_{fb1}	u_{fb2}	u_{fb3}
K_p	0.0500	-0.0010	0.0010
K_i	0.0000	0.0000	0.0000

Table 4.7: Optimized feedback gain values.

	u_{fb1}	u_{fb2}	u_{fb3}
K_p	0.0539	-0.0044	0.0240
K_i	4.0997e-07	-4.4096e-07	4.6715e-08

Table 4.8: Constraints on search space for finding optimal feedback gains.

	u_{fb1}	u_{fb2}	u_{fb3}
K_{pmin}	0.0000	-0.0100	0.0000
K_{pmax}	0.1000	0.0000	0.0750
K_{imin}	0.0000	-1.0000e-06	0.0000
K_{imax}	1.0000e-06	0.0000	1.0000e-06

4.2.4 Neural Network Agent

This section explains how the layer-recurrent neural network (LRNN) used in the Gain Optimizer agent was trained, how it was shown to perform successfully in modeling the 160 MW power plant unit, and how the Neural Network agent was shown to perform its intended function.

4.2.4.1 LRNN Training. The training of the LRNN used in the Gain Optimizer agent to evaluate the performance of candidate gain sets it trained in three stages. These stages are global training, local training, and online training. Global training is performed offline using continuous time data generated from the dynamic power plant model equations, (2.1) and (2.2), that represents the full range of stable power plant operation, defined by the power-pressure window in Fig. 2.3.

This data was obtained by dividing the power-pressure window into 9 regions, as shown in Fig. 4.11, and randomly choosing a number of power-pressure points in each region using a uniform distribution. The power-pressure points are then used to calculate corresponding control values using the steady-state equations in (2.4) and (2.5). The control values are then used to form an input vector for simulating dynamic power plant output to be used for training, where the simulation of the control points allows a certain amount of time at each point.

The control points generated from Region 1 are simulated first, progressing through the Regions in order to Region 9. This keeps the pressure from varying too much and causing the simulation to go unstable, which would ruin the collected data for training. The data is generated without feedback control, except for a feedback control

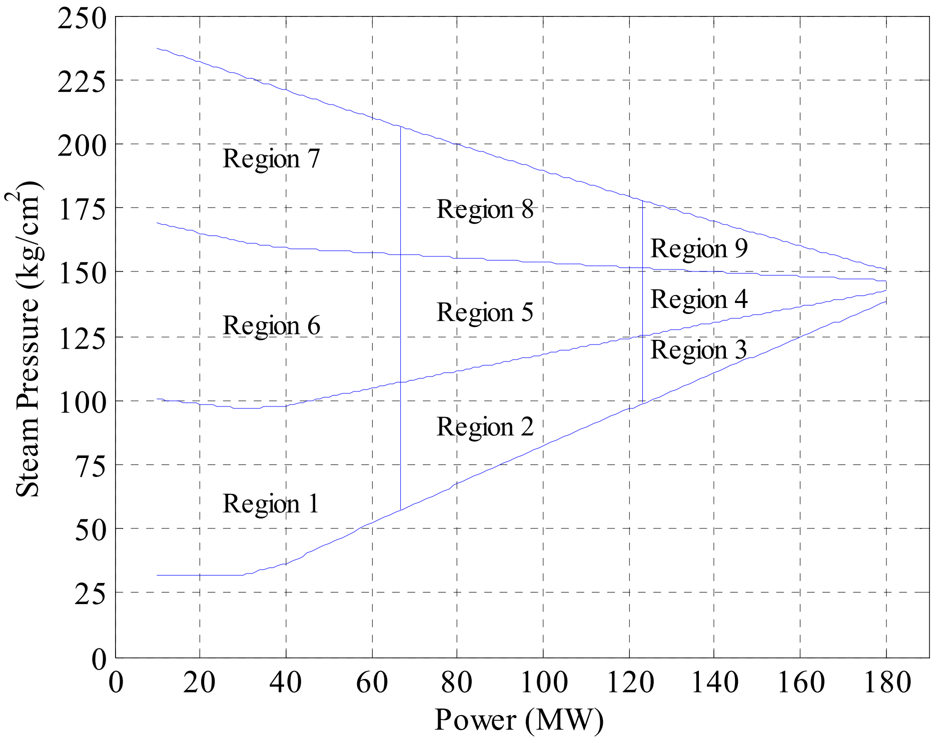


Figure 4.11. Divisions of the power-pressure window into 9 training data regions.

loop used to keep the water level from becoming unstable. Once the data is generated, it can be used to train the LRNN using the Levenberg-Marquardt training algorithm. The process of generating data and training the LRNN is done twice for global training.

Next, local training is performed offline. The method used to generate data for global training is also to generate data for local training, except data is generated in only one region. The region in which data is generated for local training depends on what power level the MACS will begin controlling the FFPU. For example, if for a given simulation the starting power level is 80 MW and the pressure setpoint from the Feedforward agent is 67.3 kg/cm^2, data from Region 2 would be used for local training. The process of generating local training data and training the LRNN is done four times, being careful not to train the local region so much that the general accuracy is lost.

The LRNN is now ready for online implementation in the Neural Network agent. Once the MACS is online, the Neural Network agent will begin collecting power plant input-output data in real-time, incrementally training the LRNN with that data. The online training will keep the LRNN accurate at simulating the power plant dynamics at the operating level of the power plant, which is what is needed for gain optimization

A test was done to determine the number of hidden neurons that would allow a trained LRNN to most accurately model the training data. The test involved generating one set of global data and using it to train LRNNs initialized with different numbers of hidden neurons. The number of hidden neurons ranged from 7 to 30.

The number 7 was chosen as the minimum since it is the minimum necessary to model any continuous function using a FFNN for a system with 3 inputs [24,25], where the LRNN is a modified FFNN. The number 30 was chosen as a maximum since a higher number of hidden neurons means longer training and simulation times. Three trials were performed for each number of hidden neurons. For each trial, a LRNN was initialized and trained with the same data for 1000 iterations using the Levenberg-Marquardt training algorithm. Then, the resulting mean-squared errors were averaged, and the number of hidden neurons producing the lowest average mean-squared error was chosen as the best. Therefore, the number of hidden neurons chosen for the LRNN is 25.

4.2.4.2 LRNN and Neural Network Performance. The ability of the LRNN to model the power plant dynamics was tested in Section 4.2.3 when it was used in the Gain Optimizer to evaluate the performance of feedback gain sets when the Gain Optimizer was called on to optimize new feedback gains. Since the gain optimization was successful, the LRNN was shown to accurately model the behavior of the 160 MW power plant. Furthermore, the Neural Network agent was shown in that simulation to perform successfully, as the LRNN used in the optimization had been trained online by the Neural Network agent when it was used by the Gain Optimizer agent.

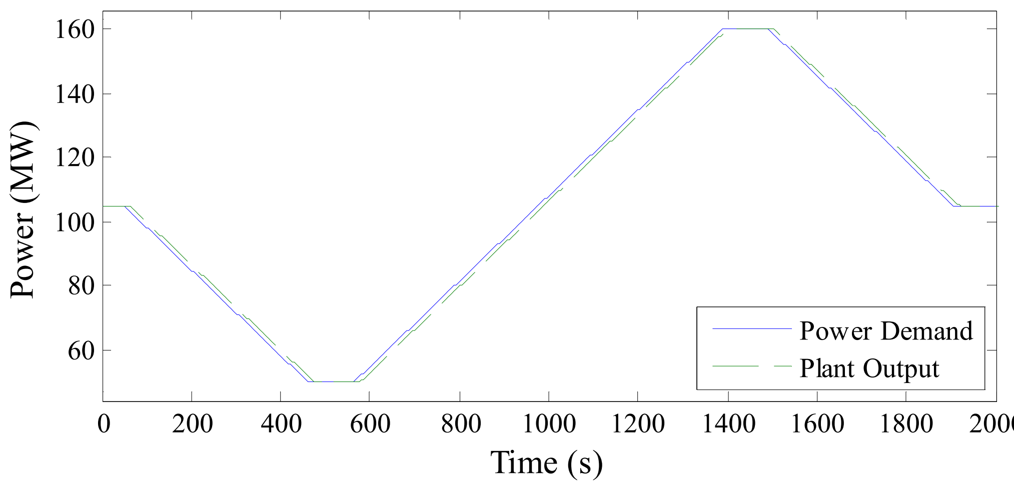
4.2.5 Overall Multi-Agent Control System

To demonstrate the successful overall function of the MACS, two simulations were performed using all of the MACS functionality described in this thesis. The first simulation tests the MACS's ability to effectively control the power plant model during wide-range operation. The second simulation tests the MACS's ability to successfully control the power plant during a real-world type operation.

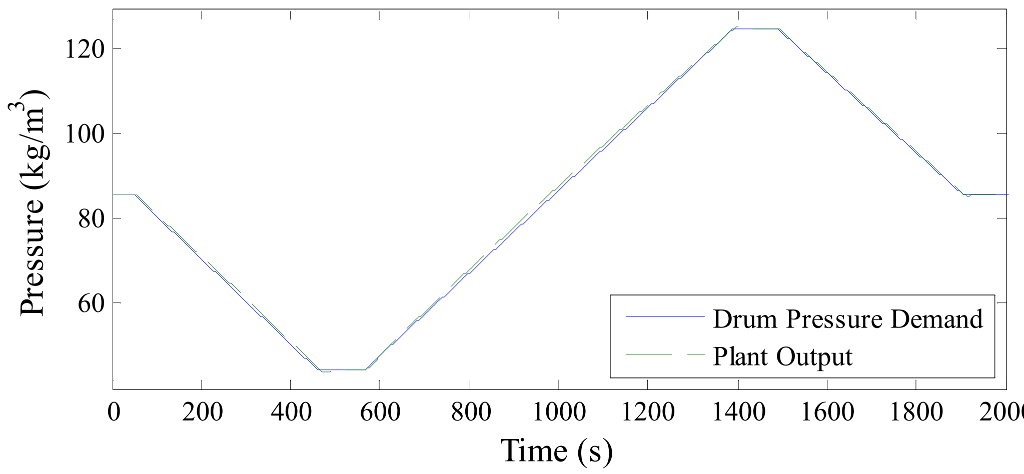
4.2.5.1 MACS Simulation Using a Wide-Range ULDC. To test the ability of the MACS to control the PPS for wide-range operation, a simulation was run using a ULDC that changes between 50 MW and 160 MW by ramping the power level at the maximum allowable of 5%/min, shown in Fig. 4.12a. The initial gain values used for feedback control were those in Table 4.5. Setpoints were dispatched from the Feedforward agent at a rate of 10 Hz to minimize oscillation in the water level, as was discussed in Section 4.2.2. Since the computers used for this simulation could not generate setpoints that quickly in real-time, the setpoints were generated beforehand. The thresholds values for initiating a gain optimization were ±1 MW, ±2 kg/cm^2, and ±10 mm for power, pressure and water level deviation, respectively. The results of the simulation are shown in Figs. 4.12 and 4.13.

The power output and pressure responses, shown in Figs. 4.12a and 4.12b, follow the demand power and pressure very closely, but lag slightly behind the demand levels during ramp changes in power level for the same reason explained in Section 4.2.2. The water level deviation, shown in Fig. 4.12c, stays within ±10 mm except for the times when the ULDC begins or ends a ramp. During these times, the water level briefly spikes and quickly returns to within ±10 mm. This simulation shows that the MACS can be used successfully to control the PPS during wide-range operation. Therefore, the MACS is shown to perform as designed under these conditions.

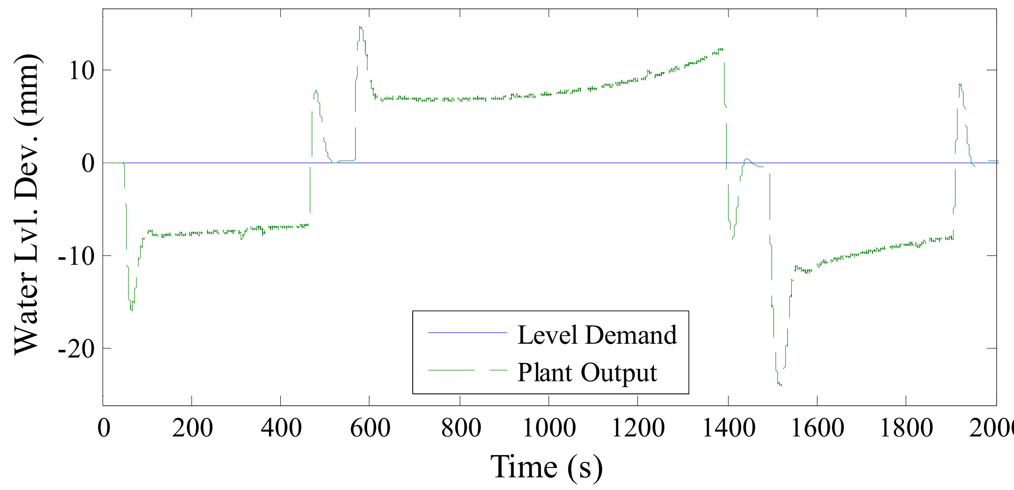
Since the spikes in the water level would have unnecessarily caused the Gain Optimizer agent to trigger, the Feedback agent was programmed not to initiate a gain optimization unless the output levels for power, pressure or water level deviation remained above their respective threshold value for more than 3 seconds. This way, the gain optimization would not trigger due to a false alarm. The delay time for when to trigger a gain optimization once an output level threshold is exceeded is a design



(a)

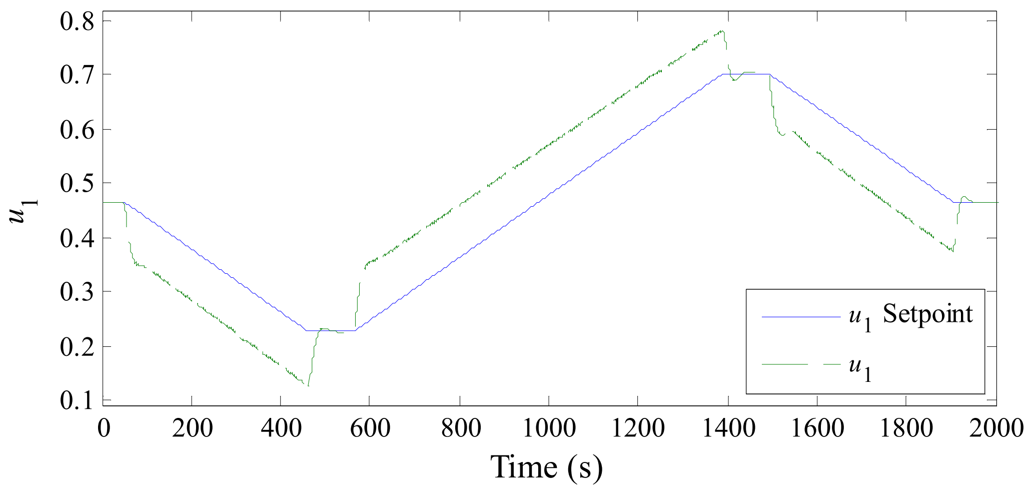


(b)

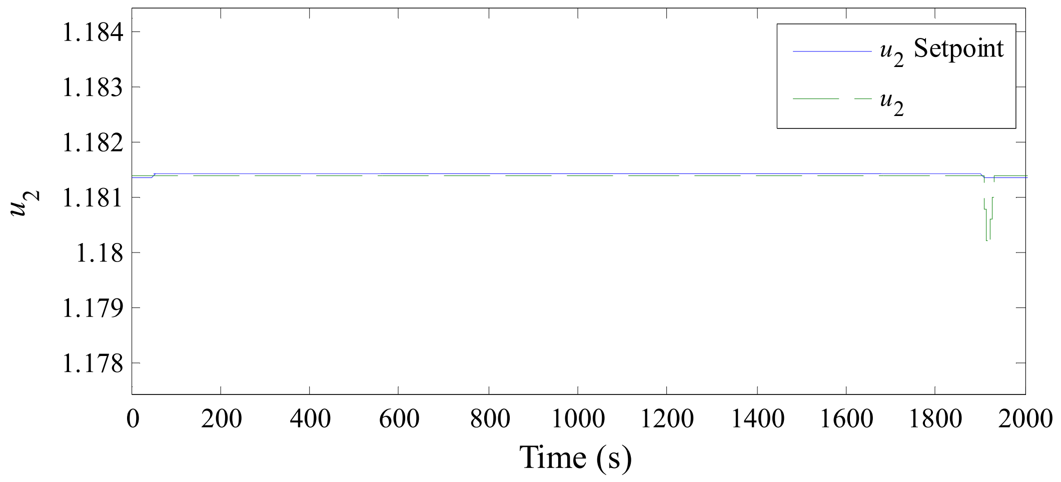


(c)

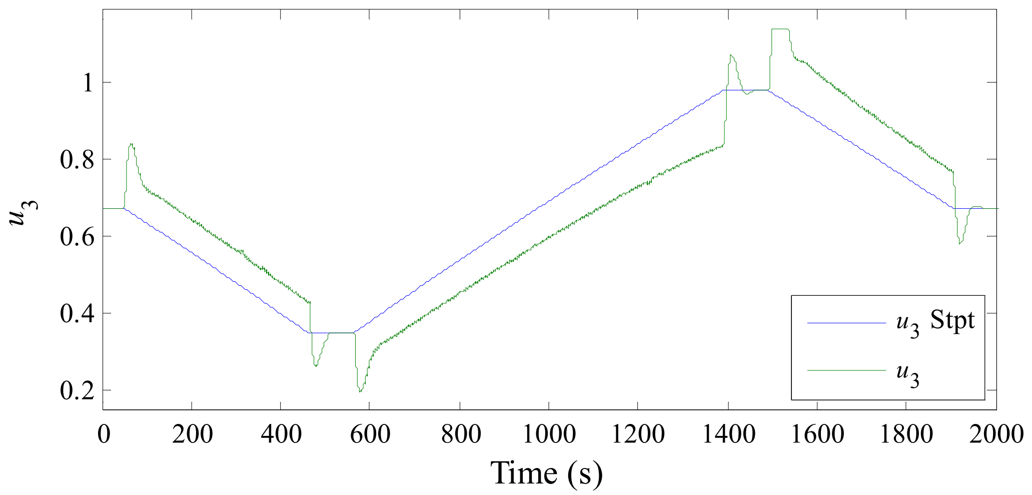
Fig.4.12. The PPS response to a wide-range load cycle under the control of the MACS.



(a)



(b)



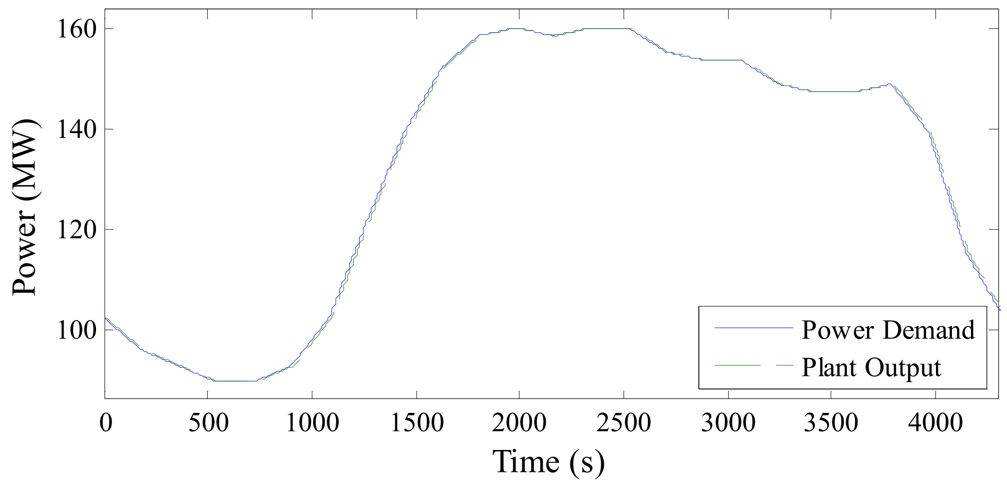
(c)

Fig.4.13. The change in control valve positions corresponding to the wide-range load cycle in Fig 4.9 under the control of the MACS.

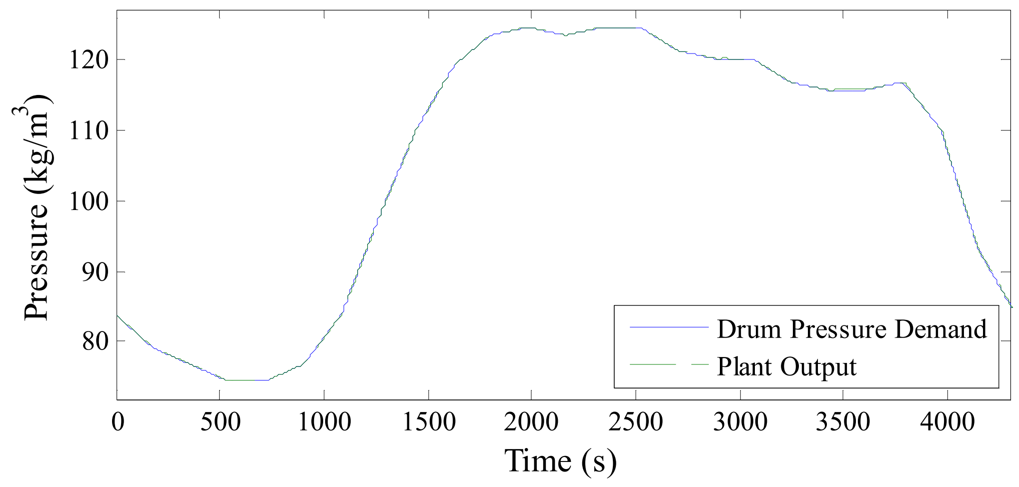
parameter dependant on the system to be controlled. The delay of 3 seconds used here was chosen arbitrarily to avoid unnecessary gain optimization. Optimizing the delay value was not a focus in this thesis.

4.2.5.2 MACS Simulation Using a Realistic ULDC. To test the ability of the MACS to control the PPS under a more realistic scenario, the load cycle for a typical summer weekday is used as the ULDC, where the load data is defined by the IEEE RTS-96 data given in [x6]. The IEEE RTS-96 data defines typical load cycles for weekdays and weekend days for each of the four seasons in terms of a percentage of the maximum generation capacity. In this case, the maximum generation capacity is 160 MW. Similar to the simulation in the previous section, the initial gain values used for feedback control were those in Table 4.5, pregenerated setpoints were dispatched from the Feedforward agent at a rate of 10 Hz and the thresholds values for initiating a gain optimization were ±1 MW, ±2 kg/cm^2, and ±10 mm for power, pressure and water level deviation, respectively. The results of the simulation are shown in Figs. 4.14 and 4.15.

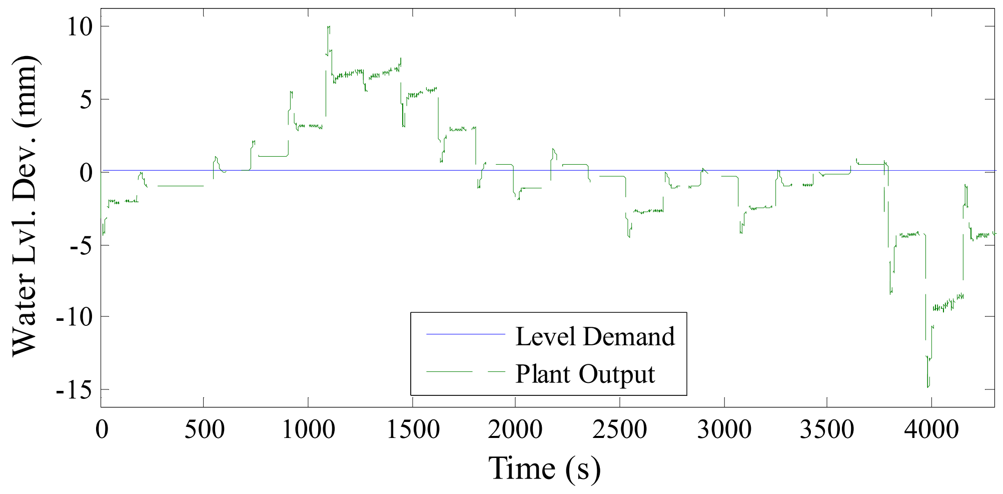
Similar to the simulation in the previous section, the power output and pressure responses, shown in Figs. 4.14a and 4.14b, closely follow the power and pressure demands, only slightly lagging behind, and the water level deviation, shown in Fig. 4.14c, stays within ±10 mm except for sharp spikes that represent overshoot due to a change in the rate at which the power demand level is changing. A 3-second delay is also used here to avoid unnecessary gain optimization. This simulation shows that the MACS can be used successfully to control the PPS for a realistic load cycle. Therefore, the MACS is shown to perform as designed under these conditions.



(a)

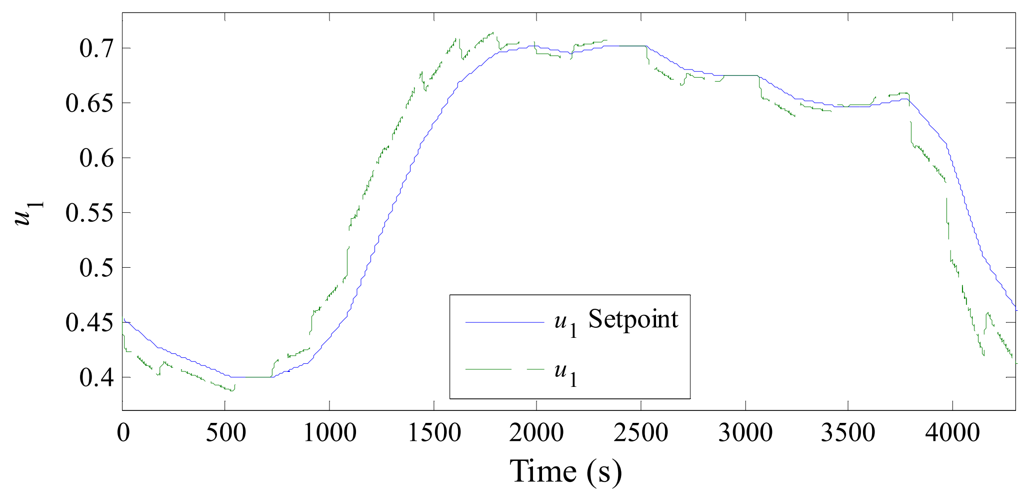


(b)

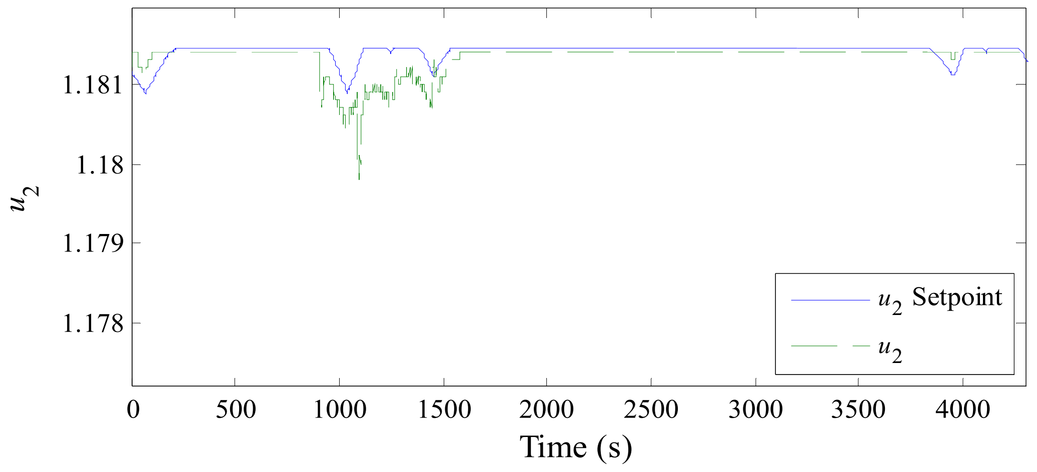


(c)

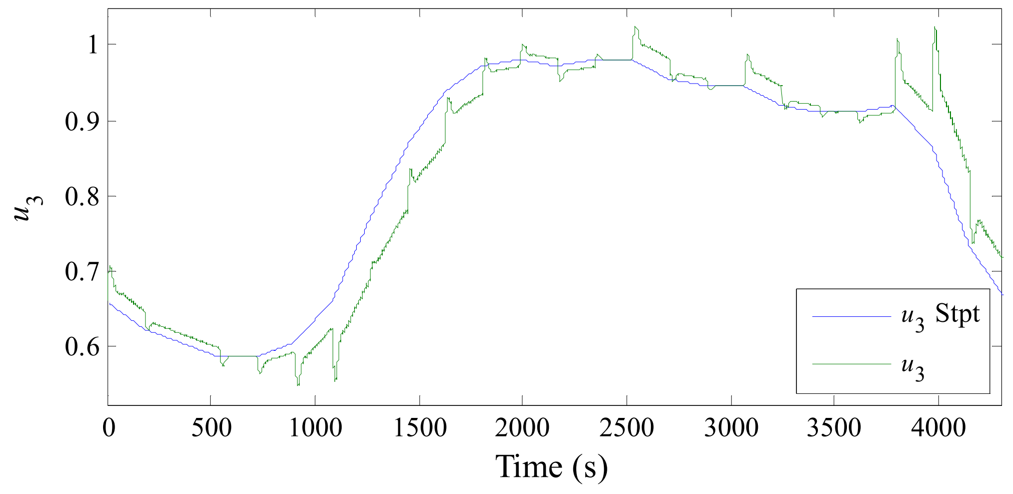
Fig.4.14. The PPS response to a load cycle representing a typical summer weekday under the control of the MACS.



(a)



(b)



(c)

Fig.4.15. The change in control valve positions corresponding to the wide-range load cycle in Fig 4.9 under the control of the MACS.

CHAPTER FIVE

Conclusions

This chapter summarizes the research presented in this thesis and draws conclusions regarding the results from the tests performed in the previous chapter. Also, potential research that can be done to extend the research presented here is discussed.

5.1 Conclusions

The goal of this thesis was to design, implement, and test a multi-agent system (MAS) intended for decentralized optimized multi-objective control of a fossil fuel power unit (FFPU). This control methodology is intended to provide optimized control of an FFPU while allowing the customization of operating goals as needed to conform to changing market situations, such as changing regulations, the cost of fuel, and load demands on the unit. To do this, a MAS was developed consisting of agents that have been designed to perform specific tasks, which in coordination, achieve the desired control. This MAS was designed for a smaller, simpler FFPU model in order to focus on developing the MAS without the added difficulty of accounting for the complexity of a larger power plant model.

The agents developed to implement the control system are the Feedforward agent, Feedback agent, Gain Optimizer agent, Neural Network agent, Interface agent, Delegation agent, Database agent, Monitoring agent and Free agent. These agents were developed and individually tested by performing various experiments, the results of which were analyzed for agent performance. The agents were also implemented in coordination as an MAS and were tested by using the MAS to control a 160 MW FFPU model and analyzing the results.

The Feedforward agent was developed to implement an optimal reference governor that provides customizable coordinated control (CC) by optimizing control setpoints according to prioritized operating objectives that can be changed online. This agent uses a feedforward artificial neural network (ANN) model to evaluate the performance of the optimal setpoints in the hybrid particle swarm (HPSO) optimization method used to determine the optimal references. Each of the operating objective functions was tested to confirm that they had the desired effect on the setpoint optimization results. Because these tests were successful, it was shown that the setpoint optimization can be successfully used to optimize control setpoints according to programmed operating objectives, thereby enabling customizable CC of the power plant unit. Also, the ANN model is shown to accurately represent the steady-state equations by producing the same result when used in the optimization.

The Feedback agent was developed to implement feedback control, which is a necessary part of any control system. This agent implements feedback control of the FFPU while monitoring the effectiveness of that control. If for any reason the gains used in the feedback controller are not performing well, the Feedback agent has been designed to automatically request new, optimized ones. To show that the Feedback agent could be effectively used to implement feedback control in the multi-agent control system (MACS), the Power Plant Simulator (PPS) was used to simulate the response of the power plant model to control by the MACS with and without the Feedback agent. The vast improvement of the power plant response when using the Feedback agent, as opposed to no feedback control, indicates that the Feedback agent can effectively provide feedback control of an FFPU.

The Gain Optimizer agent was developed to optimally tune the feedback gains in the case where the ones being used in the Feedback agent are no longer sufficient for stable efficient control of the FFPU. Similar to the Feedforward agent, this agent uses a dynamic ANN to evaluate the performance of the HPSO algorithm used to produce optimized gains for feedback control. To show that the Gain Optimizer agent can produce improved gains when needed, a simulation was run in which the MAS was used to control the PPS, but the Feedback agent was intentionally initialized with gains that would cause the power plant model output to exceed one or more of the thresholds that trigger a gain optimization. When the first threshold was exceeded, the Feedback agent successfully detected the violation and notified the Gain Optimizer agent. At that point, the Gain Optimizer performed the gain optimization within the specified time and the new gains were sent to the Feedback agent and implemented. Once the new gains were applied, the Feedback agent was able to return all output levels below the threshold values. This success showed that the Gain Optimizer agent can perform as intended.

The Neural Network agent was developed to maintain the accuracy of the ANN model used in the Gain Optimizer agent. It does so by continuously collecting real-time power plant data and using it to adapt the layer-recurrent neural network (LRNN) to be accurate at simulating the current operating level, which is needed for the gain optimization procedure. The ability of the Gain Optimizer to produce improved gains shows that the Neural Network agent performs as intended, since the Gain Optimizer uses the LRNN trained by the Neural Network agent to evaluate the performance of the gains that are implemented.

The Interface agent, Delegation agent, Database agent, and Monitoring agent were developed to provide the ability for an operator to interact with the control system, agent management, a data storage and retrieval service, and a system monitor, respectively. There was not a good way to show the individual performance of these agents, since they provide services that do not directly affect the physical processes of the plant. However, the overall success of the MACS in controlling the PPS shows that they can perform as they were intended.

The Free agent was designed to provide flexibility in the MAS by enabling the assignment and reassignment of agents as needed. This agent was also designed to serve as a redundant backup should any of the other agents fail, as a number of extra Free agents can be initialized for the sole purpose of standing by to assume a failed agent’s task in the event of an agent failure. Similar to some of the other agents in the MAS, this agent’s operation was hard to demonstrate. However, all agents start as Free agents, and the MAS would not function if the Free agents did not perform as designed. Therefore, the overall success of the MACS in controlling the PPS shows that the Free agent functions as intended. Agent failures were not tested in this thesis as this issue was not a focus.

In addition to testing the functionality of individual agents, the MACS as a whole was tested. In the two tests that were performed, the MACS was shown to be able to implement successful control of the PPS for a wide-range ULDC and for a ULDC representing the load cycle for a typical summer weekday. These results show that the agents are able to work in harmony with each other to perform the intended function of the control system.

In conclusion, this thesis shows that a MAS can be used to implement a control system intended for a FFPU. This thesis also shows that agents can be used to implement each of the control techniques discussed here. Furthermore, this thesis shows that these agents can be successfully implemented simultaneously in a MAS to achieve the coordinated goal of customizable optimized multiobjective power plant control of an FFPU.

5.2 Future Research

Though the MAS discussed in this thesis was successfully developed and tested entirely using Matlab, it is desirable to use a computing platform that is better suited for multi-threaded programming. One such platform is the Java Agent Development Environment, or JADE. Like the name implies, JADE is built on Java, which is well-suited for multi-threaded computing. Another advantage to JADE is that the communication standard put forth by the Foundation for Intelligent Programmable Agents (FIPA) used in this project is already built in. Therefore, it is a recommendation of the thesis that future work pertaining to MASs meant for power plant control applications make use of JADE, while possibly making use of Matlab to perform the more complex computations.

Although this thesis discusses an MACS designed to control a small-scale FFPU, the MAS control methodology is intended for use in larger, more complicated FFPUs. A simple FFPU is used here so that the MACS could be designed and developed without the added difficulty of accounting for the complexity of the FFPU model. There has been work done to develop a MACS to control a 600 MW FFPU as an extension to this research, which was also done in Matlab. An opportunity for further research includes the development and testing of a MACS designed for a large-scale FFPU using JADE as the MAS platform.

Another opportunity for further research would be to develop more agents to perform additional tasks not addressed in this thesis, such as fault diagnosis. One appeal of the MACS method is extensibility. Because of this characteristic, there is much room for control functionality that can be added by developing more agent types.

The MACS, as well as modern conventional control systems, is heavily dependent on computer technology, and is therefore vulnerable to cyber attack. Cyber attacks can severely damage a power plant unit, causing power outages and potentially harming personnel operating the facility, among other things. Because of this threat, there are opportunities for research in the area of cyber security to develop methods for protecting power plant control systems from malicious manipulation.

APPENDICES

APPENDIX A

Operating Window Data

The two tables below contain the data needed to construct the power-pressure and power-input operating windows for the 160 MW power plant used in this thesis.

Table A.1 Upper pressure limit data.

E (MW)	P (kg/cm^2)	ρ_f (kg/m^3)	u_1 (p.u.)	u_2 (p.u.)	u_3 (p.u.)	L (m)
10.0	237.4	318.1	0.3225	0.2102	0.1226	0.0000
20.0	231.8	312.0	0.3610	0.2362	0.1683	0.0000
40.0	220.9	300.8	0.4385	0.2965	0.2607	0.0000
60.0	210.2	290.1	0.5167	0.3551	0.3544	0.0000
80.0	199.7	278.9	0.5956	0.4251	0.4495	0.0000
100.0	189.5	268.1	0.6752	0.5032	0.5459	0.0000
120.0	179.5	257.8	0.7555	0.5907	0.6439	0.0000
140.0	169.7	247.1	0.8366	0.6890	0.7434	0.0000
160.0	160.2	236.8	0.9183	0.7996	0.7996	0.0000
180.0	150.9	226.3	1.000	0.9245	0.9245	0.0000

Table A.2 Lower pressure limit data.

E (MW)	P (kg/cm^2)	ρ_f (kg/m^3)	u_1 (p.u.)	u_2 (p.u.)	u_3 (p.u.)	L (m)
10.0	32.0	493.0	0.0785	0.4205	0.0708	0.0000
20.0	32.0	485.2	0.1274	0.6554	0.1315	0.0000
40.0	36.3	468.8	0.2287	1.0000	0.2504	0.0000
60.0	52.1	449.9	0.3392	1.0000	0.3591	0.0000
80.0	67.3	430.2	0.4486	1.0000	0.4637	0.0000
100.0	82.1	409.2	0.5574	1.0000	0.5654	0.0000
120.0	96.5	386.2	0.6656	1.0000	0.6649	0.0000
140.0	110.7	360.2	0.7734	1.0000	0.7626	0.0000
160.0	124.6	328.8	0.8808	1.0000	0.8587	0.0000
180.0	138.4	284.6	0.9879	1.0000	0.9534	0.0000

APPENDIX B

Calculation of the RGA Matrix for the 160 MW FFPU Model

The relative gain array (RGA) matrix is calculated for the 160 MW FFPU Model used in this thesis. This is done to show the strength of interaction between the input and output variables defined by the third order three-input three-output power plant model to determine the configuration of feedback control loops. The result of this calculation is used in Section 2.4 to justify the configuration of the feedback controller. Reference [26] was used as a resource for the following calculations.

The first step in calculating the RGA matrix is to obtain a linear state-space model of the dynamic equations in (2.1) and (2.2). The linearization of the model was performed in [26], resulting in a model of the form:

$$\dot{x} = Ax + Bu \tag{B.1a}$$

$$y = Cx + Du \tag{B.1b}$$

$$A = \begin{bmatrix} -0.1 & \frac{9}{8}\left(0.073u_2 - 0.016\right)P^{1/8} & 0 \\ 0 & -\frac{9}{8}0.0018u_2P^{1/8} & 0 \\ 0 & -\frac{1}{85}\left(1.1u_2 - 0.19\right) & 0 \end{bmatrix} \tag{B.2a}$$

$$B = \begin{bmatrix} 0 & 0.073P^{9/8} & 0 \\ 0.9 & -0.0018P^{9/8} & -0.15 \\ 0 & -\frac{1.1}{85}P & \frac{141}{85} \end{bmatrix} \tag{B.2b}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & C_{32} & C_{33} \end{bmatrix} \tag{B.2c}$$

$$D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 253.287 & 4.7428P & -13.9684 \end{bmatrix} \tag{B.2d}$$

and

$$C_{32} = 50\left[\frac{60\left(1/\rho_f - 0.001538\right)0.8}{\left(1-0.001538\left(0.8P-25.6\right)\right)^2} + 0.0862\left(1.1u_2 - 0.19\right)\right] \tag{B.3a}$$

$$C_{33} = 50\left[0.1307 - \frac{60\left(0.8P-25.6\right)}{\left(1-0.001538\left(0.8P-25.6\right)\right)\rho_f^2}\right]. \tag{B.3b}$$

Next, the process gain matrix, K, and the transpose inverse of the gain matrix, $R = \left(K^{-1}\right)^{\mathrm{T}}$, need to be calculated, assuming K is invertible. The process gain matrix is obtained by calculating the transfer matrix, $T(s)$, for the state-space model defined in (B.2) and (B.3), and calculating the elements of K using the equation:

$$K_{ij} = \lim_{s\to 0} T_{ij}\left(s\right),\ i = 1,2,3 \text{ and } j = 1,2,3. \tag{B.4}$$

This produces the elements of K as follows:

$$K_{11} = \frac{A_{12}B_{21}}{A_{11}A_{22}} \tag{B.5a}$$

$$K_{12} = \frac{A_{12}B_{22} - A_{22}B_{12}}{A_{11}A_{22}} \tag{B.5b}$$

$$K_{13} = \frac{A_{12}B_{23}}{A_{11}A_{22}} \tag{B.5c}$$

$$K_{21} = -\frac{B_{21}}{A_{22}} \tag{B.5d}$$

$$K_{22} = -\frac{B_{22}}{A_{22}} \tag{B.5e}$$

$$K_{23} = -\frac{B_{23}}{A_{22}} \tag{B.5f}$$

$$K_{31} = \lim_{s\to 0} -\frac{A_{32}B_{21}C_{33} - A_{32}B_{22}C_{33}}{sA_{22}} \tag{B.5g}$$

$$K_{32} = \lim_{s\to 0} \frac{A_{22}B_{32}C_{33} - A_{32}B_{22}C_{33}}{sA_{22}} \tag{B.5h}$$

$$K_{33} = \lim_{s\to 0} \frac{A_{22}B_{33}C_{33} - A_{32}B_{23}C_{33}}{sA_{22}}. \tag{B.5i}$$

Because of the *s* in the denominator of equations (B.5g) through (B.5h), it is advantageous to rewrite the process gain matrix and its inverse as follows:

$$K = \lim_{s\to 0}\begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ \frac{\gamma_{31}}{s} & \frac{\gamma_{32}}{s} & \frac{\gamma_{33}}{s} \end{bmatrix} = \lim_{s\to 0}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{s} \end{bmatrix}\begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix} \tag{B.6}$$

$$K^{-1} = \lim_{s\to 0}\begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}^{-1}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix}. \tag{B.7}$$

Defining *L* appropriately, the following substitution can be made:

$$K^{-1} = \lim_{s\to 0}\begin{bmatrix} L_{11} & L_{12} & L_{13} \\ L_{21} & L_{22} & L_{23} \\ L_{31} & L_{32} & L_{33} \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix} = \lim_{s\to 0}\begin{bmatrix} L_{11} & L_{12} & L_{13}s \\ L_{21} & L_{22} & L_{23}s \\ L_{31} & L_{32} & L_{33}s \end{bmatrix}. \tag{B.8}$$

From (B.8), R, is defined as follows:

$$R = \left(K^{-1}\right)^{\mathrm{T}} = \lim_{s\to 0}\begin{bmatrix} L_{11} & L_{21} & L_{31} \\ L_{12} & L_{22} & L_{32} \\ L_{13}s & L_{23}s & L_{33}s \end{bmatrix} \tag{B.9}$$

Now, the RGA matrix is calculated in terms of (B.6) and (B.9) using a Hadamard product, or element-by-element product, to obtain the following:

$$\Lambda = \left(K^{-1}\right)^{\mathrm{T}} * K = \begin{bmatrix} L_{11}K_{11} & L_{21}K_{12} & L_{31}K_{13} \\ L_{12}K_{21} & L_{22}K_{22} & L_{32}K_{23} \\ L_{13}\gamma_{31} & L_{23}\gamma_{32} & L_{33}\gamma_{33} \end{bmatrix} \tag{B.10}$$

From the result in (B.10), the RGA matrix can be calculated for specific combinations of P, ρ_f, u_1, u_2, and u_3 to determine the strength of interaction between inputs and outputs at different operating points.

To design the feedback controller in Section 2.4 the RGA matrix was calculated for specific values of E, P, ρ_f, u_1, u_2, and u_3 encompassing the full range of operation, shown in plot form in Fig. B.1. From these calculations, the feedback controller was simplified to consist of control loops that modify the output variables using the input variable that most strongly affects it. The values of E vary from 10 MW to 180 MW in steps of 10 MW. For each of the values of E, variables P, u_1, u_2, and u_3 were generated by the reference governor described in Section 2.3. The values for ρ_f were chosen such that the steady-state value of L is zero for the corresponding values of E, P, u_1, u_2, and u_3. The variables are chosen this way, because the control references used to govern the operation

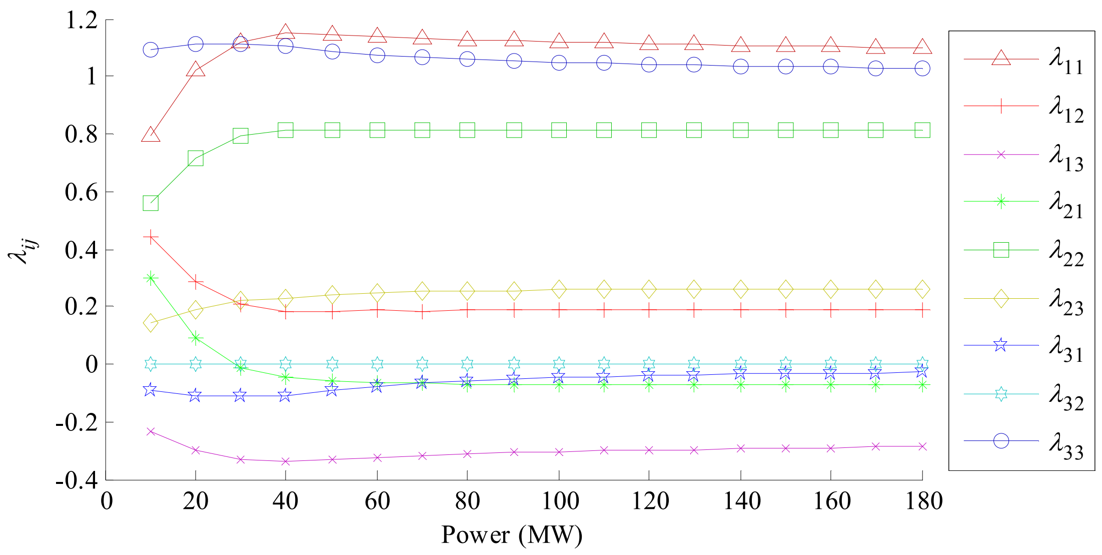


Fig. B.1. Result of RGA matrix calculations.

of the power plant unit are also chosen in this manner, and the feedback control system is meant to regulate the power plant system to match these references.

The rows of the RGA matrix, i=1, 2, 3, represent the output variables, E, P, and L and the columns of the RGA matrix, j=1, 2, 3, represent the input variables, u_1, u_2, and u_3, respectively. The elements of the RGA matrix, λ_{ij}, are normalized values representing the strength of interaction between the input and output variable corresponding to the row and column number. The strongest interaction between inputs and outputs is denoted by the value in each column closest to 1.

From Fig. B.1, it can be seen that λ_{11}, λ_{22}, and λ_{33} represent the strongest interaction for their column, for all power levels, where λ_{11} represents the interaction between the fuel valve, u_1, and the power output, E, λ_{22} represents the interaction between the steam valve, u_2, and the boiler pressure, P, and λ_{33} represents the interaction between the feedwater valve, u_3, and the water level deviation, L. Therefore, the feedback control loops shown in Section 2.4 were designed such that the error between the power output and the power setpoint, E_d-E, drives the feedback control for the fuel valve, the error between the boiler pressure output and the boiler pressure setpoint, P_d-P, drives the feedback control for the steam valve, and the error between the water level deviation and the water level deviation setpoint, L_d-L, drives the feedback control for the feedwater valve.

APPENDIX C

Artificial Neural Networks

ANNs were originally inspired by the structure, function, and processing power of the extensive networks of neurons in the human brain that coordinate sensory perception into muscular and glandular responses in the body. Comprising these networks are three types of neurons: sensory neurons, motor neurons, and interneurons. The sensory neurons are the neurons that receive input to the human neural network (HNN) from the nerve endings that make up the five senses: hearing, sight, touch, taste and smell. Based on the strength of the impulses received from the nerve endings, the sensory neurons will either fire their own impulses or do nothing. The interneurons serve to process information from the sensory layer, and relay the processed result to the motor neurons. The motor neurons are connected to different muscles and glands in the body, and have the ability to stimulate them based on the impulses received from the interneurons, forming the output layer of the human brain.

The neurons that make up the HNN are connected via axons and dendrites, which are used to send and receive electrical impulses, respectively. When a neuron receives an impulse, inhibitory or excitatory, through its dendrites, the sum of these impulses is evaluated against the neuron's excitation threshold, where the impulse strength is determined by the strength of the synaptic connection through which it is received. If the result exceeds a neuron's excitation threshold, then it will send an impulse through its axon to the other neurons or cells it is connected to.

Similar to the HNN, ANNs are comprised of artificial neurons that form an input layer, one or more processing hidden layers, and an output layer. The artificial neuron, as originally proposed by [14], models the neurons in the brain by summing its received inputs and evaluates them against a threshold to decide the output. This process is shown in Fig. B.1, where p_i are the inputs representing electrical impulses received by neurons in the brain, w_i are weights representing the strength of the synaptic connections between neurons, b is an adjustable bias value that adjusts the threshold level to assist with modeling learning in the brain, s is the sum of products of the inputs and weights plus the bias value, f is the activation function of the neuron, or excitation threshold function,

$$a = f(s) \tag{C.1a}$$

$$s = b + \sum_{i=1}^{n}(p_i w_i) \tag{C.1b}$$

which evaluates s to determine a neuron's output, and a is the resulting output, shown in equation form below where n is the number of inputs:

There are many functions that can be used as the activation function, f, in artificial neurons that make up ANNs. The log-sigmoid transfer function, or adaptations thereof, is a commonly used transfer function in artificial neurons, pictured below. This transfer function behaves differently from the way the McCullough-Pitts neuron treats incoming signals. The McCullough-Pitts neuron uses the hard-limit transfer function which outputs

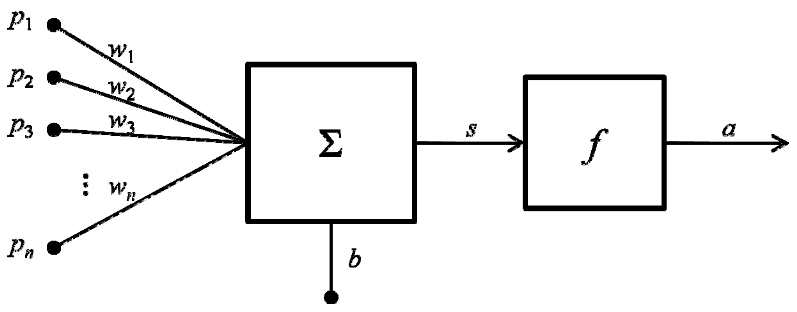


Fig. C.1. Process model of artificial neuron.

a zero when the input is below the threshold and outputs a one when the input is at or above the threshold. Instead, the log-sigmoid transfer function is an approximation of the hard-limit transfer function that allows a more dynamic response as opposed to the binary response of the hard-limit transfer function. Therefore, the log-sigmoid transfer function, shown in Fig. B.2, allows ANNs the ability to model the behavior of nonlinear systems more accurately than when using only the hard-limit function.

There are many types of ANNs that can be used to model dynamic systems, forecast system behavior, and recognize patterns, among other applications. Three main differences between types of ANNs that enable the different functionalities are the way neurons are interconnected, the number of hidden layers and artificial neurons used in each layer, and the excitation threshold functions used in the artificial neurons. Only two types of ANNs are used in this thesis: FFNNs and LRNNs.

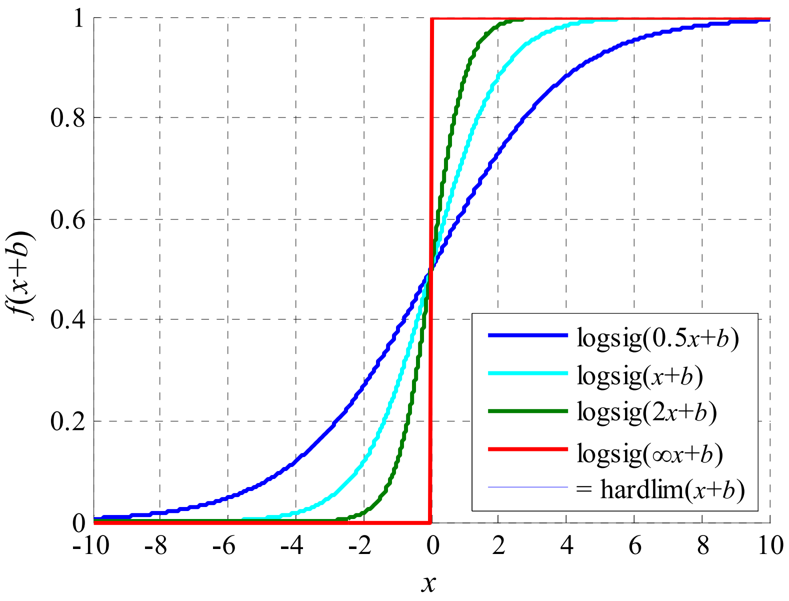


Fig. C.2. Log-Sigmoid and Hard-Limit transfer functions used in artificial neurons with a bias value, b, of zero.

BIBLIOGRAPHY

[1] *Annual Energy Outlook*, United States (U.S.) Energy Information Administration (EIA), 2011. Available: http://www.eia.gov/forecasts/aeo/

[2] *Coal-Fired Plant Upgrades*, United States Agency for International Development (USAID), 2011. Available: http://pdf.usaid.gov/pdf_docs/PNABK358.pdf

[3] Heo, J. S., K. Y. Lee , and R. Garduno-Ramirez, "Multiobjective Control of Power Plants using Particle Swarm Optimization Techniques",IEEE Transactions on Energy Conversion, Vol. 21, No. 2, pp. 552-561, June 2006.

[4] *International Energy Outlook*, United States (U.S.) Energy Information Amdministration (EIA), 2011. Available: http://www.eia.gov/oiaf/ieo/world.html

[5] Armor, A.F. (1985). Cycling of fossil plants: the key issue for the next 10 years. *Proceedings 1985 Fossil Plant Cycling Conference*. EPRI CS-4723.

[6] Heo, J. S. and K. Y. Lee, "A Multi-Agent System-Based Reference Governor For Multiobjective Power Plant Operation", IEEE Transactions on Energy Conversion, Vol. 23, No. 4, pp. 1082-1092, December 2008.

[7] S. D. J. McArthur, E. M. Davidson, V. M. Catterson, A. L. Dimeas, N. D. Hatziargyriou, F. Ponci and T. Funabashi, “Multi-Agent Systems for Power Engineering Applications – Part I: Concepts, Approaches, and Technical Challenges,” *IEEE Transactions on Power Systems,* vol. 22, no. 4, Nov. 2007.

[8] J. S. Heo and K. Y. Lee, “A Multi-Agent System-Based Intelligent Heuristic Optimal Control System for a Large-scale Power Plant,” *IEEE Congress on Evolutionary Computation,* Vancouver, BC, Canada, pp. 5693-5699, Jul. 2006.

[9] J. S. Heo and K. Y. Lee, “A Multi-Agent System-Based Intelligent Control System for a Power Plant,” *IEEE Power Engineering Society General Meeting,* San Francisco, CA, paper code: CD PESGM2005-000858.pdf, 2005.

[10] J. S. Heo and K. Y. Lee, “A Multi-Agent System Based Intelligent Reference Governor for Multi-objective Optimal Power Plant Operation,” *IEEE Transactions on Energy Conversion,* vol. 23, no. 4, 2008.

[11] R. Garduno-Ramirez and K. Y. Lee, “Multiobjective Optimal Power Plant Operation through Coordinate Control with Pressure Set-point Scheduling,” *IEEE Trans. Energy Conversion,* vol. 16, no. 2, pp. 115-122, Jun. 2001.

[12] R. D. Bell and K. J. Astrom, "Dynamic models for boiler-turbine-alternator units: Data logs and parameter estimation for 160 MW unit," Lund Institute of Technology, TFRT-3192, 1987.

[13] L. Kennedy and R. Eberhart, "Particle swarm optimization," *in Proc. IEEE International Conf. Neural Networks,* Perth, Australia, vol. IV, pp. 1942-1948, 1995.

[14] W. Pitts and W. McCoulloch, "How We Know Universals: The Perception of Auditory and Visual Forms," *Bulletin of Mathematical Biophysics*, vol. 9, pp. 127-147, 1947.

[15] C. Ku and K. Y. Lee, "Diagonal Recurrent Neural Networks for Dynamic Systems Control," *in IEEE Transactions on Neural Networks,* vol. 6, no.1, pp. 144-156, Jan. 1995.

[16] S. Russell and P. Norvig*, Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[17] P. Maes, "Artificial life meets entertainment: Life-like autonomous agents," *Commun. ACM*, vol. 38, no. 11, pp. 108–114, 1995.

[18] L. N. Foner, "Entertaining agents: A sociological case study," in *Proc. 1st Int. Conf. Autonomous Agents*, 1997.

[19] B. Hayes-Roth, "An architecture for adaptive intelligent systems," *Artif. Intell.* (Special Issue on Agents and Interactivity), vol. 72, pp. 329–365, 1995.

[20] M. Wooldridge, G. Weiss, Ed., "Intelligent Agents," in *Multi-agent Systems*. Cambridge, MA: MIT Press, Apr. 1999, pp. 3–51.

[21] J. D. Head, J. R. Gomes, C. S. Williams, and K. Y. Lee, "Implementation of a Multi-Agent System for Optimized Multiobjective Power Plant Cnotrol," *in Proc. of the 2010 North American Power Symposium*, Arlington, Texas, Sept. 2010.

[22] Foundation for Intelligent Physical Agents (FIPA), *FIPA Content Language Specifications.* 2003. [Online]. Available: http://www.fipa.org/repository/cls.php3.

[23] Foundation for Intelligent Physical Agents (FIPA), *FIPA ACL Message Structure Specification,* 2002. [Online].
Available: http://www.fipa.org/specs/fipa00061/SC00061G.html.

[24] R. Hecht-Nielsen, "Kolmogorovo's Mapping Neural Networks Existence Theorem," *Proceedings of the First IEEE International Conference on Neural Networks*, Vol. 3. Pages 112-114. San Diego 1987.

[25] R. Hecht-Nielsen, “Theory of Backpropagation Neural Networks,” *Proceedings of the International Joint Conference on Neural Networks*, Vol. 1. Pages 593-605. Washington 1989.

[26] R. Garduno-Ramirez, “Overall Intelligent Hybrid Control System for a Fossil-Fuel Power Unit,” Ph.D. dissertation, The Pennsylvania State University, 2000.