DESIGN OF A WIRELESS INTELLIGENT FUZZY CONTROLLER NETWORK

BAKHTIAR IBRAHIM SAEED

A thesis submitted to the University of Huddersfield

in partial fulfilment of the requirements for

the degree of Doctor of Philosophy

The University of Huddersfield

March 2014

Copyright statement

Abstract

Since the first application of fuzzy logic in the field of control engineering, fuzzy logic control has been successfully employed in controlling a wide variety of applications, such as commercial appliances, industrial automation, robots, traffic control, cement kilns and automotive engineering. The human knowledge on controlling complex and non-linear processes can be incorporated into a controller in the form of linguistic expressions. Despite these achievements, however, there is still a lack of an empirical or analytical design study which adequately addresses a systematic auto-tuning method. Indeed, tuning is one of the most crucial parts in the overall design of fuzzy logic controllers and it has become an active research field. Various techniques have been utilised to develop algorithms to fine-tune the controller parameters from a trial and error method to very advanced optimisation techniques.

The structure of fuzzy logic controllers is not straightforward as is the case in PID controllers. In addition, there is also a set of parameters that can be adjusted, and it is not always easy to find the relationship between the parameters and the controller performance measures. Moreover, in general, controllers have a wide range of setpoints; changing from one value to another requiring the controller parameters to be re-tuned in order to maintain a satisfactory performance over the entire range of setpoints.

This thesis deals with the design and implementation of a new intelligent algorithm for fuzzy logic controllers in a wireless network structure. The algorithm enables the controllers to learn about their plants and systematically tune their gains. The algorithm also provides the capability of retaining the knowledge acquired during the tuning process. Furthermore, this knowledge is shared on the network through a wireless communication link with other controllers.

Based on the relationships between controller gains and the closed-loop characteristics, an auto-tuning algorithm is developed. Simulation experiments using standard second order systems demonstrate the effectiveness of the algorithm with respect to auto-tuning, tracking setpoints and rejecting external disturbances. Furthermore, a zero overshoot response is produced with improvements in the transient and the steady state responses.

The wireless network structure is implemented using LabVIEW by composing a network of several fuzzy controllers. The results demonstrate that the controllers are able to retain and share the knowledge.

Acknowledgement

Table of Contents

8

List of Figures

11

List of Tables

List of abbreviations

ABFPD	Auto-Tuned Basic Fuzzy Proportional-Derivative + Integral
ACOAs	Ant Colony Optimisation Algorithms
ADC	Analogue Digital Conversion
AI	Analogue Input
ANNs	Artificial Neural Networks
AO	Analogue Output
BAs	Bees Algorithms
CE	Change of Error
CoG	Centre of Gravity
cu	Control output
DAC	Digital Analogue Conversion
DAQ	Data Acquisition
E	Error
EAs	Evolution Algorithms
FC	Fuzzy Controller
FD	Fuzzy Derivative
FI	Fuzzy Integral
FInc	Fuzzy Incremental
FLC	Fuzzy Logic Control
FP	Fuzzy Proportional
FPD+I	Fuzzy Proportional-Derivative + Integral
FPID	Fuzzy Proportional-Integral-Derivative
FS	Fuzzy Set
GAs	Genetic Algorithms
GA-FLC	Genetic Algorithm-Fuzzy Logic Control
GCE	Gain of Change of Error
GE	Gain of Error
GIE	Gain of Integral of Error
GU	Gain of Output
IE	Integral of Error
ISE	Integral of Squared Error
LA	Large
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
MD	Medium

MF	Membership Function
MIMO	Multiple-Inputs Multiple-Outputs
min	Minimum
M_p	Maximum percentage overshoot
NI	National Instruments
NI-PSP	National Instruments-Publish Subscribe Protocol
NL	Negative Large
NS	Negative Small
Op-amp	Operational amplifier
PD	Proportional-Derivative
PID	Proportional-Integral-Derivative
PL	Positive Large
PS	Positive Small
PWM	Pulse Width Modulation
RC	Resistor Capacitor
SF	Scaling Factor
SFLAs	Shuffled Frog Leaping Algorithms
SG	Scaling Gain
SISO	Single-Input Single-Output
SM	Small
SSE	Steady state error
TCP/IP	Transmission Control Protocol/ Internet Protocol
t_r	Rise time
t_s	Settling time
UDP	User Datagram Protocol
VI	Virtual Instrumentation
VL	Very Large
VS	Very Small
ZE	Zero

CHAPTER 1

Introduction

In this chapter, the context and the background of the work developed in this thesis are described. The necessity and the significance of the topic are briefly discussed. The aims and the objectives are outlined. The expected contributions and the structure of the thesis are also provided.

1.1 Overview

Many practical control systems and processes have dynamics and complex characteristics, such as non-linearity, uncertainty or changes in the process dynamics. These characteristics are not fully understood as is the case in most industrial process control systems (Babuska & Mamdani, 2008). Uncertainty arises for various reasons, for instance, lack of knowledge or the inability to obtain the required parameters of a system. This results in an imprecise understanding of the system dynamics and consequently the process of capturing and modelling of the characteristics and the attributes in mathematical models becomes too complicated (Babuska & Mamdani, 2008; Rutkowski, 2008; Shin & Xu, 2009). In addition, with ever-increasing demands on the control systems to accommodate further capabilities and features, such as autonomy and intelligent decision making abilities, these systems have tended to become increasingly more complicated than ever and consequently the modelling process become more complex.

On the one hand, the design of conventional linear controllers such as Proportional-Integral-Derivative (PID), the functionality and the performance of these controllers depend primarily

on the accuracy of the mathematical models. Hence conventional controllers have shown some limitations in controlling these systems particularly when they are applied to non-linear systems or when the circumstances surrounding a process under control are changing (Babuska & Mamdani, 2008).

On the other hand, in the real world, understanding of some complex systems does not require a high level of precision and accuracy. Such complex systems and processes can be intuitively controlled by humans. A skilled human operator can understand and model the behaviour of these systems and control them without having precise knowledge of their mathematical models (Mamdani, 1974).

The Fuzzy Logic Control (FLC) approach (Mamdani & Assilian, 1975; Zadeh, 1996), as an alternative method to the conventional control techniques, is based on an empirical approach rather than requiring an explicit and precise knowledge of the mathematical model of a system. This new control paradigm has emerged as one of the most used intelligent techniques in tackling the complexities associated with some systems. Since then, an ever-increasing employment of fuzzy logic controllers have been reported (Kumar, Nakra, & Mittal, 2011; Shen, 2008); they have been successfully applied in industrial processes and in some cases outperformed PID controllers (Saeed & Mehrdadi, 2011; Vaishnav & Khan, 2007). The prime reason for this is that FLC is deemed to be a systematic method to integrate the human knowledge in understanding, modelling and controlling a process in the form of linguistic terms to a controller. Hence, in essence, the controller emulates humans and becomes intelligent.

Despite the evident success of fuzzy controllers (FCs), there are a number of issues that might confine their applications (Altas & Sharaf, 2007; Gao, Trautzsch, & Dawson, 2002; Passino & Yurkovich, 1998).

First, there is still a lack of a systematic method to design and tune these controllers. The knowledge-base of the controller is designed using heuristic information from a skilled operator. Indeed, FCs have some essential components such as membership functions (MFs), rule-base and scaling factors (SFs) (Lee, 1990; Passino & Yurkovich, 1998; Woo, Chung, & Lin, 2000). Furthermore, each component has several parameters. The design process should consider all the components and the parameters and normally the design process may involve three stages. Firstly, appropriate membership functions are selected to map the input and output variables of the controller. In the second stage, the rule-base is constructed by translating the experience of a skilled human operator on controlling a plant into 'If-then' rules. And finally, an adjustment of the parameters is carried out.

The adjustment is a process, normally known as tuning. It is performed manually or automatically to determine appropriate controller parameters in order to achieve a better possible performance of a controller. As a nominal fuzzy controller cannot always provide a satisfactory performance, tuning is carried out to achieve a better possible response or at least to maintain the performance at a desired level whenever the characteristics of the plant under control vary, or when some disturbance is introduced.

Indeed, tuning is one of the most crucial parts in the overall design of FCs and it has become an active research field. Various techniques have been utilised to develop algorithms to fine tune the parameters from trial and error methods to very advanced optimisation techniques such as Genetic Algorithms (GAs) (Tang & Wu, 2009), Ant Colony Optimization Algorithms (ACOAs) (Juang & Chang, 2010), Shuffled Frog Leaping Algorithm (SFLAs) (Nguyen & Huynh, 2008) and Bees Algorithms (BAs) (Pham, Darwish, Eldukhr, & Otri, 2007) to name but a few.

Moreover, in general, controllers have a wide range of setpoints and changing from one value to another requires the controller parameters to be re-tuned in order to that a satisfactory performance over the entire range of the setpoints is maintained.

Also, there are other factors that make the tuning process more complex. Fuzzy controllers are, generally, non-linear (Jantzen, 2007); therefore, it is difficult to find the relationship between controller parameters and controller performance, such as rise time or overshoot. In addition, unlike conventional controllers, fuzzy logic controllers have more parameters that can be adjusted, such as: shape, position, number and type of the input and output membership functions; the input and output scaling gains; and the rule-base.

So far, a valuable amount of literature has been published on the design of tuning algorithms (Gang, 2006), but there is still, however, a lack of an empirical or analytical design study which adequately covers a systematic auto-tuning method. As most of the algorithms are application dependants, the development of a systematic auto-tuning algorithm remains of prime importance.

The second issue is that, even with the existence of an algorithm with the above mentioned attributes, the controller still ought to re-adjust its parameters whenever changes happen in the controller settings, for instance, if a particular setpoint is assigned to a controller then tuning is performed for that setpoint. At a later stage, when a new setpoint is assigned then accordingly the controller is re-tuned for this new setpoint. If the previous setpoint has to be re-assigned, the controller must start the re-tuning process, because the controller has no memory to retain the knowledge gained when the first setpoint was assigned (Antsaklis, 1999). Although re-assigning the setpoint is a repeated task, the controller does not learn anything from its past experience. If the knowledge was retained when the first setpoint was assigned then there could be opportunities to utilise it and the knowledge could be used at later stages when the same situations occur. It is equally important, that the knowledge could be shared with other controllers, thus, the controllers which receive the knowledge add it to their existing knowledge-base and hence learn from the past experience of the others.

Although considerable research has been devoted to tuning, no attention has been paid to the design of networked fuzzy controllers with the ability to retain and share the knowledge gained during tuning process. In essence, there is not a structure that incorporates memories and communications capabilities for networked fuzzy controllers in order to retain and share knowledge. Such a structure provides potential learning framework for the controllers and saves a significant amount of time required to re-tune the controllers.

An example can be found in process control for situations where a number of identical devices form a wired or a wireless network to work together towards achieving a universal goal. In order to maximise the network performance, the nodes are designed to control their local systems in an intelligent manner so that they can adapt to their setpoint changes and tune their local parameters. The knowledge gained by individual controllers is then made available to other controllers on the network. The structure could be potentially applied to systems with controlling of multiple motors with identical characteristics or several pneumatic valves on an oil pipeline.

This thesis proposes a new structure whereby a wireless intelligent controller network is designed and implemented. The controllers utilise FLC strategy to control their identical local processes. To improve their performances, the controllers have the ability to iteratively auto-tune their gains, and for this purpose a systematic algorithm is devised to perform the task. In addition, individual controllers also have the ability to retain the acquired knowledge in their memories so that it can be used when the controller requires it. Furthermore, the knowledge can be shared on the network through the communication link with other controllers.

As many real-time applications exhibit oscillation and overshoot in their step responses, and these characteristics can be modelled using a second order system (Haugen, 2009; Rowell, 2004), several standard second order transfer functions are considered as the process of the

controllers. To assess and validate the performance of the proposed algorithm, experiments are carried out on hardware circuits which represent these systems.

1.2 Problem Statement

Based on the aforementioned overview the key research question can be formulated as follows:

How can an auto-tuning algorithm with online self-learning, adaptation and extensive communication capabilities be developed for FCs in a wireless network structure?

1.3 Aims and Objective of the Research

This thesis aims to develop and devise a new auto-tuning algorithm for a group of FCs in a wireless network structure. The algorithm has the capabilities of self-learning, retaining the knowledge gained in tuning process and communications. The thesis also proposes a design framework where the algorithm can be implemented.

The major objectives of this research are set out as:

- Investigate various fuzzy controller structures and tuning techniques to select an appropriate type.
- Analyse the performance of the fuzzy controller with regard to its parameters to establish relationships between the parameters and the controller performance.
- Devise a novel systematic auto-tuning algorithm capable of online self-learning and tuning.
- Develop and implement a suitable multi-layer algorithm structure to enable the fuzzy controller to retain the knowledge gained in the tuning process.
- Develop a communication algorithm for the controller that affords an efficient exchanging and sharing of knowledge with other identical controllers.

- Assess, demonstrate and validate the effectiveness and the performance of the algorithms based on software simulation and hardware circuit experiments.

- Design and implement a wireless network structure to provide a learning platform for the controllers.

1.4 Contributions

This section briefly highlights the main expected contributions of this research work while a further detailed discussion and an assessment of the significance of the contributions are presented in the Conclusion chapter.

1.4.1 A Systematic Auto-tuning Algorithm

A systematic auto-tuning algorithm has been designed for a fuzzy logic controller with the ability to identify various plants. The controller is capable of monitoring the plant to ensure the stability and to improve its performance by determining appropriate controller parameters.

1.4.2 A New Structure for an Intelligent Fuzzy Logic Controller

A multi-layered structure algorithm has been designed to incorporate self-learning and knowledge retaining stages. The acquired experience and knowledge in the tuning process is retained and then utilised whenever a similar situation arises.

1.4.3 A New Knowledge Sharing Framework

A framework for networked fuzzy logic controllers has been designed and implemented. This new framework provides a learning facility through effective sharing of knowledge between identical FCs. The structure can be used as a new paradigm for implementing a practical multiple fuzzy controller system.

1.5 Publications

During the undertaking of the research, the fundamental parts of the work have been presented as papers at peer-reviewed international conferences and as electronic newspaper articles:

- Saeed, B.I. & Mehrdadi, B., (2011). Zero overshoot and fast transient response using a fuzzy logic controller. In *17th International Conference on Automation and Computing (ICAC)*, University of Huddersfield, UK. Conference Proceedings. 116 – 120 Retrieved from

  http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6084912&isnumber=6084889

- Saeed, B.I. & Mehrdadi, B., (2012). Design of an iterative auto-tuning algorithm for a fuzzy PID controller. In *25th International Congress on Condition Monitoring and Diagnostic Engineering (COMADEM)*, University of Huddersfield, UK. Journal of Physics: Conference Series, Volume (364) 012052 Retrieved from

  http://iopscience.iop.org/1742-6596/364/1/012052

- Saeed, B.I. & Mehrdadi, B., (2012). A Network Based Learning Architecture for Fuzzy Logic Controllers*.* In *International Conference on Advances in Electrical Measurements and Instrumentation Engineering (EMIE)*, Amsterdam, The Netherlands. Book Series: Computer Science Series, Volume (3) 127-133 Retrieved from http://searchdl.org/index.php/book_series/view/248

- Saeed, B.I. & Mehrdadi, B., (2012, September 11). Designing a Wireless Network of Intelligent Fuzzy Logic Controllers with NI LabVIEW. *National Instruments*.

  Retrieved from http://sine.ni.com/cs/app/doc/p/id/cs-14930

1.6 Thesis Layout

The rest of this thesis is organised into five chapters as follows:

Chapter 2 starts by providing some information on the complexities of modern control systems that have led to the emergence of the intelligent control field. It also reviews and evaluates the research literature of intelligent control design and its attributes. The significant role of fuzzy logic control in the design of intelligent controllers is highlighted. A focus review is given on the issues pertaining to FCs, such as structures and types. In addition, a further broad review of various tuning methods and algorithms is provided. Finally, a summary based on the reviewed literature is provided.

Chapter 3 introduces details of the proposed architecture of a wireless intelligent fuzzy controller network system. The main intelligent attributes of the system are defined and the methods to develop these characteristics are described. Additionally, the detailed design and development of the basic FPD+I controller is provided. The relationships between the basic FPD+I controller parameters and the controller performance is analysed and the design of the auto-tuning algorithm is provided. In order to evaluate and analyse the performance of the algorithms for a wide range of systems, standard transfer functions of several second order systems with different characteristics are chosen. These systems act as the plant for all simulation and experimentation tests in the thesis.

Chapter 4 gives a brief introduction to the software development platform used to implement the wireless intelligent fuzzy controller network. Also, the structure of the implemented system is provided. To apply the control algorithm to a real hardware, some standard second order transfer functions are realised and constructed using various electronic components.

Chapter 5 demonstrates the conduction of various tests and presents the results obtained. Firstly, MATLAB simulation-based models are used to test the basic FPD+I controller and the auto-tuning algorithm. Secondly, by using LabVIEW in addition to testing the basic FPD+I controller and the auto-tuning algorithm, the wireless intelligent fuzzy controller network system is tested. Furthermore, the results of the basic FPD+I controller and the auto-tuning algorithm tests on hardware representing various real-time control systems are presented. The chapter is summarised by providing discussions on the results where the relationships between the controller gains and the controller performance are established.

Chapter 6 presents the main concluding remarks and discussions on the research. It also outlines and discusses the major contributions of the thesis. In addition, the chapter provides possible future directions in the relevant areas.

CHAPTER 2

Literature Review

This chapter starts by providing some information on the complexities of modern control systems that have led to the emergence of the intelligent control field. It also reviews and evaluates the research literature of intelligent control design and its attributes. The important role of fuzzy logic in the design of intelligent controllers is highlighted. A focus review is given on the issues related to fuzzy controllers, such as structures and types. In addition, as tuning has been and remains one of the most important parts of the design of the fuzzy controllers, a further detailed review on the tuning methods and algorithms is provided. Finally, based on the reviewed literature, a summary of the literature is given.

2.1 From a Single Controller to Multi-Controller and Intelligent Control Systems

The principle of a basic feedback control system is to maintain the output of a process known as 'controlled variable' at a desired level which is referred to as the reference signal or the setpoint reference. In practical control applications, typically, the process has a single-input single-output (SISO) or multiple-inputs multiple-outputs (MIMO) (Skogestad & Postlethwaite, 2005). Depending on the control system types and requirements, various controllers are needed to control such processes ranging from plain on-off controllers (Edgar, Seborg, & Mellichamp, 2004; Visioli, 2006), through variations of proportional-integral-derivative controllers (PID) (Åström & Murray, 2009; Visioli, 2006), to adaptive and optimal controllers (Åström & Wittenmark, 1995).

With the ever-increasing demands on control systems to accommodate further capabilities and features, these systems have become increasingly complex than ever. In addition, most of these systems are inherently non-linear, time-varying or the system dynamics are very complex and not fully understood - as is the case in most industrial process control systems (Babuska & Mamdani, 2008).

However, recent advances in the fields of computer science, communication systems and embedded control technologies have led the experts in the control engineering field to develop several new control strategies, architectures and algorithms to meet the new requirements of these systems and to deal with the complexities.

For this reason, control system structures with a multi-controller function (Breemen & Vries, 2000) have significantly evolved as the most dominant control architectures, where multiple controllers are utilised to control a whole system.

Systems with multiple controllers have become standard in the control community. In the literature, the subject of multi-controller systems has been scattered over a variety of different research areas and mainly studied under various names and terms, such as distributed control (Tan, Yoo, & Yi, 2008a), multiple-control (Breemen & Vries, 2000), multi-agent control (Jennings & Bussmann, 2003), cooperative control, collaborative control and distributed learning control systems (Choi, Oh, & Horowitz, 2009).

These architectures have been successfully used in various application areas, such as industrial processes, power systems (McArthur et al., 2007), telecommunications, robots (Kelly & Keating, 1998) and automobiles, to mention but a few application areas.

With a quick survey on these architectures, it becomes clear that they have more advantages over a single control structure and they have been proposed for two main purposes. Firstly, to

manage the complexity of these systems more efficiently (Narendra, 2005), and secondly to achieve learning through collaboration between the different parts of a system (Choi et al., 2009).

In the monitoring of complex process control systems, an architecture of a multi-agent system has been introduced in (Tan, Yoo, & Yi, 2008b). In the context of learning, various works have been reported (Ahny, Chenz, & Moorex, 2011; Jorge Ierache, 2010; Kelly & Keating, 1998; Lu, Zhang, & Xie, 2011; Schöllig, Alonso-Mora, & D'Andrea, 2010). Researchers in (Kelly & Keating, 1998), have described a learning algorithm for small autonomous mobile robots, where two robots learn how to avoid obstacles and the learned knowledge is then passed on to each other. An analytical investigation of a multi-agent environment was carried out in (Schöllig et al., 2010), where the agents perform similar tasks and exchange information between them. The results showed a performance improvement and a faster learning rate of the individual agents.

In parallel to the aforementioned control architectures, intelligent control has emerged as one of the fastest growing areas in the field of control engineering over the last decades to deal with the complexities of these control systems.

Intelligent control utilises and develops algorithms and designs based on the emulation of intelligent behaviours of biological beings, such as in the way they perform a task or how they can find an optimal solution to a problem, to design, model and control complex systems (Antsaklis, 1999; Narendra, 1991; Shin & Xu, 2009). These behaviours may include adaptation to new situations, learning from experience, automation and cooperation in performing tasks (Antsaklis, 1999).

Preliminary work on intelligent control systems was undertaken in 1971 by Fu (1971), where the term of 'intelligent control' was coined (Antsaklis, 1994, 1999; Lima & Saridis, 1996;

Zilouchian & Jamshidi, 2001). Since then, the foundation of intelligent control systems was proposed by Lima and Saridis (1996), in which an architecture consisting of a three-level hierarchal structure was discussed and suggested. Since then, only a handful of papers and articles have been produced (Antsaklis, 1994, 1999; Åström, 1989; Lima & Saridis, 1996; Narendra, 1991; Passino & Yurkovich, 1998). These resources are still the main sources, but the works have tended to focus on general principles of intelligent controllers rather than fundamental procedures on the design and implementation of such controllers.

In Åström (1989), some features of automatic tuning and adaptation were outlined. The concept of combining these properties with supervision and monitoring were also given in order to develop a new generation of control systems, known at then as 'expert control systems'.

The necessity of possessing long term memory for adaptive controllers was discussed by Lima and Sardis (1996). Additionally, the concepts of knowledge-based systems in coordinating the various control algorithms used in process control were suggested.

A further detailed illustration of a functional architecture is given by Antsaklis (1999). The architecture consists of three levels, namely from bottom to top they are execution, coordination and management or organisational levels. This architecture is regarded as a standard structure in the design of intelligent control systems.

In general, intelligent control has various techniques and tools to design intelligent controllers. The tools are commonly known as soft computing or computational intelligence tools (Eberhart & Shi, 2007; Engelbbrecht, 2007; Rutkowski, 2008; Shin & Xu, 2009; Zilouchian & Jamshidi, 2001), and the major and widely used examples are listed below:

1- Fuzzy Logic (FL).

2- Artificial Neural Networks (ANNs).

3- Evolution Algorithms (EAs).

The tools have been extensively used in the design of many intelligent control systems, a diverse range of applications are reported in (Eberhart & Shi, 2007; Engelbbrecht, 2007; Liu & Wang, 2006; Ruano, 2005; Rutkowski, 2008; Shin & Xu, 2009). As this thesis deals with the design of intelligent fuzzy controllers, only the role of fuzzy logic in the field of control engineering in general and specifically in the design of intelligent controllers will be discussed and reviewed in the following sections.

2.2 Fuzzy Logic Control

Many practical control systems and processes have dynamic and complex characteristics, such as uncertainty and non-linearity. In addition, the complexity of such systems increases with the ever increasing demands for autonomy and intelligent decision making abilities. Accurate mathematical models to capture and model all these characteristics and attitudes are either not easily attainable or too complicated (Babuska & Mamdani, 2008; Rutkowski, 2008; Shin & Xu, 2009).

The design of conventional controllers such as PID and the functionality and performance of these controllers depend on the accuracy of the models. Therefore, applying conventional controllers has revealed some limitations in controlling these systems particularly when they are applied to non-linear systems or when the circumstances surrounding a process under control are changing (Babuska & Mamdani, 2008).

On the other hand, such systems and processes can be intuitively controlled by humans. A skilled human operator can understand the behaviour of these systems and control them without having precise knowledge of their mathematical models (Mamdani, 1974).

Fuzzy logic control (Mamdani & Assilian, 1975; Zadeh, 1996) is one of the most successful application fields of fuzzy sets (FSs) invented by Zadeh (Zadeh, 1965). It is an approach to integrate the emulation of the human brain in understanding, modelling and controlling uncertainties and non-linearities in complex systems into a controller. The first application of fuzzy logic in the field of control engineering was carried out in 1974 (Mamdani, 1974; Mamdani & Assilian, 1975) . The motivation was to find a way to introduce the human knowledge in controlling a process, into a controller. The researchers successfully carried out an experiment to control a dynamic plant which consisted of a laboratory model steam engine and a boiler combination. Since then, fuzzy logic has been successfully applied to a broad spectrum of real-world application areas ranging from industrial processes, consumer electronics, power systems, telecommunications, robots and automobiles, to mention but a few (Grigorie, 2011; Iqbal, Boumella, & Garcia, 2012; Ross, 2004). Perhaps the most popular area has been in the field of modelling, controlling and monitoring complex control systems (Shen, 2008).

2.3 Intelligent Fuzzy Controllers

Fuzzy logic has a more significant role to play in the design of control systems, and in particular in the design of intelligent control systems than any other method and this seems to be for a number of reasons. In the first place, the first application of fuzzy logic in the field of control engineering was as a direct controller, to control a time-varying and non-linear plant; therefore, it is widely used as a normal controller in direct action with a process. Secondly, because fuzzy logic is also seen as a universal approximation tool, it can be used to model various complex control systems. Finally, fuzzy logic has the capability to incorporate decision support at higher levels of the control hierarchy, such as supervision and monitoring; therefore, it is widely used as a tool in switching and decision-making processes (Sala, Guerra, & Babuška, 2005). In summary, fuzzy logic is regarded as a systematic method to integrate the human knowledge in understanding and modelling of complex control problems into a

controller. The integrated knowledge is in the form of linguistic terms, which is very well understood and in addition, it can be easily changed.

However, there are still limits as to how far a controller, utilising fuzzy logic as a tool, is considered to be intelligent. With a closer look at the literature, all fuzzy logic controllers designed so far have been, literally, regarded and entitled as intelligent controllers (de Silva, 1995; Kia, Far, Omid, Alimardani, & Naderloo, 2009; Kumar & Garg, 2004; Mary, Marimuthu, & Singh, 2007; Saifizul, Zainon, Osman, Azlan, & Ibrahim, 2006; Wang, Liao, Liao, Suen, & Lee, 2009; Yu, Huang, & Zeng, 2005). Critics have argued that not all fuzzy controllers could be considered intelligent (Antsaklis, 1999), and this appears to be quite reasonable. It is true that human knowledge is well represented and integrated into fuzzy controllers, but this knowledge is static (fixed) and cannot update itself. A mechanism is essential to expand and update this knowledge in order to make the controller acquire further information to improve its performance, adapt to new situations and learn from its own experience.

Even with a controller with such a mechanism, an issue arises in achieving learning feature for fuzzy controllers. For example, an intelligent controller with an adaptation mechanism can adapt to new situations by adjusting some of its parameters. In a sense, the controller acquires new information and learns how to adapt itself and to improve its performance. However, at a later stage, if the same situation arises, the intelligent controller still needs to re-adjust its parameters, this is because the controller has no memory and therefore no information is retained. Consequently, for a fuzzy controller to be intelligent, memory is essential so that the information acquired in the adaptation or tuning processes in previous circumstances is retained and can be used at later stages when the same situation occurs (Antsaklis, 1999). Consequently, beside the standard knowledge-base of a fuzzy controller, an additional knowledge-base is gradually built based on the past experience of the controller.

It is also possible that a fuzzy controller can learn from other identical fuzzy controllers performing the same task. This can be achieved by forming a network of identical fuzzy controllers to establish a framework for exchanging information between them. In order to maximise the network performance, the controllers are designed to control their local systems in an intelligent manner so that they can adapt to their setpoint changes and tune their local parameters. The knowledge gained by a controller in performing a particular task, tuning for example, can be shared with other controllers, thus, the controller which receives the knowledge extends its existing knowledge-base.

In addition to the abovementioned issues regarding intelligent fuzzy controllers, there are still several other issues to be addressed, such as selection of the controller type and structure and the tuning. These issues add further complexities to the design of intelligent fuzzy controllers. In the coming sections, critical reviews on these issues are provided.

As various terms related to fuzzy control systems will be used throughout this thesis, the reader is referred to Appendix A, where a basic design example of a fuzzy controller is given to illustrate various parts and components of the controller.

2.4 Choice of the Controller Type and Structure

Mainly, there are two types of fuzzy logic controllers (Engelbbrecht, 2007; Jantzen, 2007), Mamdani, (Mamdani, 1974) also referred to as a 'standard fuzzy system' (Passino & Yurkovich, 1998) and Takagi-Sugeno controllers (Takagi & Sugeno, 1985). In both types, the rule-base is constructed by a collection of empirical linguistic rules in the form of the 'If-then' format. The antecedent part on the 'if' side accommodates the input variables with their fuzzy set values, while the consequent part on the 'then' side accommodates output variables with their fuzzy set values. Thus, the rule-base expresses a mapping between the input and the output linguistic variables. The main difference between the two types of fuzzy controllers is that, in the Mamdani type, the antecedent part is statically constructed, whilst in the Takagi-Sugeno

type it is dynamic, whereby the output is a linear combination of the input variables or it is a complex function of the inputs. In some literatures, the Takagi-Sugeno type is referred to as 'functional fuzzy systems' (Passino & Yurkovich, 1998).

Table-based fuzzy controllers mentioned in some literatures (Engelbbrecht, 2007; Jantzen, 2007), can represent one of the two types, where the controller's universes of discourse are discrete known as 'singletons'; in this case it is possible to calculate all the controller outputs in advance and store them in a table. Then, based on the inputs of the controller, the output is found in the table as a look-up search process. The execution speed is improved as the output values are identified faster.

As is the case in the conventional PID control, for a particular application it is not necessary to adopt all three terms of proportional (P), integral (I) and derivative (D) (Visioli, 2006). Therefore, the selection of the controller structure is essential in the overall design of a control system. Fuzzy controllers are constructed in various structures. Several structures have been proposed and extensively studied, such as: fuzzy-proportional (FP), fuzzy-proportional-derivative (FPD), fuzzy-proportional-integral (FPI) or fuzzy-incremental (FInc) and fuzzy-proportional-integral-derivative (FPID) (Duan, Deng, & Li, 2013; Jantzen, 2007; Mann, Hu, & Gosine, 2001; Yung-Yaw & Chen-Cheng, 1992).

An FPD controller results in a stable system, but with a steady state error, while an FPI controller achieves a zero steady state error, but with a less stable response (Yung-Yaw & Chen-Cheng, 1992). To combine the advantages of FPD and FPI controllers, various structures for an FPID controller have been proposed. Indeed, in (Shin & Xu, 2009) a normal FPID with three inputs (the error, the change of error and the integral of error) has been proposed and implemented. Although the controller has a plain structure, the construction of a three-dimensional rule-base becomes more difficult as the number of rules exponentially increases

with the increasing number of controller inputs. Furthermore, constructing rules based on an integral of error is rather difficult (Engelbbrecht, 2007; Jantzen, 2007).

To overcome these limitations, a parallel structure which is a combination of FPI and FPD, and FPD+I, has been proposed (Li, 1997; Pivonka, 2002). The Parallel structure has two inputs, resulting in a two-dimensional rule-base. Additionally, it has the basic properties of a general PID controller as the three control actions are included, but at the same time further computational time is required to determine the controller output as there are two fuzzy controllers in the structure. The FPD+I is normally constructed by combining a crisp integral action with the output of an FPD controller, hence the rule-base is still two-dimensional and the controller also has the merits of a general PID controller (Engelbbrecht, 2007; Jantzen, 2007).

Further configurations have been found in the literature, such as: FPID with incremental output, rule coupled FPI+FPD, rule de-coupled FPID, FP+I+D and FPI+D controllers (Pivonka, 2002). In de-coupled rule-base structures, a separate rule-base for every controller input is constructed. The key problem, however, with the existence of various structures is that more complexities are introduced to the design and tuning of fuzzy logic controllers.

2.5 Tuning Fuzzy Controllers

Fuzzy logic controllers have some essential components, such as membership functions, rule-base and gains. Each component has several parameters. In general, the design process of these controllers should consider all the components and parameters and therefore it may involve three main stages (Jantzen, 2007; Passino & Yurkovich, 1998; Woo et al., 2000). Firstly, appropriate membership functions are selected to map the input and output variables. In the second stage, the rule-base is constructed by translating the experience of a skilled human operator on controlling a plant into 'If-then' rules. And finally, an adjustment of the parameters is carried out to achieve a better possible performance.

Additionally, as the knowledge-base of a fuzzy logic controller is designed based on heuristic information from skilled operators, the performance of such controllers begins to deteriorate when the dynamics of the system change, or unexpected disturbances occur or the setpoints change. Therefore, a mechanism is required to adjust the controller parameters to maintain the system performance at a satisfactory level.

The adjustment process that has been mentioned in the literature generally refers to tuning the parameters of the membership functions and the gains, while self-organising refers to adjustments of the rule-base components. Furthermore, self-tuning, auto-tuning, self-learning, learning and adaptive are also used as alternative terminologies (Jie, Liyun, Derong, Yanbo, & Shiyu, 2008; Mamdani & Assilian, 1975; Wang, 1994; Zacharie, 2010).

Indeed, tuning is one of the most important parts of the design of fuzzy controllers and it has a great impact on improving the performance of these controllers. Regardless of the adjustment mechanism, the tuning process is performed to improve and maintain the performance of the controllers, if the characteristics of the plant under control vary, or some disturbance is introduced. Moreover, in general, controllers have a wide range of setpoints; changing from one setpoint to another requires the controller parameters to be re-tuned so that a satisfactory performance over the entire setpoint range is achieved and maintained.

However, there are some serious issues that make the tuning process of fuzzy logic controllers more complex. These controllers are generally non-linear and therefore it is difficult to find the relationship between controller parameters and controller performance, such as rise time or overshoot (Jantzen, 2007). In addition, unlike conventional controllers, fuzzy logic controllers have several parameters that can be adjusted, such as: shape, position, number and type of the input and output membership functions; the input and output gains and the rules. Furthermore, there are no general rules for tuning these parameters. Most of the methodologies have been based on heuristic information. However, as the fuzzy logic control

has its foundation in conventional control, some techniques and ideas applied in tuning conventional controllers can still be utilised to some extent (Passino & Yurkovich, 1998).

As an extension to the original fuzzy logic controller, the first self-organizing fuzzy logic controller has been developed (Procyk & Mamdani, 1979) which has the ability to change the control policy and to improve its performance automatically based on a pre-determined performance index. Since then, tuning has become one of the most active research fields in the design of fuzzy controllers. Various techniques and algorithms have been utilised and developed to fine-tune the controller parameters, ranging from trial and error methods to very advanced optimisation techniques.

Tuning algorithms involve adjusting various controller parameters including input and output gains, input and output membership functions' parameters and fuzzy rules. In the next sections the main methods will be reviewed in further detail.

2.5.1 Tuning via Trial and Error Methods

In most cases, a nominal fuzzy controller is designed for a plant and then, based on some prior knowledge, the controller parameters are adjusted by a process of trial and error until a satisfactory performance is achieved (Nguyen & Huynh, 2008; Passino & Yurkovich, 1998). The adjustment might include the input and output membership functions' parameters, the input and output gains or the rule-base system. This approach is very easy and straightforward, but it is a tedious and time-consuming task (Murad, Cheok, & Das, 2009), particularly when it is carried out on-line. Furthermore, there are many cases where prior knowledge about the plant is unknown. Therefore, the technique is impractical and yields no significant contribution to the design of intelligent fuzzy controllers.

2.5.2 Tuning via Intelligent Optimisation Techniques

In order to deal with the difficulties in tuning fuzzy logic controllers and find optimal values of the parameters, the use of several intelligent optimization tools has been proposed by many researchers. These tools, also known as Evolutionary Algorithms (EAs), are inspired by the functionality of intelligent biological systems, either in the way they perform a task or how they find an optimal solution to a problem (Antsaklis, 1999; Narendra, 1991; Shin & Xu, 2009).

The primary application of these tools is to find a global optimal point in the search domain; therefore, they are suitable for optimisation or search problems such as optimal design of controller parameters. Tuning fuzzy controller parameters can be formulated as search or optimisation problems; hence, in this case, the tools can be effectively used to find better values of the parameters.

The tools have been successfully utilised to reduce the time and effort involved in the design and fine-tuning of fuzzy controllers (Hoffmann, 2001). Some examples are Genetic Algorithms (GAs) (Tang & Wu, 2009), Ant Colony Optimization Algorithms (ACOAs) (Juang & Chang, 2010), Shuffled Frog Leaping Algorithms (SFLAs) (Nguyen & Huynh, 2008) and Bees Algorithms (BAs) (Pham et al., 2007). Furthermore, in some cases hybrids of these techniques have been used (Dalci, Uzunoglu, & Kucukdemiral, 2004; Khan et al., 2008; Pham et al., 2006; Soliman, Guan-zheng, & Abdullah, 2008).

As these tools share similar principles in operation, only a brief illustration of the most widely used is given below. Genetic algorithms have been used as search and optimization techniques to find the appropriate value of fuzzy controller parameters. They can be used in two ways: off-line or on-line. The off-line approach requires the plant transfer function and then a simulation is carried out, while the on-line method is used whilst running the controller in real-time. The controller parameters are encoded into the algorithm, and a fitness function

comprising of the desired performance is defined. Then a population of solutions is generated. After that, the specification of individuals in the solutions is measured against the fitness function in order to determine how close each solution is to the desired performance. Finally, the fitted individual solution is selected as an optimal solution for the controller (Johnson & Picton, 2001; Karr, 1999; Rutkowski, 2008).

Using GAs with fuzzy control is normally known as GA-FLC; it has been successfully used to design fuzzy controllers and fine tune their parameters. For instance, it has been used to optimize the membership functions' parameters (Kumar & Garg, 2004). In (Khan et al., 2008), this approach has been used to optimise the rules, the membership functions' parameters and the gains. Also, it has been used in multi-objective optimization problems (Stewart, Stewart, Gladwin, & Parr, 2009).

Most recently, Bees Algorithms as another populated-based optimization algorithm, have been utilised in the design and optimisation of fuzzy controller parameters (Pham et al., 2006). The algorithm has been used to tune fuzzy controller parameters (Pham et al., 2007; Pham & Kalyoncu, 2009).

Although these algorithms are powerful and their successes have been proved, they are computationally expensive (Lu & Mahfouf, 2010). Because they are population-based algorithms, a considerable number of solutions are generated; individuals of these generations are needed to be tested for their fitness function. Additionally, some individuals cannot be tested in real-time in safety-critical applications. Therefore, these techniques are best suited for simulation-based designs where the plant transfer function is available.

2.5.3 Tuning via Supervisory Algorithms

These algorithms are mainly based on the observation of some of the control system signals, such as the error, the change of error, the control signal, the plant output or (a combination of them) to adjust the controller parameters. In this scheme, an upper level algorithm or a fuzzy

controller acting as a system supervisor, normally called a 'supervisor controller', is added to a system to monitor the performance of another direct fuzzy controller. Based on the observation of the system performance, the upper level controller performs adjustments to the direct fuzzy controller parameters (de Silva, 1995; Jantzen, 2007; Passino & Yurkovich, 1998; Wang, 1994); hence, the performance is improved or maintained.

Using the error and the change of error signals, an online tuning of the input gains has been designed by Chopra, Mitra and Kumar (2008), where some of the performance parameters, such as overshoot, rise time, settling time and Integral of Squared Error have been used to measure the controller performance. Based on the error and the control signal, another system has been designed to tune the input gains (Kanagaraj, Sivashanmugam, & Paramasivam, 2008, 2009). The proposed system has been tested in a real-time non-linear pressure process. In (Jie et al., 2008), by using only the error signal as input, an adaptive fuzzy controller to tune the input and output gains has been proposed.

2.5.4 Tuning Methods from Conventional Control Algorithms

As fuzzy logic control has its foundation in the conventional control, there are some similarities between them (Li, 1997; Lilly, 2011; Pivonka, 2002). For example, under certain conditions, an FPID controller is equivalent to a PID controller. In both, the control output is obtained from a combination of the error, the change of error and the integral of error. The combination is linear in the case of the PID controller, while it is 'fuzzified' in the case of the FPID controller and the control strategy in the FPID controller is formulated in the form of linguistic terms (Jantzen, 2007; Li, 1997; Mizumoto, 1992; Pivonka, 2002).

There are several well-known and understood methods for tuning PID controllers, such as Ziegler-Nichols, Kappa-Tau and pole placement (Åström & Murray, 2009; Åström & Wittenmark, 1995). Even when these methods are not utilised, an expert can often manually tune PID controller gains in order to achieve a stable response (Lilly, 2011). This approach has

enabled researchers to establish the use of various ideas and analysis methods from conventional PID control in the design and tuning of fuzzy controllers.

Indeed, in (Mizumoto, 1992), a PID controller was realised by using fuzzy control methods and it was argued that PID controls are a special case of fuzzy controls. A comparative gain design has been presented in (Li, 1997), where the values of the gains of a well-tuned conventional controller were used as the initial values of fuzzy controller gains. Based on the physical relationship between a PID controller and some fuzzy PID controller structures, such as FPI+FD and FP+FI+PD controllers, the well-tuned gains of the PID controller were transferred to the fuzzy controllers and used as the initial values of the fuzzy controller gains (Pivonka, 2002).

A detailed design procedure to design and tune a fuzzy PID controller, has been proposed in (Jantzen, 2007). The process started with the design of a tuned PID controller and then the PID controller was subsequently replaced by an equivalent linear fuzzy controller. The fuzzy controller was made non-linear, and finally its gains tuned manually.

Although the above-mentioned procedures have produced well-tuned fuzzy controllers with a satisfactory performance, the major drawback of these techniques is that the procedure is too long, and sometimes the tuning procedure is overlooked. Indeed, some of them were involved in the design of an equivalent PID controller, which increases the burden required to design and tune a fuzzy controller. Additionally, the transferred gains were used as the initial values for the fuzzy controller and then the controller was required to be further tuned manually.

2.6 Summary

As mentioned in the previous sections, fuzzy controllers have various structures and types. In addition, the components of a fuzzy controller have several parts, such as: number, type, position of the input and output membership functions; the input and output gains; and the rules. These variations in the controller structure have significant effects on the performance of the fuzzy controller.

The issues of fuzzy controllers have been partially addressed by many researchers in the context of their intended applications. Due to non-linearity and inconsistent structure of fuzzy controllers, difficulties have also arisen when attempts have been made to design a general purpose fuzzy logic controller.

Although valuable research has been carried out on the design of auto-tuning algorithms for fuzzy controllers, there is still a lack of an empirical or analytical design study which adequately covers a systematic auto-tuning method. In addition, most of the tuning algorithms involves in tuning of several controller parameters which make the tuning process more complex. Furthermore, providing a clear and physical meaning to the tuned parameters, as is the case in PID controller (Visioli, 2006), has been overlooked.

Indeed, the tuning efforts have remained limited and local to a controller and there is still a lack of a framework to retain the knowledge for future use and to share this knowledge with identical controllers performing similar tasks.

CHAPTER 3

Design of a Wireless Intelligent Fuzzy Controller Network

This chapter provides details of the proposed architecture of a wireless intelligent fuzzy controller network system. The main intelligent attributes of the system are also defined and the methods to develop these characteristics are illustrated. Additionally, the design of a basic FPD+I controller is provided in details. The relationships between the basic FPD+I controller gains and the controller performance is analysed and established. Then the design of the auto-tuning algorithm is also presented.

3.1 Attributes

The purpose of any intelligent control system is to incorporate, to a certain extent, some intelligent attributes normally associated with humans or biological beings into a system. Therefore, it is appropriate at this point to briefly define some essential attributes required for any intelligent controller and any network of intelligent controllers. They should have:

1- A systematic method to easily represent and manipulate knowledge. This is because an intelligent controller needs to manage different aspects of a system under control. The knowledge representation mainly deals with different control strategies and the relation between input and output of a system.

2- The ability to identify various plants, such as stable plants, unstable plants; and to learn about them, this is to determine appropriate control algorithms required to control each plant type.

3- The capability to improve the performance of the controller through auto-tuning its parameters. For this, a systematic auto-tuning algorithm is essential.

4- The ability to adapt to new situations, this is to maintain the performance of the controller at a satisfactory level for the whole operating setpoints of the system.

5- The ability to learn from past experience and knowledge. Past knowledge may include; assigned setpoints, controller settings and type of plant; and past experience for example may include auto-tuning controller parameters, and adaptation to new situations. Learning can be achieved in single controller structures and in networked controller structures. In a single controller, memory is essential, where the acquired experience and knowledge are retained and restored when a similar situation arises. While in networked controllers this could be achieved by sharing the acquired experience and knowledge between different identical controllers. In essence, possessing memory and sharing knowledge.

6- The ability to monitor the plant and supervise various tasks in order to ensure the stability of the plant.

3.2 Architecture of the Intelligent Fuzzy Controller Network

Based on the reviews in the previous chapter, the architecture of a network of intelligent fuzzy controllers shown in Figure 3.1 is proposed. The architecture consists of a network of several identical wireless intelligent fuzzy controllers. A network coordinator is used to establish and to facilitate communications between the controllers for exchanging information. Based on the application requirements, the communication link could be achieved through using any variants of standard wireless local area network (WLAN) Wi-Fi/ IEEE standard 802.11, wireless personal area networks (WPAN), such as ZigBee/ IEEE standard 802.15.4, WirelessHART (Elahi & Gschwender, 2009) or any other type of wireless networks (Millan, Vargas, Molano, & Mojica,

2011; Tipsuwan & Chow, 2003; Yunfeng, Yan, & Rui, 2011; Yuqing & Huajing, 2005). The controllers have two interfaces, one to the users and the other to their respective processes: the user interface is graphical and allows setpoints assigning and monitoring, while the process interface is physical and interacts directly with the process. Intelligent control algorithms are used to control the processes and the controllers have the capability of identification, auto-tuning, adaptation and learning. Each controller can operate independently or dependently inside the network to achieve its goal.
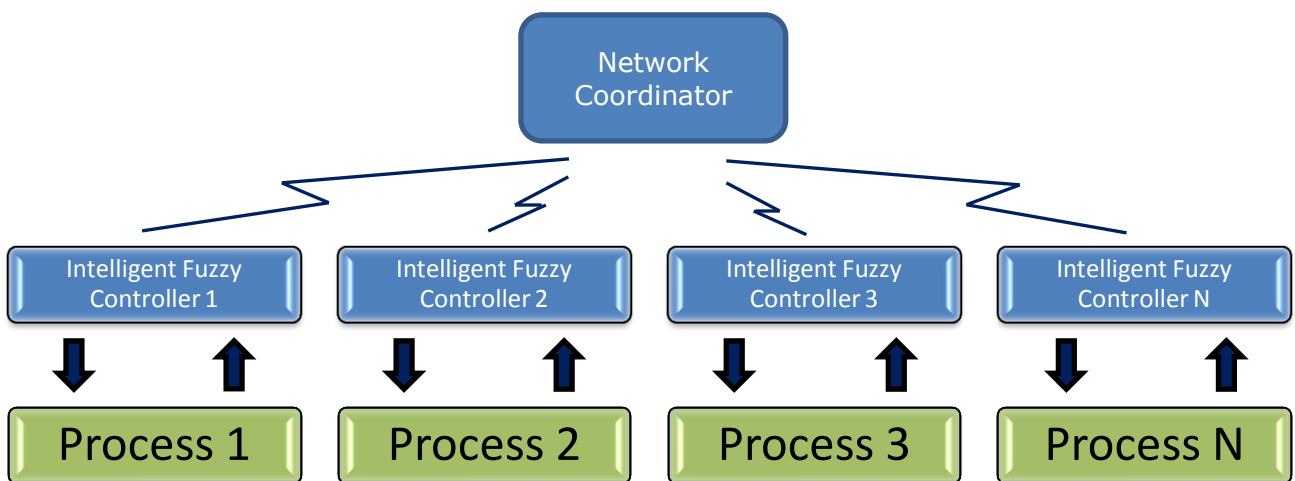
Figure 3-1. The proposed architecture of a wireless intelligent fuzzy controller network.

3.3 Intelligent Fuzzy Controller Structure

As the controllers act independently to achieve the required attributes; these attributes need to be implemented inside the controllers. Additionally, the attributes are quite diverse and complex to be structured in a single-level or in a single-loop control algorithm. Therefore, it is often convenient and easy to implement in a hierarchal multi-layer algorithm structure, for this, a general intelligent controller structure mentioned in (Passino, 2005) was adopted.

Figure 3.2, shows the inside structure, and it consists of a three-level 'functional architecture' (Antsaklis, 1999; Passino, 2005), where at each level a different function is performed. The levels are management, coordination and execution.
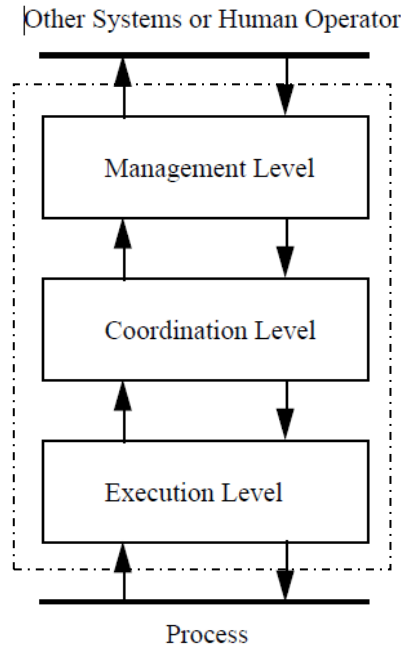
Figure 3-2. A general hierarchical structure of intelligent controllers.

The structure has a user interface at the top level and an interface to the process to be controlled at the bottom level. The functionalities are organised into three levels. The highest level, management level, involves in all management and supervision of the overall operation of the lower levels. The middle level coordinates and performs various tasks: an auto-tuning algorithm to perform tuning process of a basic fuzzy controller in the lower layer; a supervision unit to monitor the performance of the process and to guarantee the stability of the system; a memory unit to retain knowledge acquired in the tuning process; and a communication unit that maintains the communication between the controllers on the network in order to exchange information. The lowest level has a direct interface to the process and executes all the required control signals. This level includes a basic fuzzy controller in a direct action to the process in a feedback control form. Figure 3.3 shows the components and units incorporated at each level of the hierarchal structure.
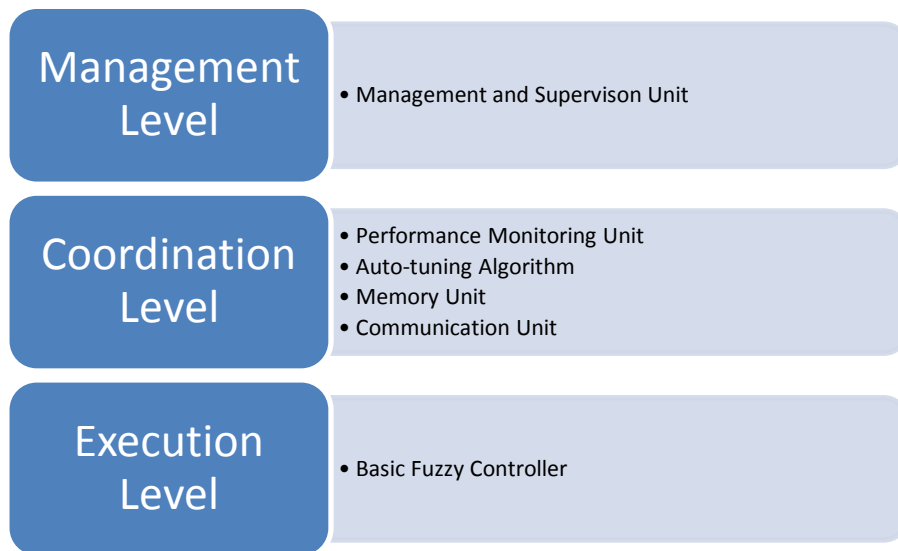
Figure 3-3: The components and units of an intelligent fuzzy controller.

3.4 Performance Assessment of the Controller

The main advantages of implementing a feedback control system are tracking the input step and reducing disturbances. The tracking is the ability of a controller to maintain the process output as close as possible to the setpoint. The disturbance rejection is the capability of a controller to reduce the effects of the changes in the process parameters or in the conditions surrounding the process.

Often, control systems are also designed to achieve several other objectives known as closed-loop specifications (Passino, 2005), which are used to measure the performance of a control system. According to a predefined and required performance, the controller should be able to achieve the desired response by adjusting its parameters. The performance is normally defined in terms of transient and steady-state responses. The transient response exists for a short time whilst the steady-state response may persist for a longer time following any change in the controller input (Dorf & Bishop, 2001).

In this thesis, to measure the performance of the fuzzy controllers; maximum percentage overshoot (M_p), rise time (t_r) and settling time (t_s) are chosen. Also, the integral of squared error (ISE) is chosen as a performance index, as it is easy to use and convenient for analytical and computational purposes (Dorf & Bishop, 2001).

3.5 Design and Development of a Basic Fuzzy Controller and an Auto-tune Algorithm

In this section, the detailed design and development of a basic fuzzy logic controller and an auto-tuning algorithm are provided. The relationships between the basic FPD+I controller gains and its performance is analysed.

3.5.1 The Basic Fuzzy Controller

The Fuzzy PD+I controller reported in (Jantzen, 2007) is shown in Figure 3.4 and it was adopted in this project. The structure of the controller is described below in some details.
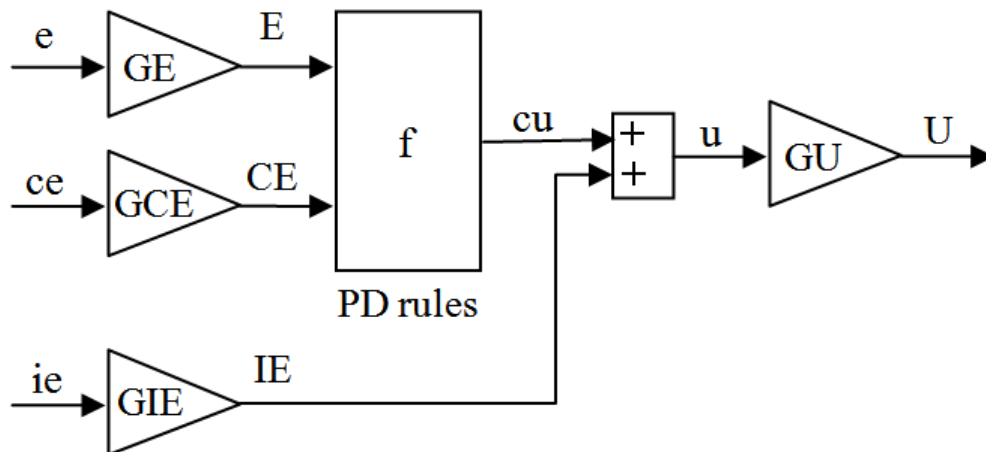
Figure 3-4: Fuzzy PD+I controller.

The controller consists of a normal FPD controller with added an integral action; therefore it is known as an FPD+I controller, henceforth it will be referred to as the basic FPD+I controller. The FPD controller action depends on the error (E) and the change of error (CE). The integral

of error (IE) is then added to the output of this controller (cu) to form the FPD+I controller. The controller has the following gains: gain of the error (G_E), gain of the change of error (G_{CE}), gain of the integral of error (G_{IE}) and the output gain (G_U). To show the effects of the gains, signals are represented by lower case symbols before the gains and upper case symbols after the gains. These gains could be fixed, or could be adjusted to achieve the best possible performance. The gains G_E, G_{CE}, G_{IE} and G_U are analogous to the proportional, derivative and integral gains in the conventional PID controller.

This controller type and the structure were selected on the basis of several factors. Firstly, the controller incorporates the three control actions, proportional, derivative and integral, those are normally required for most of the control systems. The effects of each action can be easily decreased or increased through setting the vales of the gains. Secondly, as the FPD controller acts on two inputs, E and CE; therefore, the rule-base has fewer rules in comparison to an FPID controller rule-base where it is constructed based on three inputs. The number of rules is exponentially increased with the increase in the number of controller inputs. Hence, it is computationally more efficient. Finally, the gains can be conveniently adjusted to tune the controller (Jantzen, 2007).

MATLAB (MathWorks, 2013) (v7.9 R2009b) and Simulink were used to build and simulate the controller model; this is on account of the plotting capabilities of the software and the simplicity of control characteristic computations, such as overshoot and rise time.

The basic FPD+I controller in a closed-loop feedback control system is shown in Figure 3.5. The model is primarily comprised of the controller and a process block. The error signal (e) was obtained from the difference between the setpoint (r) and the measured process output (y). The change of error signal and the integral of error signal were produced by passing the error signal through derivative and integral blocks respectively. A scope was used to show the

50

closed-loop and the open-loop responses. Figure 3.5 also shows the open-loop control system for comparison purpose.
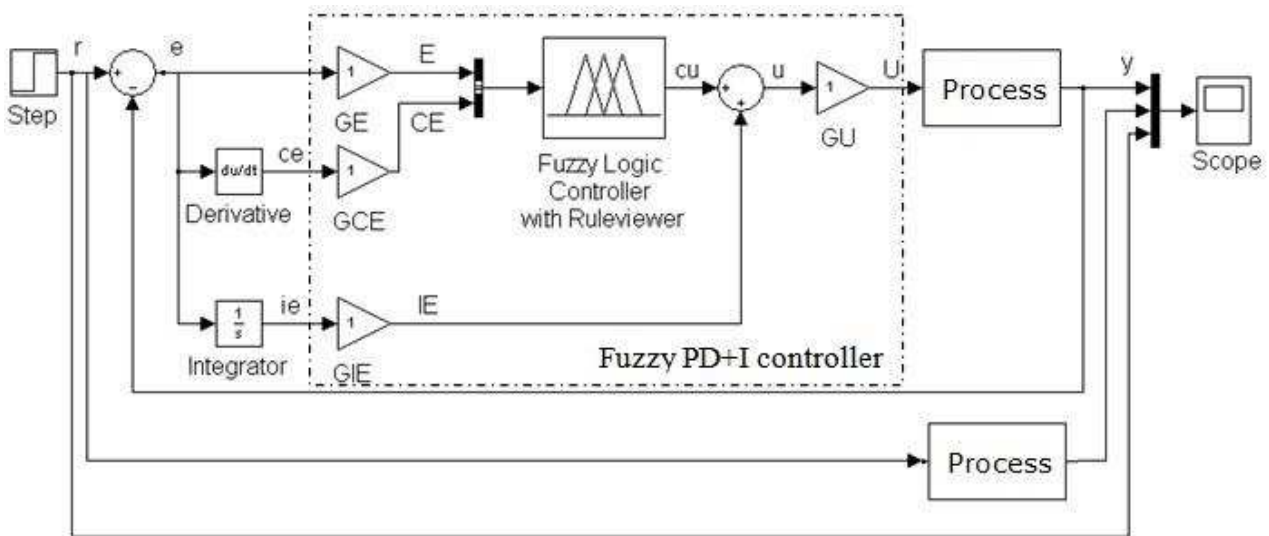
Figure 3-5: Simulation model of the closed-loop and open-loop of the basic FPD+I controller.

To represent the values of the inputs (E and CE) and the output (cu), three symmetric triangular shape and two trapezoidal shape membership functions at the extreme ends for E and CE with 50% of overlap were chosen (Jantzen, 2007; Passino & Yurkovich, 1998). Although the choice of membership function shapes is subjective, triangular shapes were chosen, because they are most popular and convenient (Altas & Sharaf, 2007). The interval of [-1, 1] was used for the universes of discourse of the input variables, while [-2, 2] was used for the output variable. The output universe of discourse is a summation of the input universes; this is to achieve an approximate conventional PD controller which makes the controller tuning process easier (Jantzen, 2007).

The linguistic descriptions of the inputs membership functions, error and change of error, are negative large (NL), negative small (NS), zero (ZE), positive small (PS) and positive large (PL); and of the output membership functions are very small (VS), small (SM), medium (MD), large (LA) and very large (VL). These are shown in Figure 3.6 and Figure 3.7 respectively.
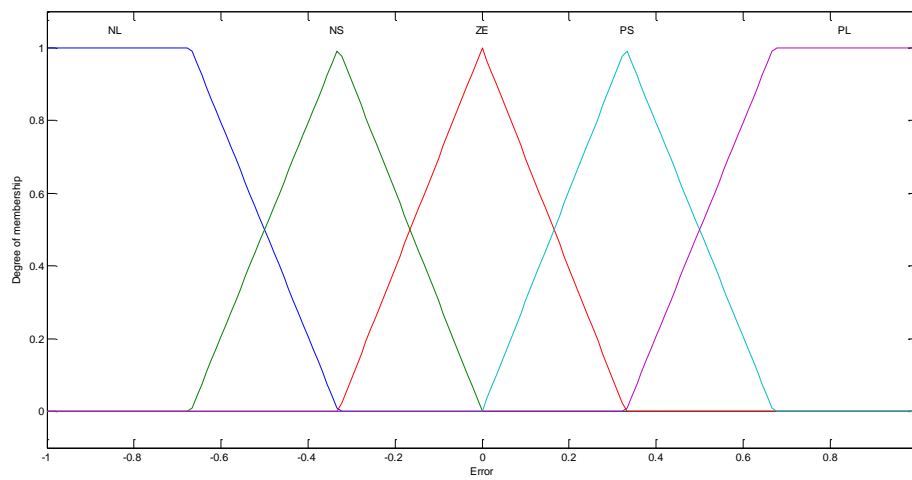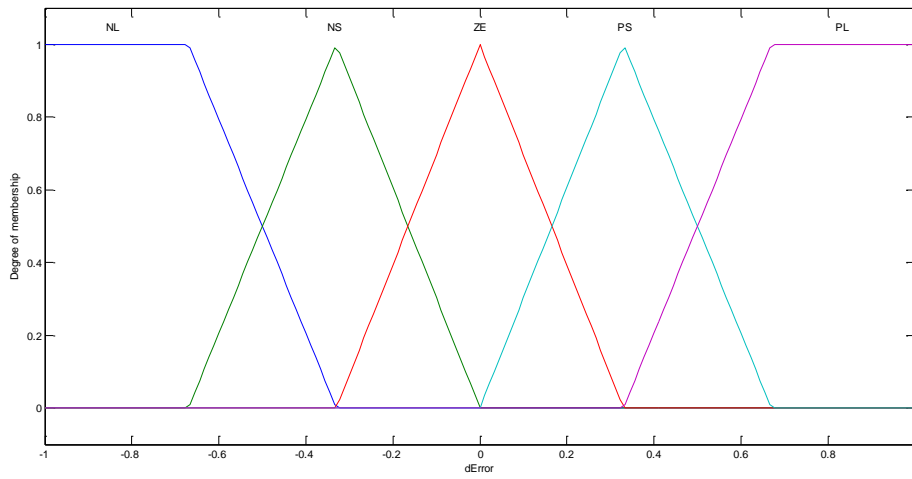
Figure 3-6: Error and change of error (dError) membership functions.
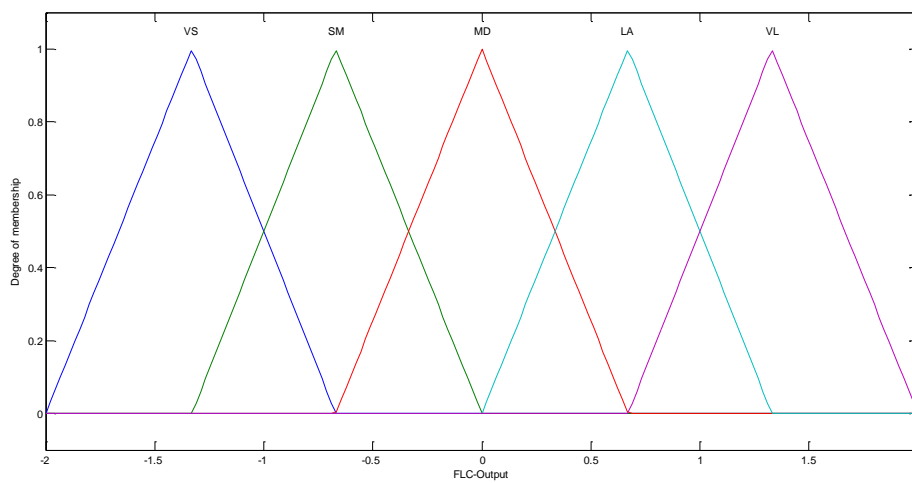
Figure 3-7: Output membership functions.

The fuzzy PD rule-base represents a mapping between the inputs and the output and it contains normal heuristic control rules of controlling a process. The rules have the following form:

If error is PL and change of error is PL, then output is VL

The above rule implies that if the error is positive large (measured output is far away from the setpoint) and the change of error is positive large, then the control signal should be very large to return back the output near the setpoint. As there are 5 linguistic variables for each input, 25 rules were created. The rules are presented in Table 3.1.

Table 3-1: Fuzzy PD controller rules

Controller Output (cu)		Change of Error (CE)				
		NL	NS	ZE	PS	PL
Error (E)	NL	VS	VS	SM	SM	MD
	NS	VS	SM	SM	MD	LA
	ZE	SM	SM	MD	LA	LA
	PS	SM	MD	LA	LA	VL
	PL	MD	LA	LA	VL	VL

The 'min' (minimum) operation was selected to implement the 'and' operator in the antecedent part of the rules, and the most popular and standard method of defuzzification process known as centre of gravity (CoG) was chosen.

To measure the performance of the controller; maximum percentage overshoot (M_p), rise time (t_r) and settling time (t_s) were used to assess the characteristics of the closed-loop system.

Finally, the **Basic FPD+I Controller Script Code**, listed in Appendix B, was developed to simulate the model shown in Figure B.2 in Appendix B, and to generate the required responses. A fixed-size (0.01 second) sampling interval was chosen for the simulation.

3.5.2 The Auto-tuning Algorithm

As discussed in Section 2.5, there are various methods to tune the parameters of a fuzzy controller in order to improve its performance. In the design of the auto-tune algorithm, adjusting the controller gains was chosen. This method is frequently used and it has been regarded as an effective way to tune fuzzy controllers (Chopra et al., 2008; Chung, Chen, & Lin, 1998; Jie et al., 2008; Mary et al., 2007; Murad et al., 2009; Victor & Dourado, 1997). First of all, the adjustments have a significant effect on the performance of fuzzy controllers (Chopra et al., 2008). Secondly, as well as there are fewer parameters to tune, in contrast to other methods where there are several parameters to tune, the tuning process is computationally more efficient. Finally, the gains can be considered as conventional controller gain parameters; therefore, they are convenient to tune and some ideas from conventional controller tuning can be borrowed (Chopra et al., 2008; Jie et al., 2008; Kanagaraj et al., 2009; Mary et al., 2007; Passino & Yurkovich, 1998; Tyan & Wang, 1995; Victor & Dourado, 1997).

After applying the basic FPD+I controller to several second order transfer functions, this will be discussed in further details in Section 3.6.1, and based on the observation of the relationship between the adjustments of the controller gains and the controller performance, an approach was concluded to devise an algorithm to automatically tune the controller gains.

As mentioned in Section 3.4, the maximum percentage overshoot (M_p) and the rise time (t_r) were chosen as performance metrics to measure the performance of the basic FPD+I controller. It is important to know the relationship between these metrics and the gains. This will indicate the effects of these gains on the performance of the controller. Therefore, the relationship between G_{IE} and G_U on one side and the maximum percentage overshoot and the rise time on the other side were obtained. The relationship is a 3D plot, where the gains were automatically and simultaneously adjusted in their defined intervals, [0, 1] for G_{IE} and [1, 50] for G_U, with a step size of (0.1). For each adjustment, the maximum percentage overshoot and

the rise time were calculated. The **Mp and tr 3D Plot Script Code** listed in Appendix B, was written to simulate the basic FPD+I controller for several second order transfer functions. For example plots of one of the second order transfer functions, Case 1 as described in Section 3.6.1, are shown in Figure 3.8 and Figure 3.9. The plots of other cases are shown in Figure C-1 - Figure C-8 in Appendix C.
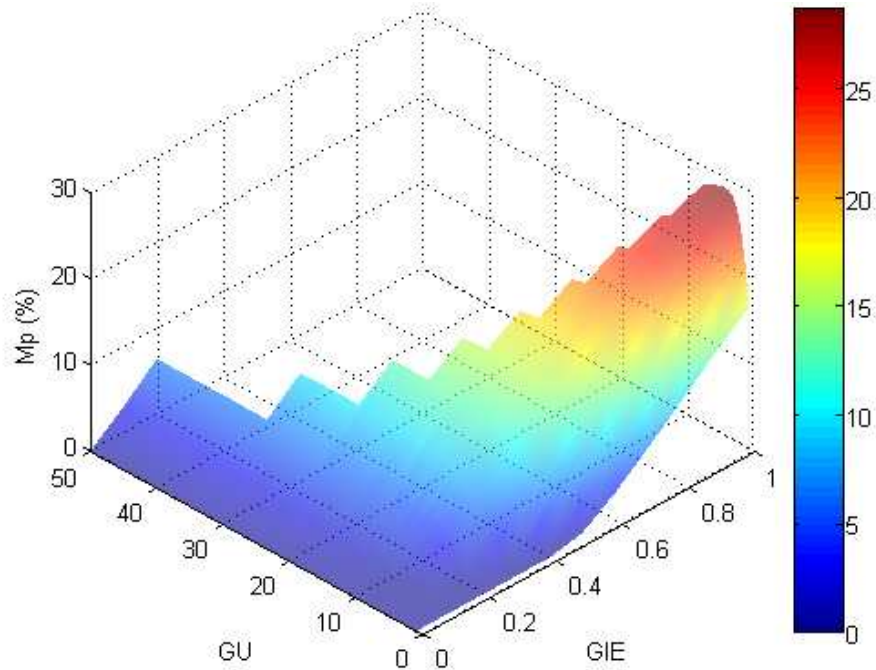
Figure 3-8: The 3D plot between M_p and the basic FPD+I gains (G$_{IE}$, G$_U$) for Case 1.
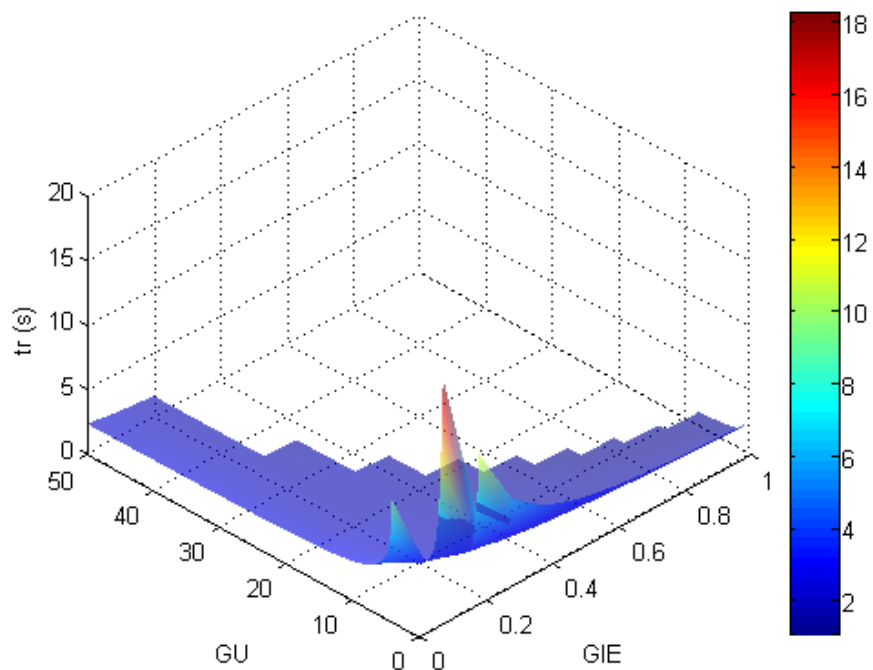
Figure 3-9: The 3D plot between t_r and the basic FPD+I gains (G$_{IE}$, G$_U$) for Case 1.

The graphs indicate that not all the values of M_p and t_r are valid as the values of the gains point to unstable systems. As can be seen form the graphs, decreasing G_{IE} and increasing G_U affect both M_p and t_r, so that both are decreasing, which improve the performance of the controller. This trend is also noticed for all other cases.

To show more clearly the effects of these changes, the transient response of the same case, Case 1, was taken for different values of G_{IE} and G_U. The value of G_{IE} was decreased from 1 to 0 in a fixed step size of 0.1, and the value of G_U was increased from 1 to 31 in a fixed step size of 3. Then for each combination of these values, G_{IE} and G_U, a transient response of a step input was plotted and denoted as iteration. Finally, the transient responses of all iterations were combined together to form a 3D plot as shown in Figure 3.10. The plots of other cases are shown in Figure C-9 - Figure C-12 in Appendix C. The **Transient Response 3D Plot Script Code** listed in Appendix B was used to obtain the plots.
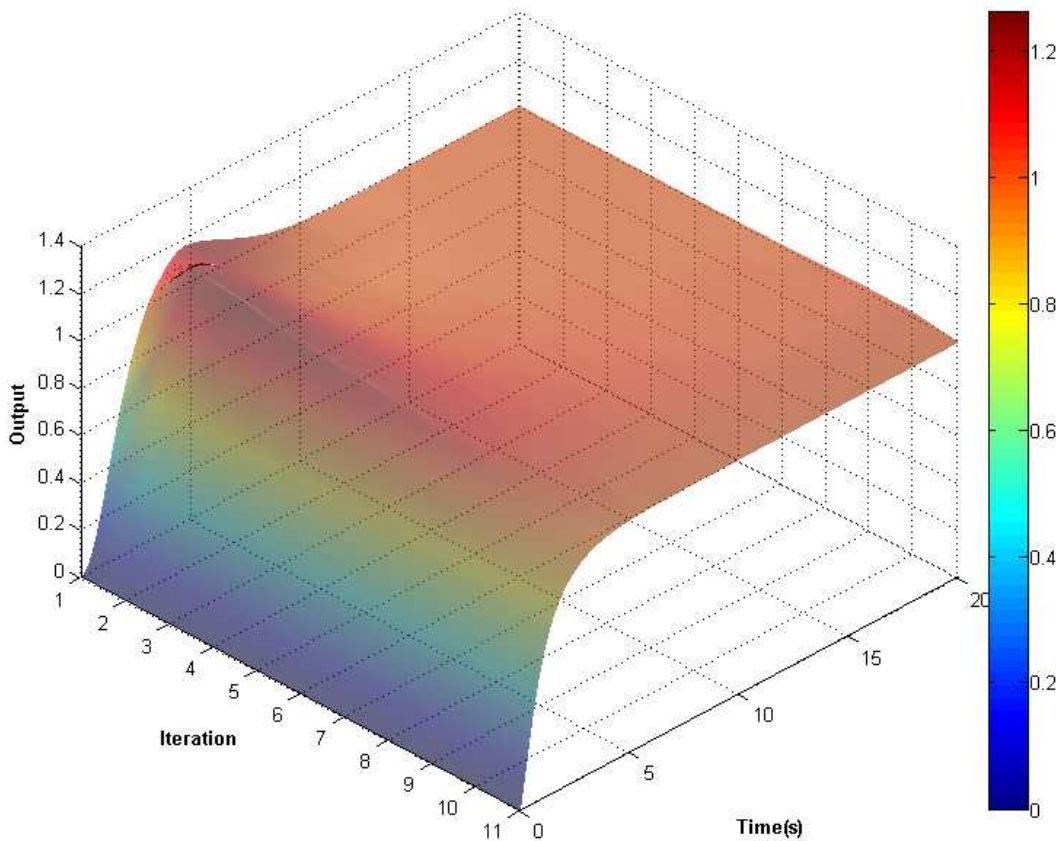
Figure 3-10: The transient response of Case 1 for different values of G_{IE} and G_U.

Figure 3.10 shows the effects of decreasing G_{IE} and increasing G_U. The graphs indicate that not all the iterations are valid, Figure C-10 - Figure C-12 in Appendix C, as the values of the gains point to unstable systems.

It is apparent from the graphs that a stable system with a zero overshoot response is achieved when the value of G_U is high and at the same time the value of G_{IE} is too low. These results were led to formulate the relationship between G_U and G_{IE} and M_p as follows:

$$G_U = M_p \tag{1}$$

$$G_{IE} = 1 / (2 * M_p) \tag{2}$$

Empirical Equations (1) and (2) were derived from the analysis of the results of the basic FPD+I controller, where extensive simulation-based tests were conducted to determine the relationship between the controller gains and the overshoot.

The structure of the auto-tuning algorithm (Saeed & Mehrdadi, 2012a) was comprised of two layers, a basic-layer that incorporates the basic FPD+I controller and an upper-layer which performs the auto-tuning process. Additionally, the algorithm is able to monitor the performance of the system and to maintain its stability.

The details of the algorithm can be summarised in the following steps:

Step 1: To identify a process, a closed-loop test on the system is performed by applying the basic FPD+I controller. The controller gains are set to their default values of 1, and the output is bounded so that the overshoot is not allowed to exceed 100%, where the system could become unstable.

Step 2: The overshoot is measured.

57

Step 3: If the overshoot is higher than 1%, then the values of G_U and G_{IE} are calculated from Equations (1) and (2). Then the controller is re-applied with the new values of G_U and G_{IE}.

Step 4: When the overshoot is less than 1%, which is normally accepted, then the gains, G_U and G_{IE}, are kept unchanged.

Step 5: To reduce the rise time, the value of G_{CE} is decreased in a step size of 0.1.

Step 6: If the system performance is not satisfactory, for instance if there is any sustained steady state error, the value of G_{IE} is increased in a step size equal to twice the value of the initial value of GIE. For each subsequent step this step size is doubled again.

The last two steps are performed iteratively. The value of G_E is kept unchanged during the whole tuning process.

The maximum percentage overshoot (M_p) and the integral of squared of (ISE) shown in Equation (3) were chosen to measure the performance of the controller. The algorithm was able to determine the values of the gains by minimising the value of ISE.

$$Integral\ sequare\ of\ error\ (ISE) = \int_0^\infty (e)^2\, dt \qquad (3)$$

The flowchart shown in Figure 3.11 illustrates the operation of the auto-tuning algorithm. The **Auto-tuning Algorithm Script Code** of the algorithm is listed in Appendix B.
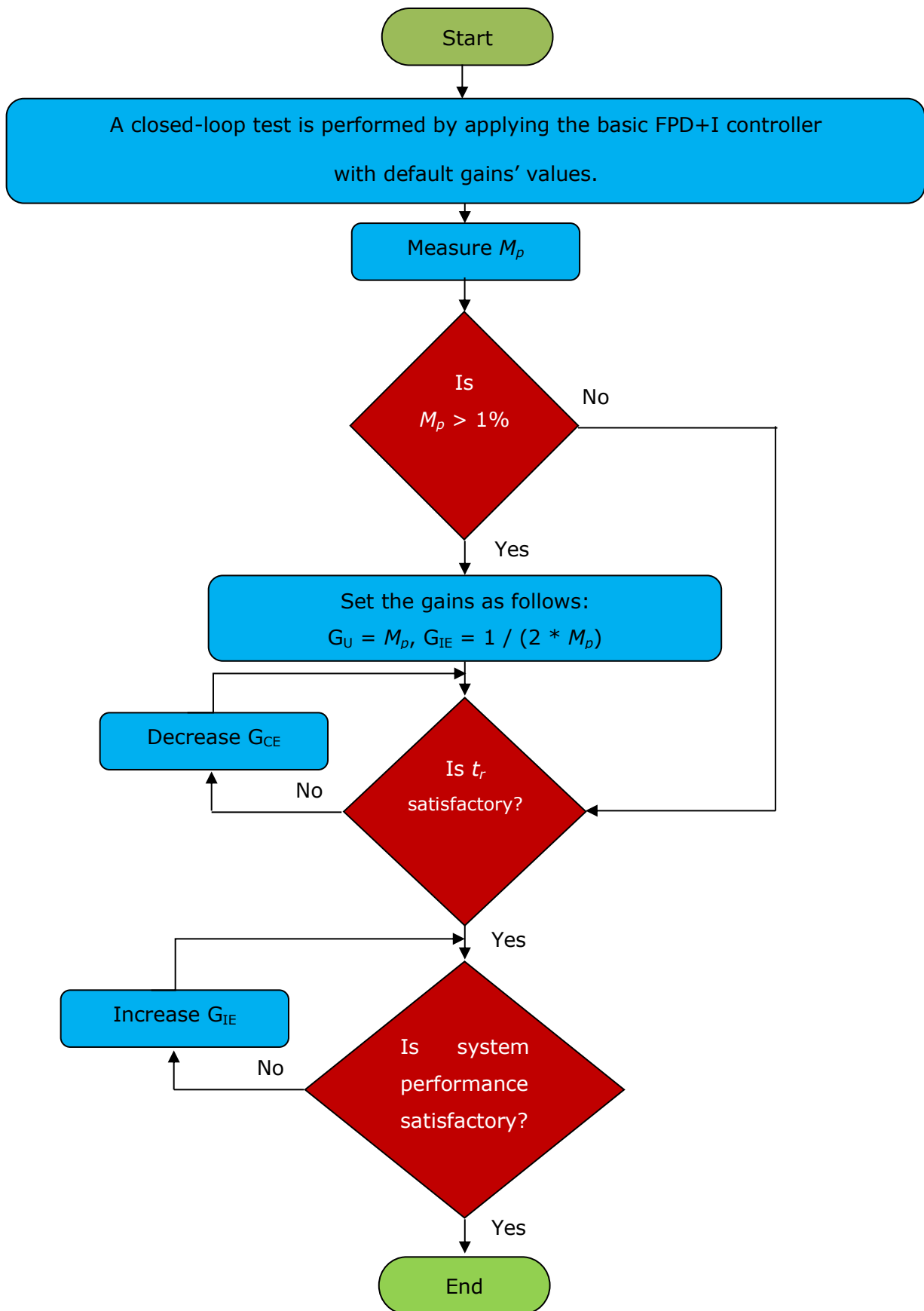
Figure 3-11: The auto-tuning algorithm flowchart.

3.6 Performance Analysis and Evaluation

In order to evaluate and analyse the performance of the basic FPD+I controller and the auto-tuning algorithm for a wide range of systems, standard transfer functions of several second order systems with different characteristics were chosen. At this stage, only some mathematical analysis is shown and at later stages simulation-based analysis and experimental investigations will be presented.

3.6.1 Second Order Systems

Many real-time applications exhibit oscillation and overshoot in their step responses. These characteristics can be modelled using a second order system (Haugen, 2009; Rowell, 2004). Furthermore, this will help to understand the response of higher order systems. Consider the standard second order transfer function (Dorf & Bishop, 2001; Shen & Chiang, 2004) in Equation (4).

$$G_s = \frac{K \, \omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{4}$$

Where s is the Laplace transform, K is the gain, ζ is the damping ratio and ω_n is the natural frequency. The system has different responses depending on the location of its poles. The poles of Equation (4) are the roots of the denominator and can be determined as:

$$p1, p2 = -\zeta\omega_n \pm \sqrt{(\zeta^2 - 1)} \; \omega_n \tag{5}$$

The value of ζ determines whether the poles are real or complex conjugate. From Equation (4), suppose $K = 1$ and $\omega_n = 1$. Depending on the value of ζ, there are five distinct cases as the following:

- If $\zeta \geq 1$, the poles are real:

  - $\zeta = 1$, a critically damped system, denoted as Case 1.

  - $\zeta = 1.5$, an overdamped system, denoted as Case 2.

- If $0 \leq \zeta < 1$, the poles are complex conjugates:

  - $\zeta = 0.5$, an underdamped system, denoted as Case 3.

  - $\zeta = 0$, an undamped (marginally stable) system, denoted as Case 4.

- If $\zeta < 0$, the poles are a combination of a real and a complex conjugate:

  - $\zeta = -0.1$, an unstable system, denoted as Case 5.

According to the values of ζ, different system transfer functions are shown in Table 3.2 and the step responses of these systems are shown in Figure 3.12.

Table 3-2: Standard second order transfer functions

Case	ζ	Transfer Function
1	1	$\dfrac{1}{S^2 + 2S + 1}$
2	1.5	$\dfrac{1}{S^2 + 3S + 1}$
3	0.5	$\dfrac{1}{S^2 + S + 1}$
4	0	$\dfrac{1}{S^2 + 1}$
5	-0.1	$\dfrac{1}{S^2 - 0.2S + 1}$
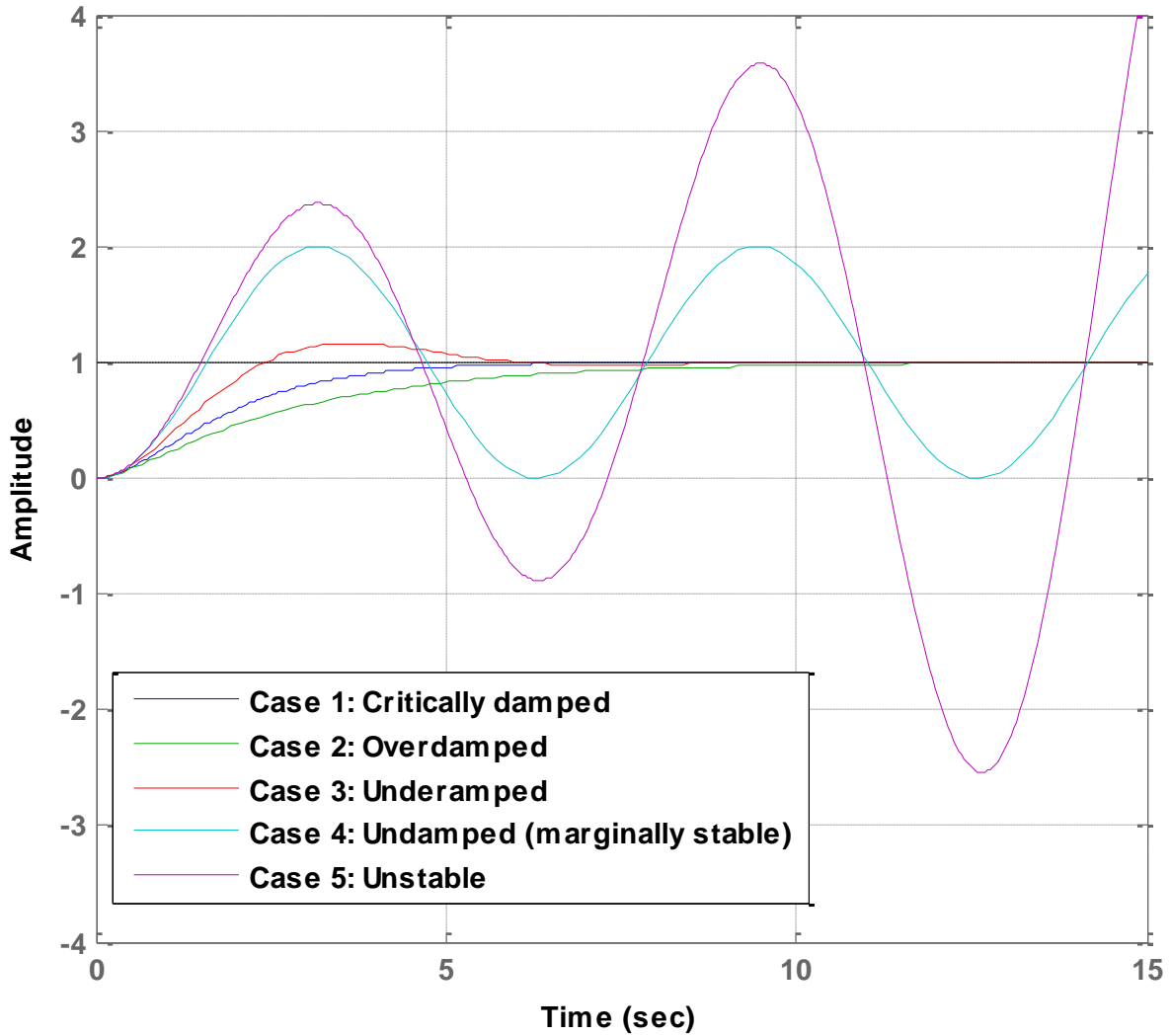
Figure 3-12: Step response of various standard second order systems.

Throughout this thesis, the above five cases will be used as the plant of all simulation-based analysis and experimental investigations.

3.6.2 Mathematical Evaluation

In order to mathematically evaluate the effects of the gains found by Equations (1) and (2), the closed-loop transfer function of the feedback control system shown in Figure 3.5 is obtained and then the values of G_U and G_{IE} are replaced. The closed-loop transfer function is:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{GF(s) * G(s)}{1 + \ GF(s) * G(s)} \qquad (6)$$

Where $T(s), Y(s)$ and $R(s)$ are the Laplace transform of the closed-loop transfer function, the output signal and the input signal respectively. $G(s)$ is the Laplace transform of the transfer function of the process under control, and $GF(s)$ the Laplace transform of the transfer function of the basic FPD+I controller. By refereeing to Figure 3.5, $GF(s)$ can be constructed as follows:

$$U(t) = \left[f\left(GE * e(t), GCE * \frac{de(t)}{dt} \right) + GIE \int e(t)dt \right] * GU \qquad (7)$$

The fuzzy PD control output, function f, represents a fuzzy relationship of the error and the change of error. The relationship is called control surface and it is obtained and shown in Figure 3.13.
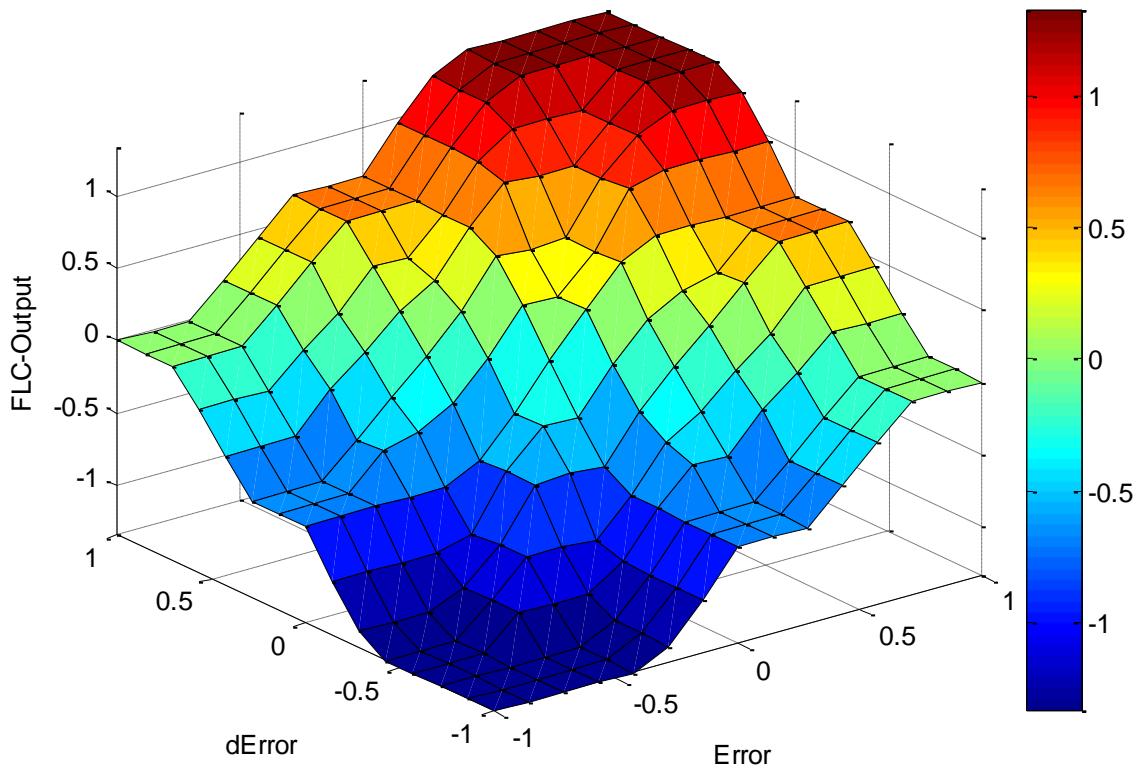
Figure 3-13: The control surface of the basic FPD+I controller.

From Figure 3.13, the maximum output (FLC-output) is 1.34. This value could be represented as an approximate summation of the input values, Error and dError. Therefore, the output

value, 1.34, could be achieved by adding each input value multiplied by the value of 0.67; thus:

$$f\left(GE * e(t), GCE * \frac{de(t)}{dt}\right) \cong 0.67 * GE * e(t) + 0.67 * GCE * \frac{de(t)}{dt} \quad (8)$$

By substituting Equation (8) in Equation (7):

$$U(t) = \left[0.67 * GE * e(t) + 0.67 * GCE * \frac{de(t)}{dt} + GIE \int e(t)dt\right] * GU \quad (9)$$

Taking the Laplace transform of Equation (9)

$$U(s) = \left[0.67 * GE * E(s) + 0.67 * GCE * s * E(s) + GIE \frac{1}{s} * E(s)\right] * GU \quad (10)$$

Then, from solving Equation (10), the transfer function of the basic FPD+I controller can be calculated as:

$$GF(s) = \frac{U(s)}{E(s)} = \left[0.67 * GE + 0.67 * GCE * s + GIE \frac{1}{s}\right] * GU$$

$$GF(s) = \frac{(0.67 * GCE * S^2 + 0.67 * GE * S + GIE) * GU}{S} \quad (11)$$

Suppose $G(s)$ is one of the systems presented in Table 3.2, namely Case 1:

$$G(s) = \frac{1}{S^2 + 2S + 1} \quad (12)$$

By substituting Equations (11) and (12) in Equation (6)

$$T(s) = \frac{Y(s)}{R(s)} = \frac{\dfrac{(\,0.67 * GCE * S^2 + \ 0.67 * GE * S + \ GIE) \ * \ GU}{S} * \dfrac{1}{S^2 + \ 2S + \ 1}}{1 + \dfrac{(\,0.67 * GCE * S^2 + \ 0.67 * \ GE * S + \ GIE) \ * \ GU}{S} * \dfrac{1}{S^2 + \ 2S + \ 1}}$$

$$T(s) = \frac{Y(s)}{R(s)} = \frac{(\,0.67 * GCE * GU)S^2 + \ (\,0.67 * GE * GU)S + \ GIE * GU}{S^3 + (2 + \ 0.67 * GCE * GU)S^2 + \ (1 + \ 0.67 * GE * GU)S + \ GIE * GU} \qquad (13)$$

If all the gains are set to their default values of 1, this is when the basic FPD+I controller is normally applied, then the closed loop transfer function becomes:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{0.67\,S^2 + \ 0.67 * S + \ 1}{S^3 + 2.67\,S^2 + \ 1.67\,S + \ 1} \qquad (14)$$

A step response of the transfer function in Equation (14) is shown in Figure 3.14. The response is identical to that produced by applying the basic FPD+I controller in the simulation study mentioned in Section 5.1.1. The closed-loop step response of Case 1 using the basic FPD+I controller is presented in Figure C-13 in Appendix C.
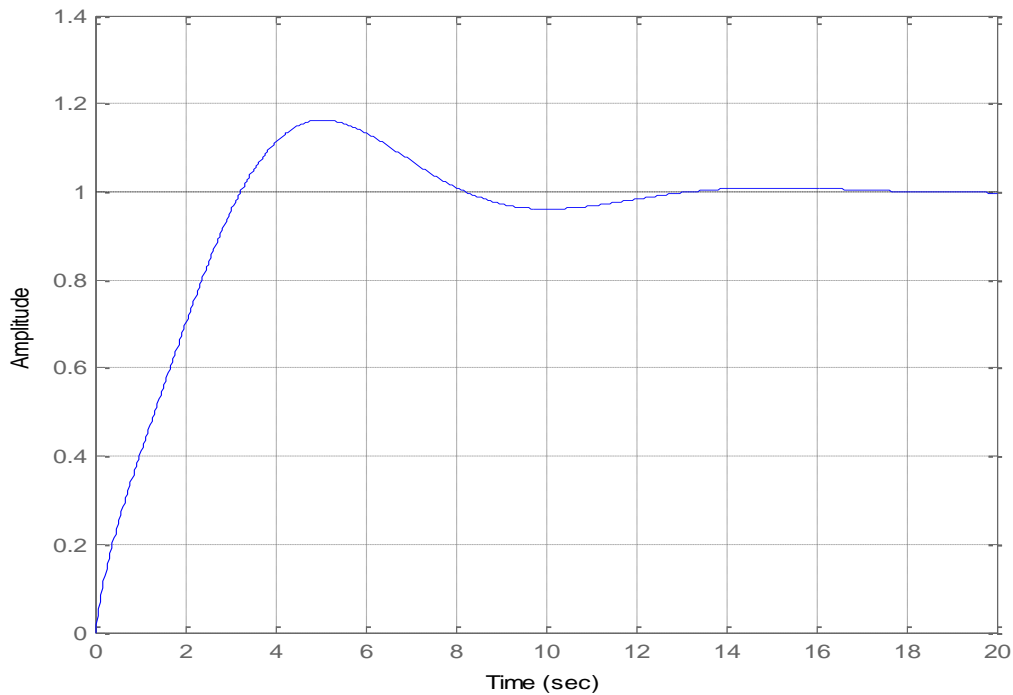
Figure 3-14: Step response of the transfer function in Equation (14).

65

As mentioned in the steps of the auto-tuning algorithm, the gains G_E and G_{CE} are kept to their default values of 1, while G_{IE} is decreased and G_U is increased. Assuming that both gains are set to 0.1 and 50 respectively, from the findings of Section 3.5.2, then Equation (13) becomes as:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{3.35\,S^2 + 3.35\,S + 5}{S^3 + 5.35\,S^2 + 4.35\,S + 5} \qquad (15)$$

The step response of this transfer function, Equation (15), is shown in Figure 3.15.
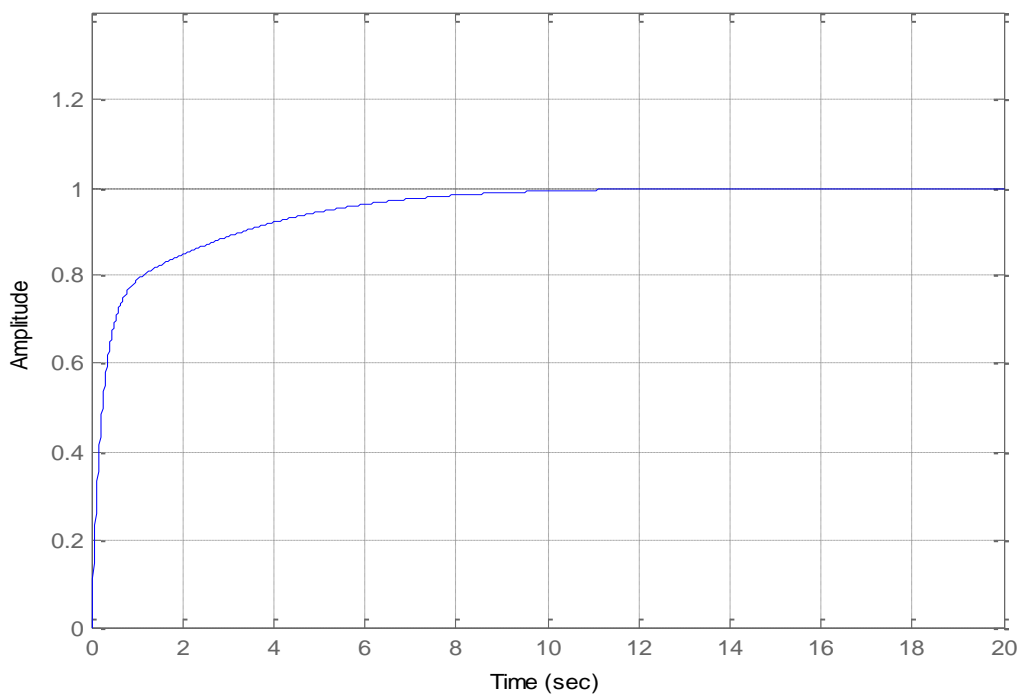
Figure 3-15: Step response of the transfer function in Equation (15).

As it can be seen, the new values of G_{IE} and G_U have drastically eliminated the overshoot. These results show the effectiveness of the algorithm and the feasibility of the determined values of the controller gains by Equations (1) and (2), as both methods, the mathematical analysis and the simulation-based studies produced identical results.

3.7 Summary

This chapter has illustrated the main attributes of an intelligent controller. It has also presented an architecture for the wireless intelligent fuzzy controller network system and the main components as well as the functionality of each part are described.

Furthermore, using MATLAB, the design of a basic FPD+I controller has been provided in more details. The relationships between the controller gains, G_{IE} and G_U, and the controller performance metric in terms of M_p and t_r analysed and established. Empirical equations have been derived to calculate the initial values of the gains. Some mathematical analysis has been presented in order to assess the effectiveness of the new found gains' values.

Based on the analysis of the basic FPD+I controller performance, a systematic auto-tuning algorithm has been devised.

In the next chapter, the implementation of the wireless intelligent fuzzy controller network and its components will be presented.

CHAPTER 4

Implementation of the Wireless Intelligent Fuzzy Controller Network

In this chapter, a brief introduction to the development platform used for the implementation of the wireless intelligent fuzzy controller network is given. Also, the structure of the implemented systems is provided. To apply the control algorithm to a real hardware, some standard second order transfer functions are realised and constructed using various electronic components.

4.1 The Development Platform

Designing a wireless intelligent controller network encompasses many stages including developing an intelligent control algorithm, establishing and maintaining the communication link between the controllers and monitoring the stability of the system. Therefore choosing the right development platform is essential. National Instruments (NI) LabVIEW 2011 (NI, 2013b), a graphical software development environment for creating applications that interact with real-world data or signals, was chosen to implement the proposed design in Section 3.2. This was on account of some essential and useful features provided in the LabVIEW package. First of all, it provides an integrated software and hardware platform that simplifies development of any system that needs measurement and control which hardware interfacing is made very quickly and easy. Secondly, the easy-to-use graphical development environment makes it the ideal choice for developing complex algorithms by providing various tools for debugging and direct

viewing of the results (Bitter, Mohiuddin, & Nawrocki, 2007; Ponce-Cruz & Ramírez-Figueroa, 2010; Saeed & Mehrdadi, 2012b).

A LabVIEW program is comprised of two parts, 'Front Panel' and 'Block Diagram', and as the 'Front Panel', also known as the user interface, appearance and operation intimate physical instruments it is called virtual instrument (VI).

The 'Front Panel' is used as a user interaction and display unit and accommodates controls, such as knobs, push buttons and other input mechanisms; and indicators such as graphs, LEDs and other output display units. The 'Block Diagram' contains the program source code constructed using LabVIEW's graphical programming, G language, in which programming functionalities are used in the form of icons instead of lines of text. The elementary parts of the 'Block Diagram' consist of functions and known as VIs or just functions. For example the square function shown in Figure 4.1 is used to compute the square of the input x.

Figure 4-1: The LabVIEW square function.

4.2 Overall Structure

To implement the proposed architecture, PC-based controllers were chosen to simplify the coding and debugging stages as they tend to be more straightforward than microcontroller-based controllers. At a later stage the design can be easily coded and deployed on microcontrollers.

A network of four wireless-enabled PCs was established through a dedicated wireless router to ensure there is no other network traffic to disturb the communication of the PCs; the overall structure is shown in Figure 4.2. The wireless is the standard wireless local area network (WLAN) Wi-Fi/ IEEE standard 802.11 (Makaya & Pierre, 2012), and based on the requirements of the application the wireless network could be replaced by any other wireless technologies, such as ZigBee/ IEEE standard 802.15.4 (Gislason, 2008).

PC1　　　　　**PC2**　　　　　**PC3**　　　　　**PC4**

Figure 4-2: The architecture of wireless fuzzy logic controllers[*].

In order to connect the PCs to real-time control systems, each PC was equipped with a NI PCI-6025E data acquisition (DAQ) card (NI, 2013c) that facilities interfacing to real-time control systems. This will be covered in more detail in Section 4.8.

4.3 The Basic Fuzzy Controller

The basic FPD+I controller was redesigned by using LabVIEW. The **LabVIEW Control Design and Simulation module** was used to design and simulate controller components, and the **LabVIEW Fuzzy System Designer** was used to design the fuzzy parts of the controller, such as input/output membership functions and the rules.

---

[*]*Permission has been obtained from (Berezyuk, 2013) to use the icons appeared in the figure.*

The main VI was comprised of two parts, 'Front Panel' and 'Block Diagram'. The 'Front Panel' is the user interface to the controller, while the 'Block Diagram' contains all the codes in the form of VI and functions composing the controller. As various LabVIEW VIs will be utilised throughout this chapter, the reader is referred to Appendix E, where brief descriptions of the used VIs are provided.

Figure 4.3, shows the user interface where the main sections are denoted by numbers for the purpose of illustration. The process under control is any standard second order system, denoted by 1, where its parameters are set through section 2. The open-loop step response is displayed by the graph denoted by 3. The open-loop time response characteristics, such as rise time, percentage overshoot and settling time are shown in section 4. The basic FPD+I controller, denoted by 5, is in a direct control-loop feedback. The controller has four gains: G_E, G_{CE}, G_{IE} and G_U where their respective values are set via the gain values section, denoted by 6. A step input, section 7, changing from 0 to 10 with an increment of 1 is applied to the system. The closed-loop step response is displayed on the graph denoted by 8 and the closed-loop time response characteristics, such as rise time, percentage overshoot and integral of squared error are shown in section 9. External disturbances could be introduced at the input and at the output of the process to determine the stability and robustness of the control system. The magnitude, type and time when the disturbances are applied can also be set through section 10.
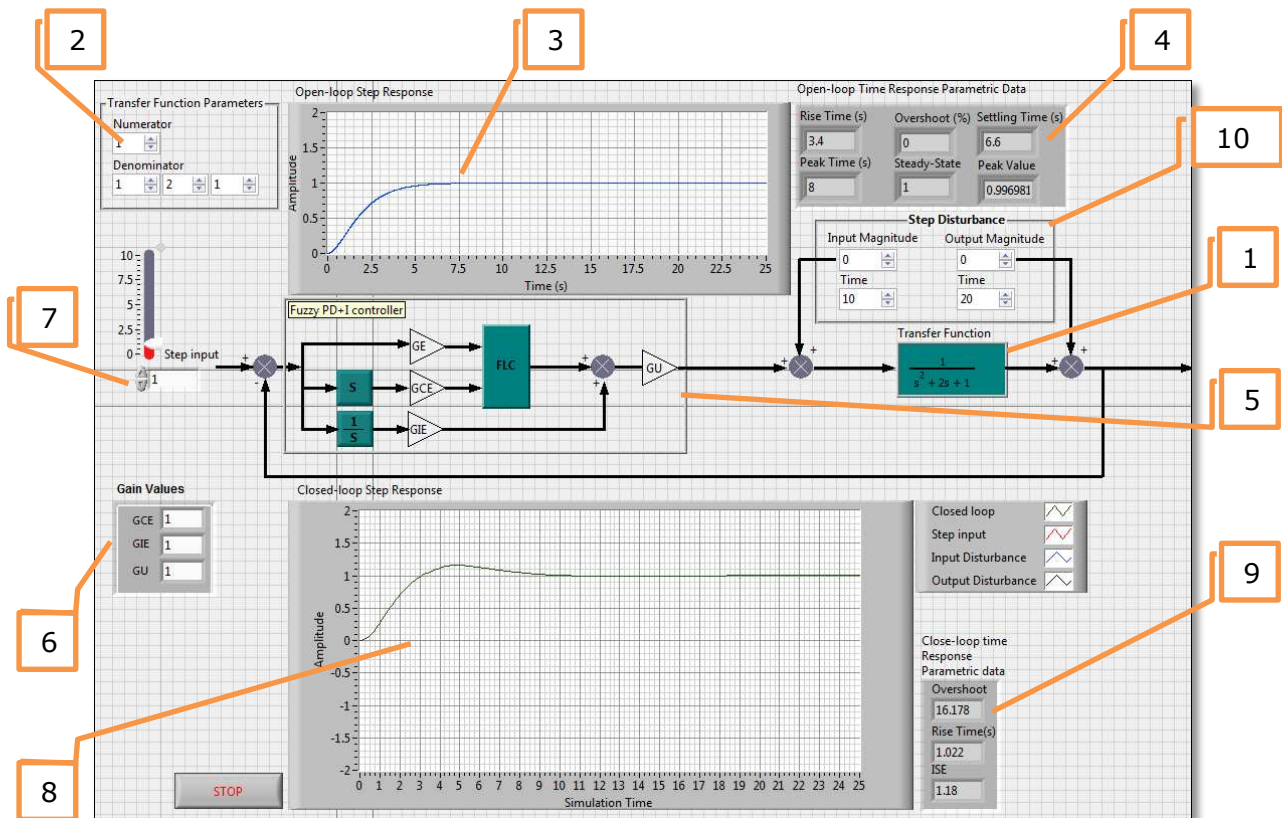
Figure 4-3: The front panel of the basic FPD+I controller VI.

The block diagram part, shown in Figure 4.4, accommodates essential VIs of the basic FPD+I controller. The major components are located in a nest of two loops, a **While Loop** to repeat the execution of the controller until a stop command is received and a **Control and Simulation Loop** which contains the controller structure, **FL Fuzzy Controller VI** and **Transfer Function VI**.

The fuzzy controller settings were loaded to the fuzzy controller through **FL Load Fuzzy System VI** and the **Transfer Function VI** was configured via **CD Construct Transfer Function Model VI**. The reader is referred to Appendix E where the functionalities of these VIs are illustrated.

The **CD Draw Transfer Function Equation VI** was used to draw the transfer function equation on the user interface, while **CD Step Response** and **CD Parametric Time**

72

Response VIs were used to obtain the open-loop step response and the open-loop time response parametric data respectively.
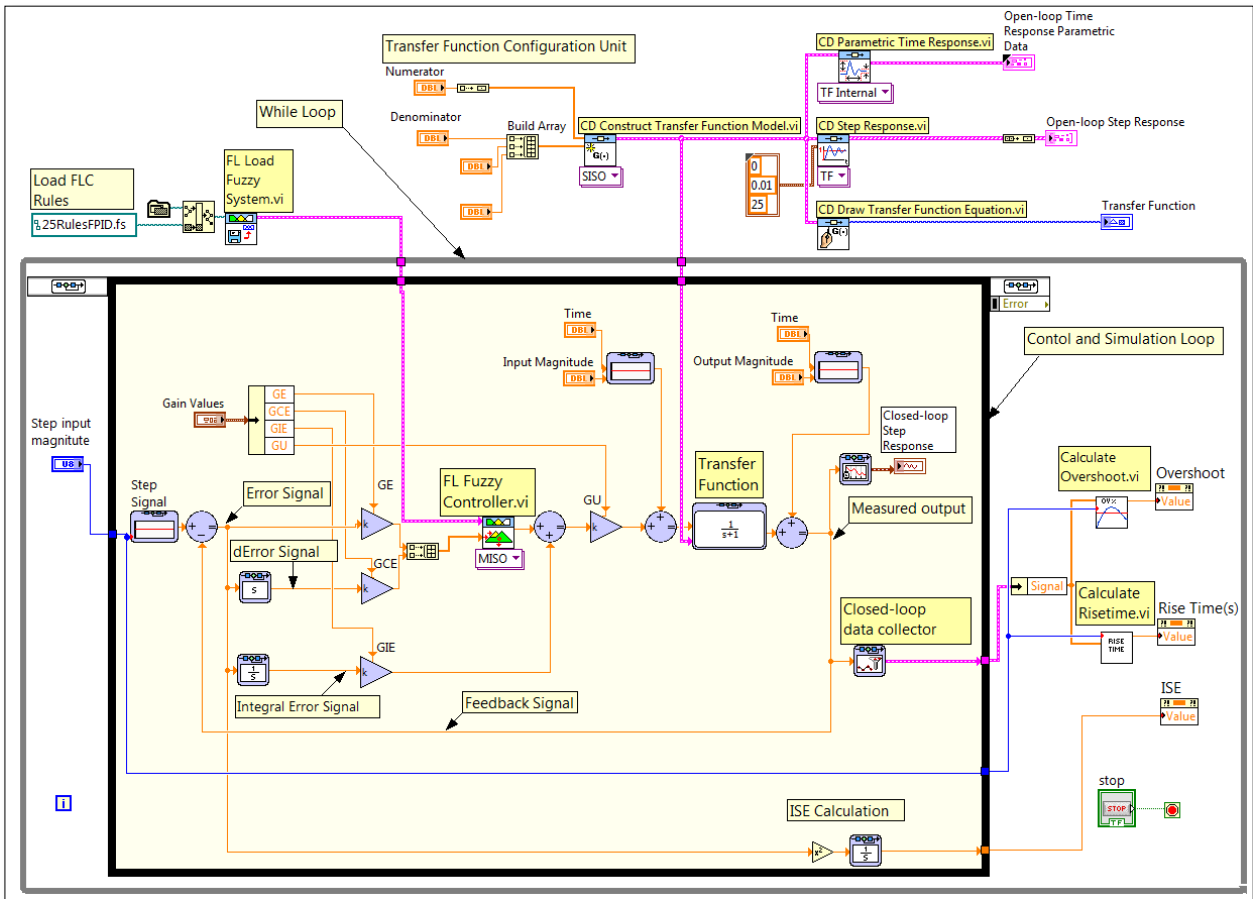
Figure 4-4: The block diagram of the basic FPD+I controller VI.

Figure 4.5 shows the implementation of the FPD+I controller where the fuzzy controller settings are loaded to the fuzzy controller, **FL Fuzzy Controller VI**, through **FL Load Fuzzy System VI**.
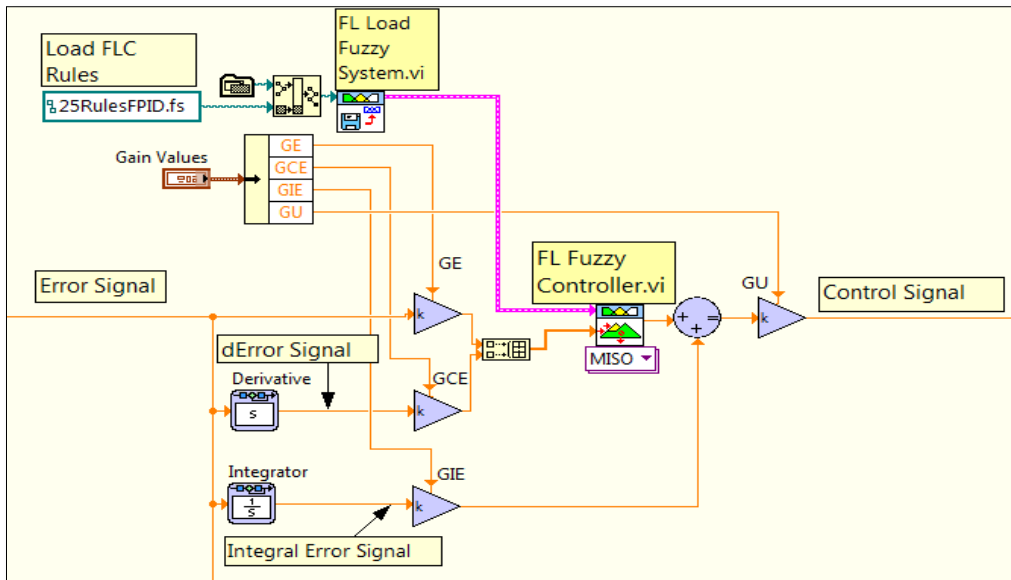
Figure 4-5: Implementing the basic FPD+I Controller.

4.3.1 Creating Input and Output Linguistic Variables and the Rule-Base

The basic FPD+I controller has two input variables, E and CE and one output variable (cu). In this section variables are renamed to Error, dError and FLC-Output respectively. The **LabVIEW Fuzzy System Designer** was used to design the fuzzy controller input/output membership functions and the rules.

To represent the values of the inputs (Error and dError) and the output (FLC-Output), the membership functions shown in Figure 4.6 are used. The interval of [-1, 1] was used for the universe of discourses of the input variables, while [-2, 2] was used for the output variable.

The linguistic descriptions of the input membership functions were used are negative large (NL), negative small (NS), zero (ZE), positive small (PS) and positive large (PL), while very small (VS), small (SM), medium (MD), large (LA) and very large (VL) were used for the output membership function.

74

As there are 5 linguistic variables for each input, 25 rules were created, Figure 4.7 shows the rules. The 'min' (minimum) operation was selected to implement the 'and' in the antecedent part of the rules, and the centre of gravity (CoG) was chosen as the defuzzification method.
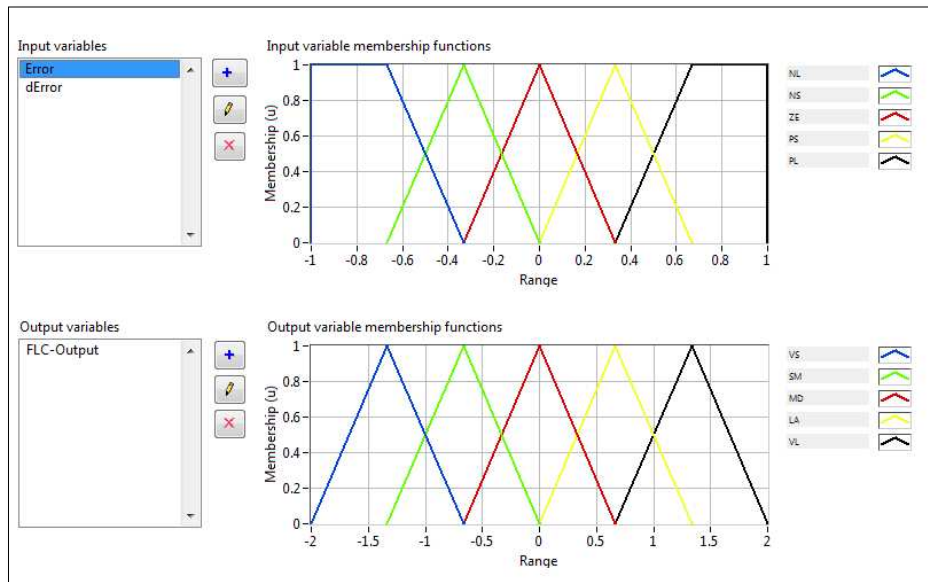
Figure 4-6: Input and output membership functions of the basic FPD+I Controller.
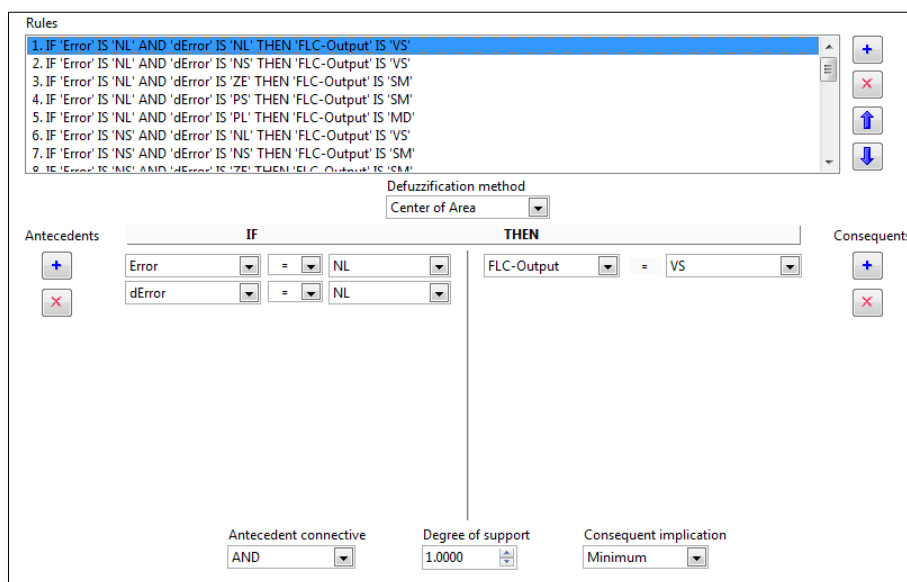
Figure 4-7: Samples of the basic FPD+I controller rules.

4.3.2   *Performance Measurement*

To measure the performance of the controller, the maximum percentage overshoot (M_p), the rise time (t_r) and the integral of squared error (ISE), discussed in Section 3.4, were chosen. As these are not included in the LabVIEW library the necessary Sub VIs were developed to calculate these parameters.

Figure 4.8 shows the calculation of the percentage overshoot, where the output of the system was continuously collected in an array variable, **Array Max & Min VI**, then from the array the maximum value was obtained which is the overshoot value, and then Percentage Overshoot was calculated as follows:

$$\text{Percentage Overshoot} = \frac{Maximum\ Value\ of\ Overshoot - Step\ Input\ Final\ Value}{Step\ Input\ Final\ Value} * 100 \quad (16)$$
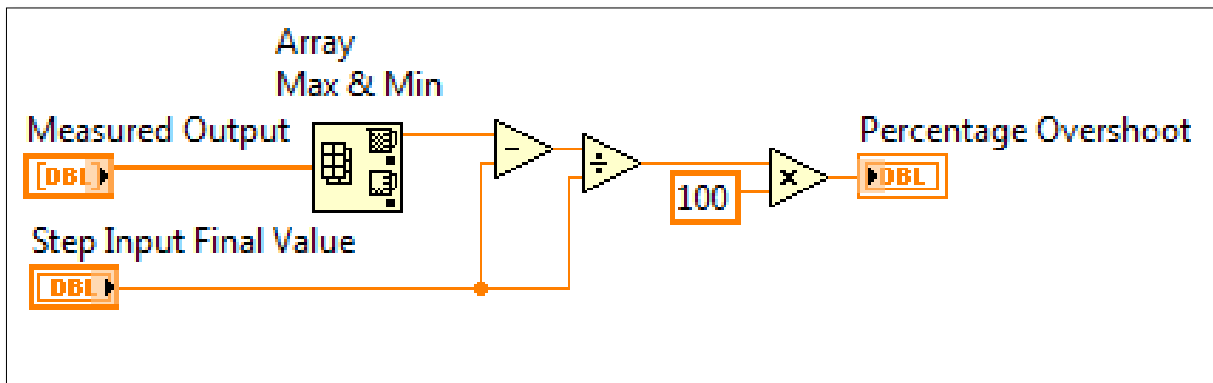
Figure 4-8: Overshoot calculation sub VI.

To calculate the rise time, normally, the time taken by the system output to rise from 10% to 90% of the final value of the output is measured (Dorf & Bishop, 2001). This is illustrated in Figure 4.9. The output value was measured at two points, 10% and 90%, through the **Threshold 1D Array VI** and then the rise time was calculated as follows:

$$\text{Rise Time} = (90\% \text{ of the Measured Output} - 10\% \text{ of the Measured Output}) * \text{Sampling Time} \qquad (17)$$
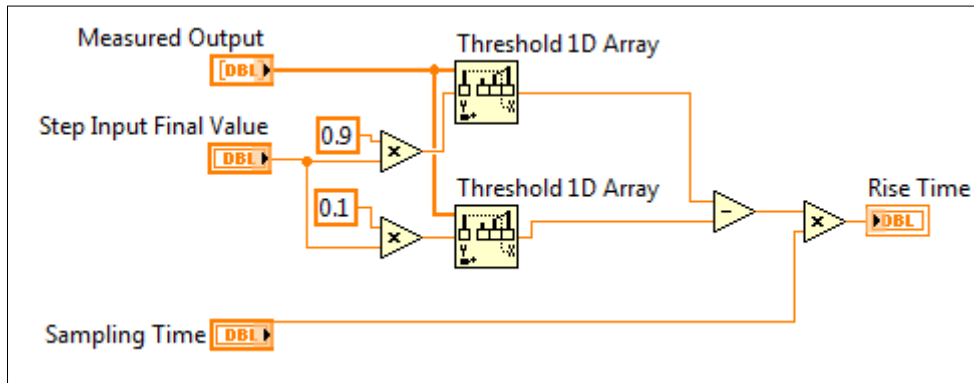
Figure 4-9: Rise time calculation sub VI.

Finally, the integral of squared error was calculated by integrating the squared error signal as it is shown in Figure 4.10.
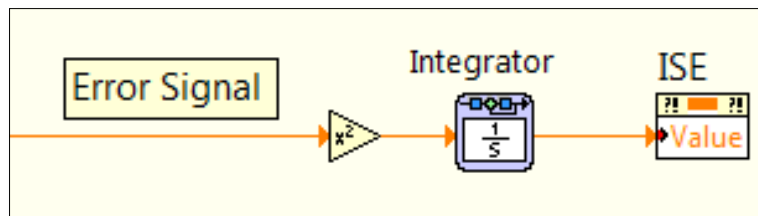
Figure 4-10: Integral of squared error calculation sub VI.

4.4 The Auto-tuning Algorithm

LabVIEW provides several code architectures, known as 'design patterns' (NI, 2013a), such as Producer/Consumer, Master/Slave Model and State Machine which can be used as a base structure for implementing any application. These design patterns provide ease of use, readability and maintainability of the code. Depending on the intended application an optimal design pattern can be chosen.

The most fundamental architecture in LabVIEW is the state machine. This design is used to implement complex decision making algorithms which are normally represented with different states or cases. Each state performs one or multiple tasks. They are also used when an application requires a user interface, when different user actions trigger the execution of different code segments.

Figure 4.11 shows a standard LabVIEW state machine design pattern. It is comprised of a case structure, which executes different code segments for various states, inside a continually running while loop. Each of the segment codes determines the next state.

To keep track the states of the state machine, the structure has two shift registers, as shown in Figure 4.11, 'Beginning State' and 'Next State', which store the current state and the next state values, 'Initialize' and 'Stop' respectively. Depending on the code structure, the case structure can accommodate various cases and the shift registers can be set to different values programmatically or through the application user interface. In this manner, the execution of the code is running continuously and it changes from a state to another according to the value of the shift registers.
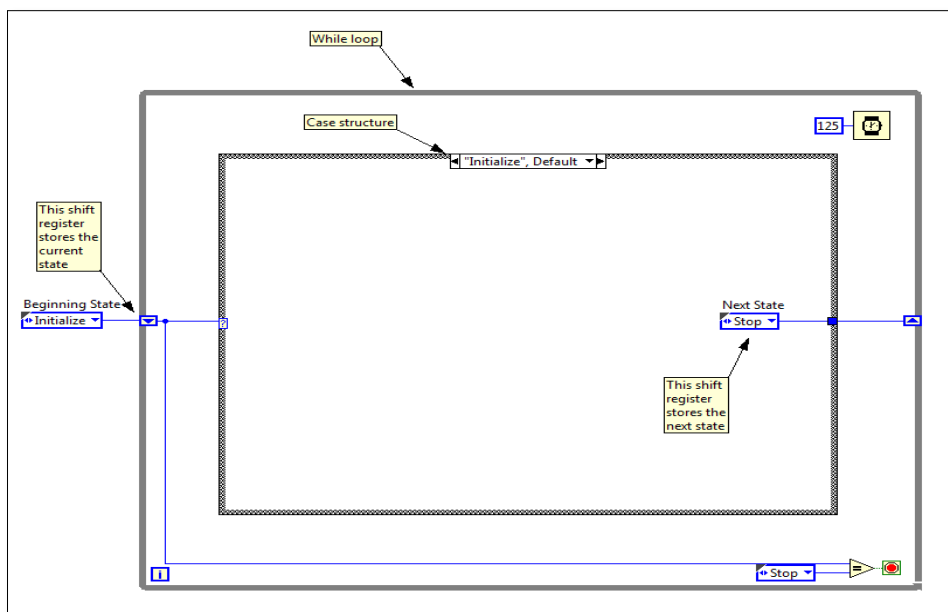
Figure 4-11: The standard LabVIEW state machine design pattern.

78

The auto-tuning algorithm, illustrated by the flowchart in Figure 3.11 in Section 3.5.2, performs different tasks, such as starting, running and stopping the controller; resetting the controller gains; and setting to auto-tuning mode. These tasks are performed in response to commands received from the user interface of the controller and the control algorithm transits from one state to another.

It was decided to choose the sate machine design pattern as a base structure to implement the auto-tuning algorithm as this design pattern is well-chosen and convenient for implementing such type of complex decision-making algorithms.

The auto-tuning algorithm includes the following states: Start, Run, Auto-tune GU, Auto-tune GCE, Auto-tune GIE, Stop and Reset. Some of these states are triggered through the user interface and the rest are set programmatically during runtime. Table 4.1 provides the description of each state.

Table 4-1: The auto-tuning algorithm states descriptions.

State	Description
Start	Initialises the application and the controller settings.
Run	Runs the controller and resumes the execution when the code is halted temporarily.
Auto-tune GU	Auto-tunes GU gain when it receives auto-tune command from the user interface.
Auto-tune GCE	Auto-tunes GCE gain after accomplishing the Auto-tune GU task.
Auto-tune GIE	Auto-tunes GIE gain after accomplishing the Auto-tune GCE task.
Stop	Stops the execution of the controller.
Reset	Resets the controller settings to their default values.

The new user interface of the basic FPD+I controller is shown in Figure 4.12. As it can be noticed, comparing to the previous user interface shown in Figure 4.3, some extra buttons, such as Auto-tune, Reset Gains and Reset Shared Gains; and indicators such as Tuned Gains, are added to accomplish the controller tasks. The new items are denoted by 1 and 2.

The controller can be set to auto-tuning mode through the 'Auto-tune' button, where the algorithm progressively tunes the controller gains G_U, G_{CE} and G_{IE}. To achieve better performance, the auto-tuning process is repeated for every new settings of the 'Step input'. The step input range was set from 1 to 10 with an increasing step of 1.
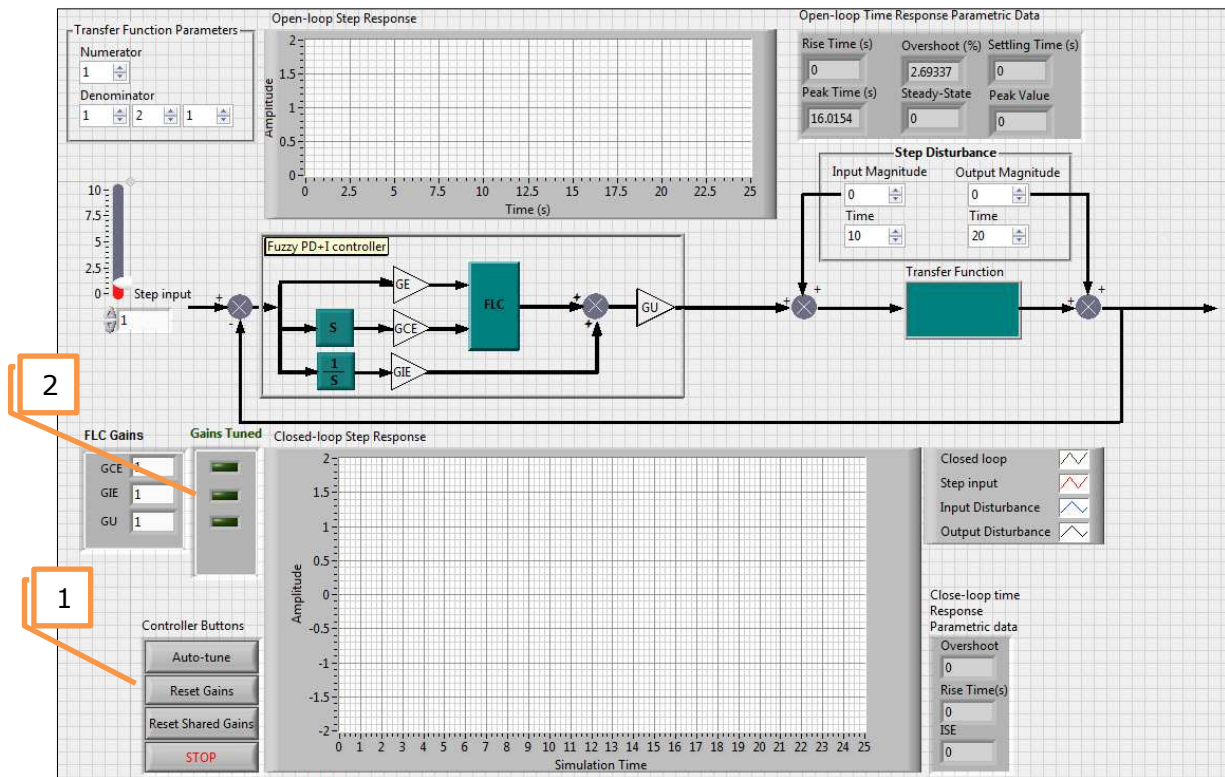
Figure 4-12: The user interface of the basic FPD+I controller with the auto-tuning algorithm.

4.5 The Memory Unit

One of the main features of the auto-tuning algorithm is the ability to learn from the past experience and knowledge. To implement this, memory is essential to retain the acquired experience and knowledge and to restore them at a later stage when a similar situation arises.

In this case the acquired knowledge is the tuned gains of the controller for each setpoint setting. As the setpoint changes from 1 to 10, there will be a similar set of values for each

gain. A two dimension array was used to store the values of the setpoint, G_U, G_{IE} and G_{CE}. Also the values of the percentage overshoot and ISE corresponding to each set of the gains were stored in arrays. To preserve the best settings of the gains, previous values of the gains, the percentage overshoot and the ISE values were also kept in arrays.

To indicate whether a gain value is tuned a flag was set for the corresponding gain. An algorithm was set to store and retrieve the best settings of the gains when the setpoint changes to a new setting.

4.6 The Communication Unit

In order to implement the sharing capability of the auto-tuning algorithm, it is essential to add communication facilities to the controllers to enable them share the tuned gains stored in their local memories.

LabVIEW provides shared variable feature, introduced with LabVIEW 8.0 in 2005, that enables designing distributed applications easier and simplifies the complexities of the programming required to establish communication facilities for such type of applications (NI, 2013d).

The shared variable enables sharing of data between various parts of a VI on a single machine or between various VIs across a network. Its principle is similar to global variables with the advantage of accessing the variable from any node on a network, and therefore it is sometimes referred to as 'network variables'.

Shared variables use a NI proprietary protocol called NI Publish-Subscribe Protocol (NI-PSP) to transport across a network. The NI-PSP utilises standard transport layer protocols of the Transmission Control Protocol/ Internet Protocol (TCP/IP) suite, such as TCP and User Datagram Protocol (UDP) protocols. Since the release of LabVIEW 8.5, in 2007, TCP is used as

the main transport protocol which is more efficient and reliable in transmitting data over a network than UDP, but also has its own overhead and complexities in programming.

To establish the communication link between the controllers, different shared variables were created and they were accessible by the controllers to store and to retrieve their data. Indeed, the local variables created in Section 4.5 to store local data were converted to shared variables. Figure 4.13 shows all shared variables collected in one library file called 'Shared Gains.lvlib'. The properties of one of the gains, G_U shared variable is shown in Figure 4.14, which is created as a numeric single array variable and configured as a network-published variable type.

Another state, reset shared variables, was added to the auto-tuning algorithm that resets the shared variables when it is requested by the user through the user interface button 'Reset Shared Gains'.
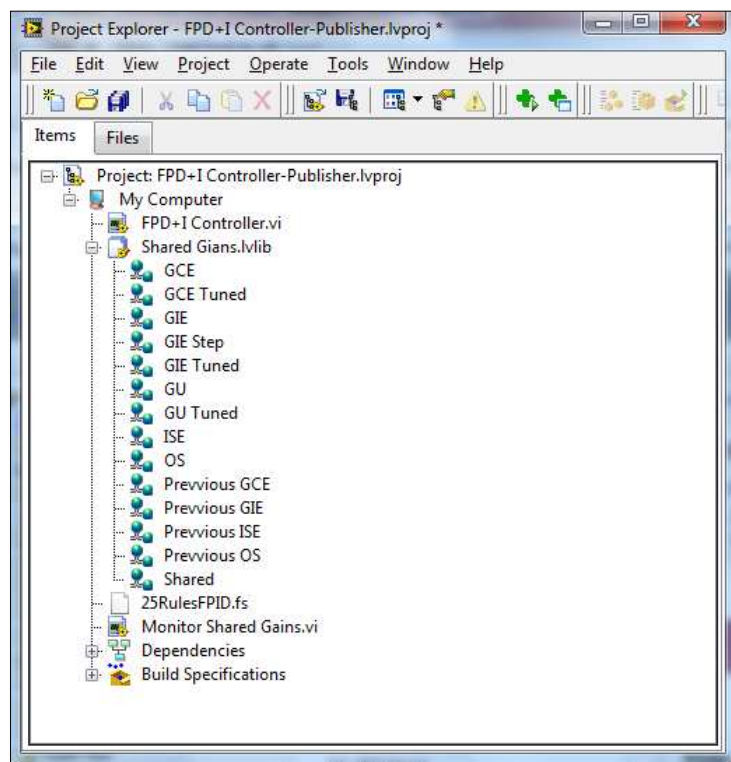
Figure 4-13: The shared variables of the basic FPD+I controller with the auto-tuning algorithm.
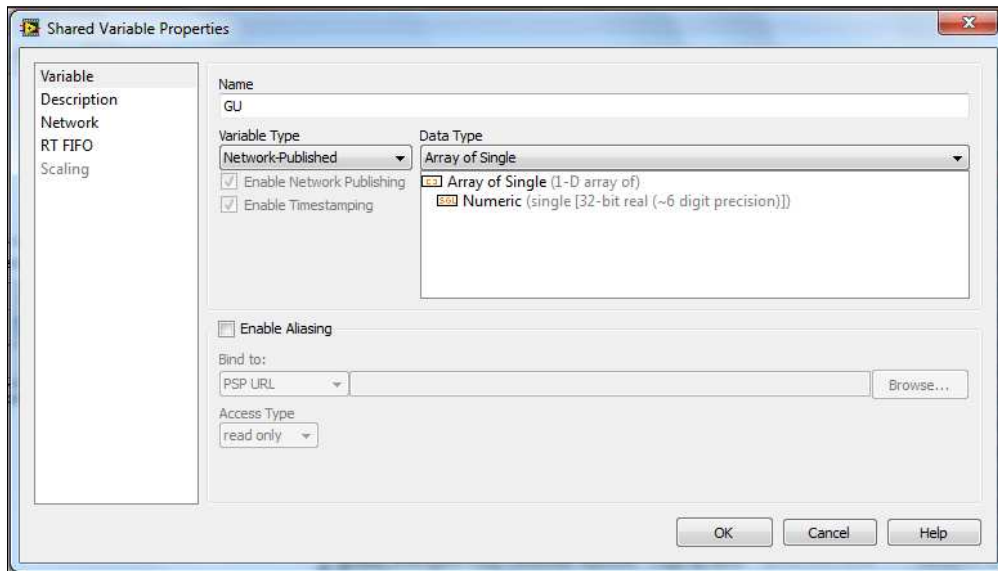
Figure 4-14: The properties of the G_U shared variable.

4.7 The Management and Supervision Unit

As mentioned in details in Section 3.3 and illustrated in Figure 3.3, the overall operation of the individual intelligent fuzzy controllers is managed by the management level. The operations include the auto-tuning algorithm unit, the memory unit and the communication unit.

To accomplish this task an algorithm was coded to manage and supervise the overall operation of the controllers. The following steps illustrate the major tasks of the algorithm. First, a closed-loop test on the system is performed by applying the basic FPD+I controller. Secondly, the auto-tuning algorithm is started and a check is performed to indicate whether the controller gains, G_U, G_{IE} and G_{CE}, are tuned and shared by other controllers. If the gains are ready in the memory unit, then the gains are retained and set as new values to the controller gains, otherwise the controller starts to auto-tune its gains and then makes the new tuned values available to the memory unit through the communication unit. The reading and writing processes from and to the memory unit are performed based on the comparison of the values of ISE and percentage overshoot so that the values resulted in a better performance are retained.

83

4.8 Hardware-in-the-loop

To validate the feasibility of the proposed architecture and test the auto-tuning algorithm on actual hardware, different electronic circuits were constructed and implemented by using operational amplifiers (op-amps) and resistor-capacitor (RC) circuits that emulate the characteristics of various real-time second order systems. The realised circuit then connected to the PCs through the NI PCI-6025E data acquisition (DAQ) card.

4.8.1   Second Order Systems Realisation

In control systems, it is common to use operational amplifiers to implement different type of controllers or compensators (Dorf & Bishop, 2001; Golnaraghi & Kuo, 2010). The operational amplifiers, with RC circuits, can also be utilised to realise various transfer functions as they provide an easy and convenient way to construct such systems. In addition, the approach reduces the time, the cost, and the risk associated with the implementation of control algorithms on real control systems.

The realisation process is also known as electronic filter design (Deliyannis, Sun, & Fidler, 1998) in which active electronic components, such as op-amps and passive components, such as resistors and capacitors are normally used to design various types of filters. By constructing first order transfer functions and second order transfer functions any higher order system can be constructed by cascading the required number of first and second order systems. Furthermore, any second order system can be implemented by connecting two first order systems in series (Deliyannis et al., 1998; Golnaraghi & Kuo, 2010).

From Table 3.2, four different second order transfer functions were chosen, Case 1, Case 2, Case 3 and Case 5 which they represent critical, overdamped, underdamped and unstable systems respectively.

Using the 'Filter Design Tool' provided in (OKAWA-Electric-Design, 2008), the electronic circuit diagrams shown in Figure 4.15 and Figure 4.16 were constructed to implement the abovementioned cases. Depending on the values of the resistors and the capacitors, the circuit diagram 1 as shown in in Figure 4.15 was used to approximately implement Case 1, Case 2 and Case 3; while the circuit diagram 2 as shown in Figure 4.16 was used to implement Case 5.
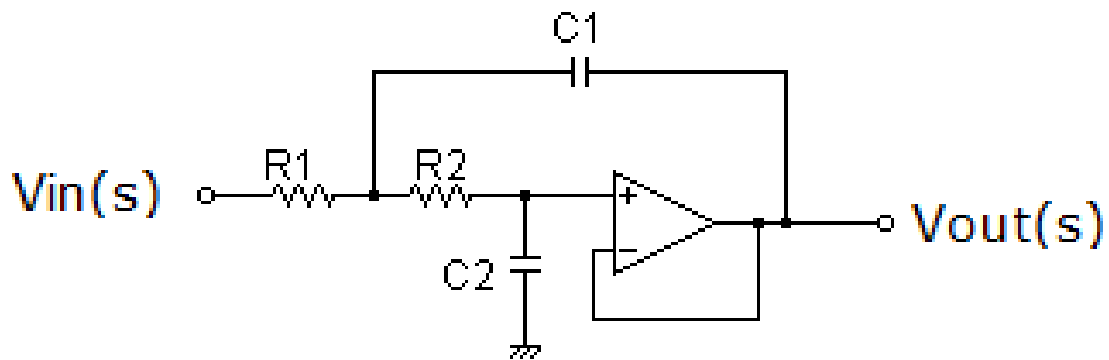
Figure 4-15: Circuit diagram 1 of constructing critical, overdamped and underdamped second order systems.
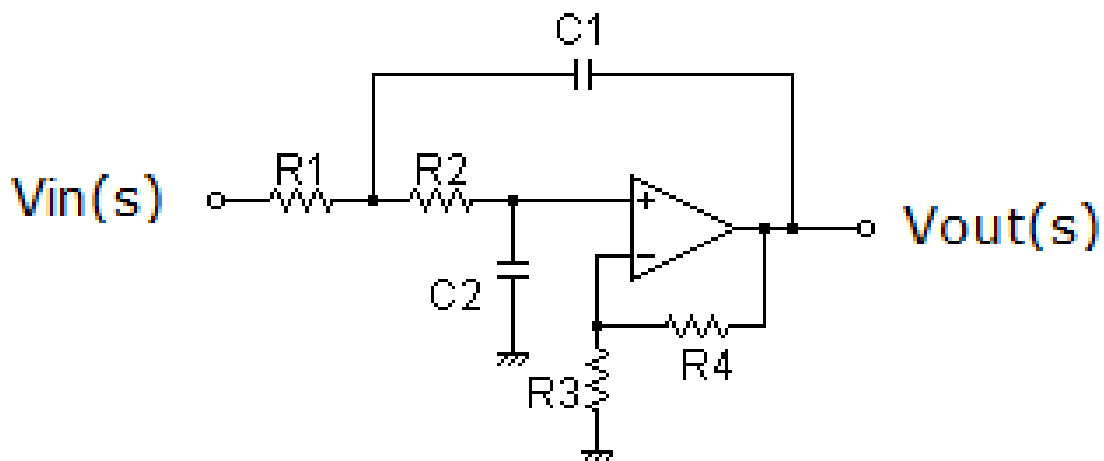
Figure 4-16: Circuit diagram 2 of constructing unstable second order systems.

The transfer functions of circuit diagram 1 and circuit diagram 2 are shown in Equation (18) and Equation (19) respectively (OKAWA-Electric-Design, 2008):

85

$$\frac{Vout(s)}{Vin(s)} = \frac{\frac{1}{R1C1R2C2}}{S^2 + S\left(\frac{1}{R2C1} + \frac{1}{R1C1}\right) + \frac{1}{R1C1R2C2}} \qquad (18)$$

$$\frac{Vout(s)}{Vin(s)} = \frac{\frac{G}{R1C1R2C2}}{S^2 + S\left(\frac{1}{R2C1} + \frac{1}{R1C1} + \frac{1}{R2C2}\right) * (1-G) + \frac{1}{R1C1R2C2}} \qquad (19)$$

Where G in Equation (19) is:

$$G = \frac{R3 + R4}{R3}$$

By choosing appropriate and different values of resistors and capacitors for circuit diagram 1 and substituting the values in Equation (18), the transfer functions of Case 1, Case 2 and Case 3 were calculated. The resistors and capacitors values and the calculated transfer functions are shown in Table 4.2.

Table 4-2: Implementation of critical, overdamped and underdamped transfer functions using various RC values.

Case	Resistor (Kilo Ohm)		Capacitor (Micro Farad)		Transfer Function
	R1	R2	C1	C2	
1	100	100	10	10	$\dfrac{1}{S^2 + 2S + 1}$
2	8	5.5	100	220	$\dfrac{1.03}{S^2 + 3.07S + 1.03}$
3	33	15	100	22	$\dfrac{0.92}{S^2 + 0.97S + 0.92}$

To construct Case 5, the values of resistors and capacitors were chosen for circuit diagram 2 and these values were substituted in Equation (19) to calculate the transfer function. The resistors and capacitors values and the calculated transfer functions of this case are shown in Table 4.3.

Table 4-3: Implementation of an unstable transfer function using various RC values.

Case	Resistor (Kilo Ohm)				Capacitor (Micro Farad)		Transfer Function
	R1	R2	R3	R4	C1	C2	
5	1000	1000	10	0.39	10	0.1	$\dfrac{1.04}{S^2 - 0.19\,S + 1}$

It is apparent that the transfer functions obtained from the realisation, listed in Table 4.2 and Table 4.3, are not identical to the transfer functions listed in Table 3.2, this was on account of using available standard values of resistors and capacitors.

4.8.2 PC Interfacing

In order to connect the PCs to the electronic circuits, each PC was equipped with a NI PCI-6025E data acquisition (DAQ) card (NI, 2013c). The NI PCI-6025E is a multifunction DAQ card that provides analogue-to-digital conversion (ADC) and digital-to-analogue conversion (DAC) functionalities through 16 analogue input (AI) channels, single-ended, with 12-bits resolution; and two 12-bits analogue output (AO) channels. The signal range of the channels is from 0 to +20 volts, but can be configured to produce a maximum bipolar voltage of -10 and +10.

During the simulating of the control algorithm in Section 3.5.2 and Section 4.4 the control signal was changing between -20v and +20v. As the maximum output voltage of the NI PCI-6025E is limited to -10v to +10v, and then it was required to amplify the output signal from the NI PCI-6025E to achieve the required voltages of -20 and +20. Therefore, an amplification

circuit, using op-amp, was added between the NI PCI-6025E and the realised electronic circuits.

A schematic diagram is shown in Figure 4.17 where an electronic circuit of a critical damped system, Case 1 in Table 4.2, is connected to a PC via the NI PCI-6025E interface card. The control signal was applied to the circuit thought AO0, pin 20, with an amplification stage circuit of a gain of two in-between, and then the output of the system was measured and connected through AI0, pin 3. The interface card shows only the pin out diagram of the used ports. Henceforth the whole electronic circuit, amplification and the realised circuits are referred to as the hardware circuit.
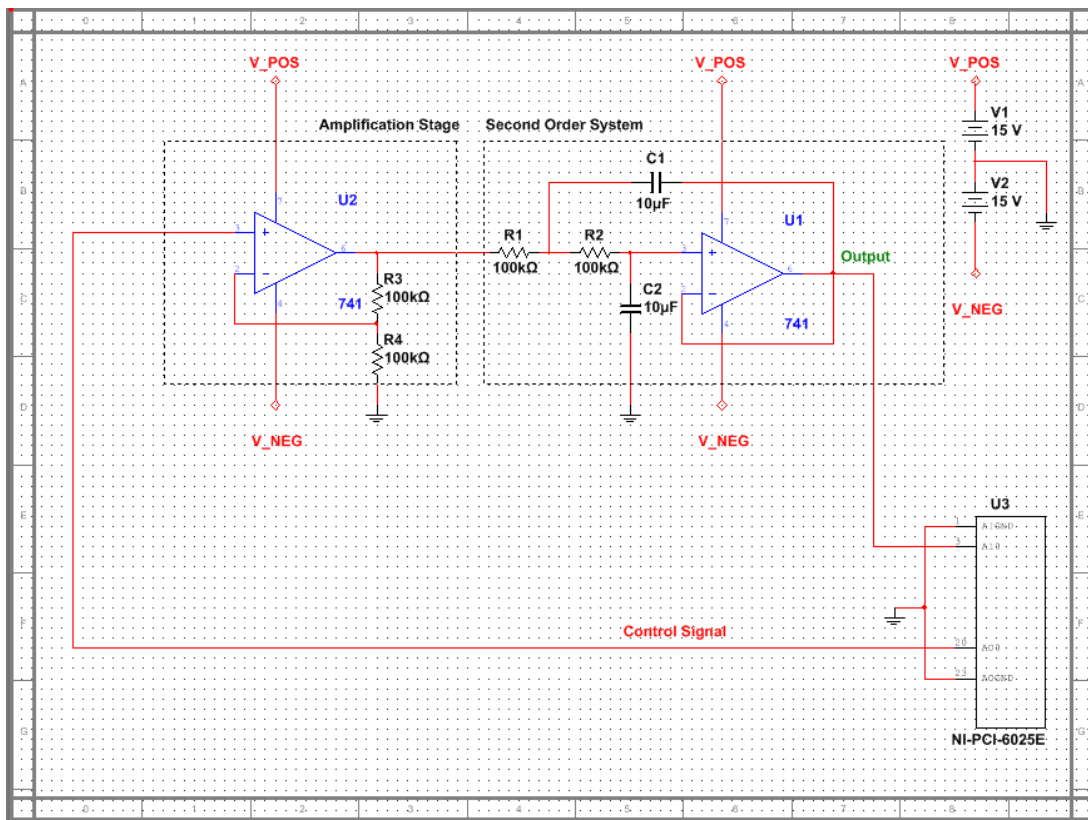
Figure 4-17: Schematic diagram of the hardware circuit interface with a PC.

To connect the controller software, Fuzzy PD+I controller VI designed in Section 4.4, to the hardware circuit the **Transfer Function VI** of Figure 4.3 was replaced with the **DAQ Assistant Express VI** which is a software module used to interactively input and output signals to and from LabVIEW programs. Figure 4.18 shows the new block diagram of the controller VI where two instances of NI-DAQmx VI were used: Output control VI, denoted by 1, to manage the connection to the hardware circuit and output the control signal; and Measure Output VI, denoted by 2, to measure the output of the system. The ports AO0 and AI0 of the interface card were used to output the control signal and measure the output of the system respectively.
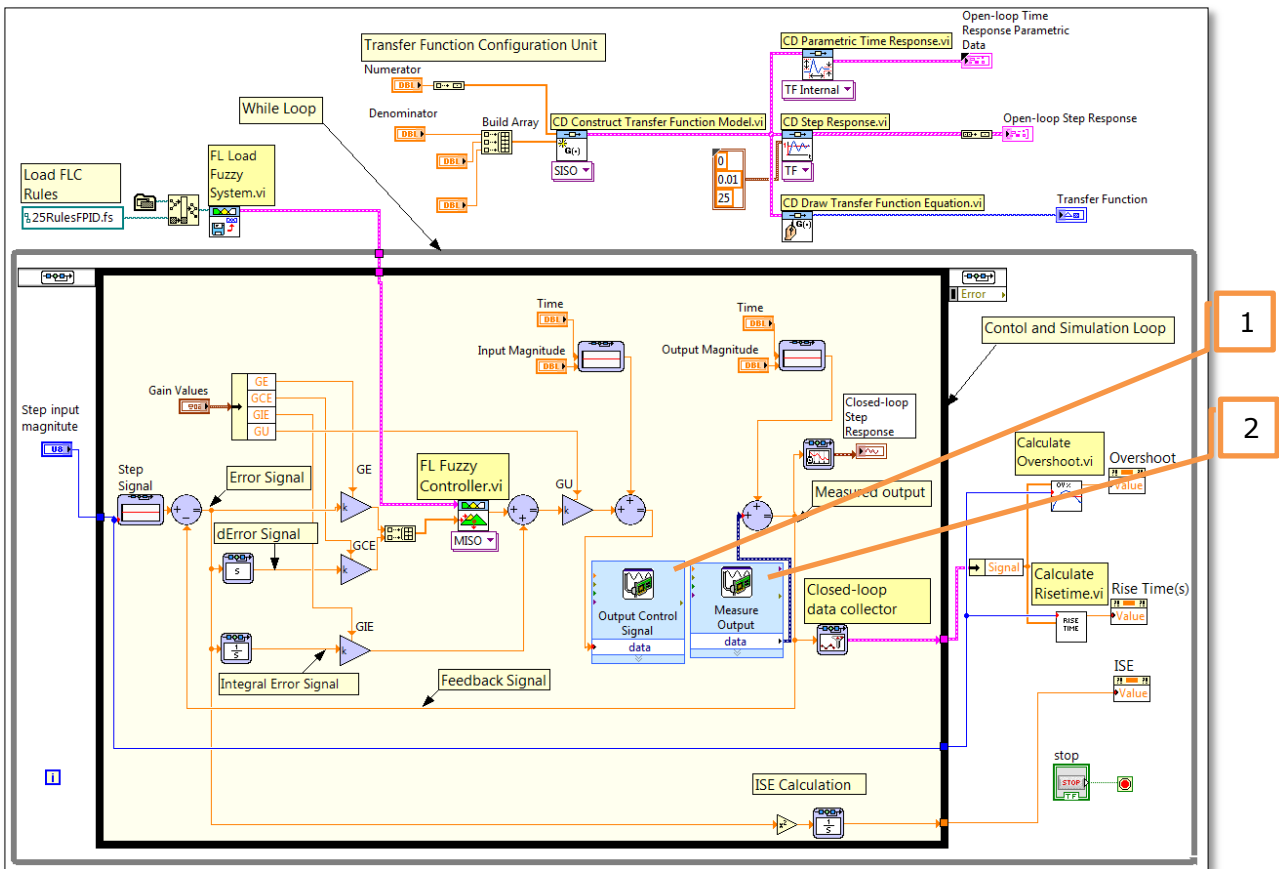
Figure 4-18: Block diagram of the basic FPD+I controller VI interfaced with the hardware circuit.

4.9 Summary

This chapter provided details of the development of the wireless intelligent fuzzy controller network. The LabVIEW package has been used as the main development platform. The design of the basic FPD+I controller, the auto-tuning algorithm, the memory unit and the communication unit are presented in detail.

In order to test the system and the auto-tuning algorithm, hardware circuit representing various standard second order transfer functions constructed from various electronic components.

In the next chapter, the wireless intelligent fuzzy controller network, the basic FPD+I controller and the auto-tuning algorithm will be tested using MATLAB, LabVIEW and the hardware circuit.

CHAPTER 5

Tests and Results

One of the main steps in the design process of any control algorithm is to perform various tests to investigate the performance of the control algorithm under different circumstances. An essential test such as step response is normally conducted to assess the achievement of certain design specifications, such as percentage overshoot, rise time in transient response and steady stated error. Further tests may include step input tracking, disturbance rejection, robustness or stability tests.

This chapter demonstrates the conduction of various tests and presents the results obtained. Firstly, MATLAB simulation-based models were used to test the basic FPD+I controller and the auto-tuning algorithm. Secondly, by using LabVIEW in addition to testing the basic FPD+I controller and the auto-tuning algorithm, the wireless intelligent fuzzy controller network system is tested. And finally, the basic FPD+I controller and the auto-tuning algorithm are tested on hardware systems representing various real-time control systems.

5.1 MATLAB-Based Tests

MATLAB offers integrated capabilities in plotting and computing of control characteristics, such as overshoot and rise time. These capabilities provide an efficient platform for developing, testing and debugging different control algorithms.

In this section, various simulation-based models are used to test the basic FPD+I controller and the auto-tuning algorithm. In addition, the auto-tuning algorithm is tested for various disturbances.

5.1.1 The Basic FPD+I Controller

Using the designed simulation model of the basic FPD+I controller presented in Section 3.5.1, a unit step input was applied to all the systems with the transfer functions listed in Table 3.2. The **Basic FPD+I Controller Script Code**, listed in Appendix B, was developed to simulate the model shown in Figure B.2 in Appendix B, and to generate the required responses. The controller gains: G_E, G_{CE}, G_{IE} and G_U were set to 1, and the simulation time was set to 20 seconds with a fixed 0.01 second sampling interval.

Figure 5.1 shows the step response of the closed-loop system alongside the open-loop system of Case 3, where the maximum percentage overshoot (M_p), the rise time (t_r), the settling time (t_s) and the steady state error (SSE) for both systems are shown. The step responses of Case 1, Case 2, Case 4 and Case 5 are shown in Figure C-13 – Figure C-16 in Appendix C.
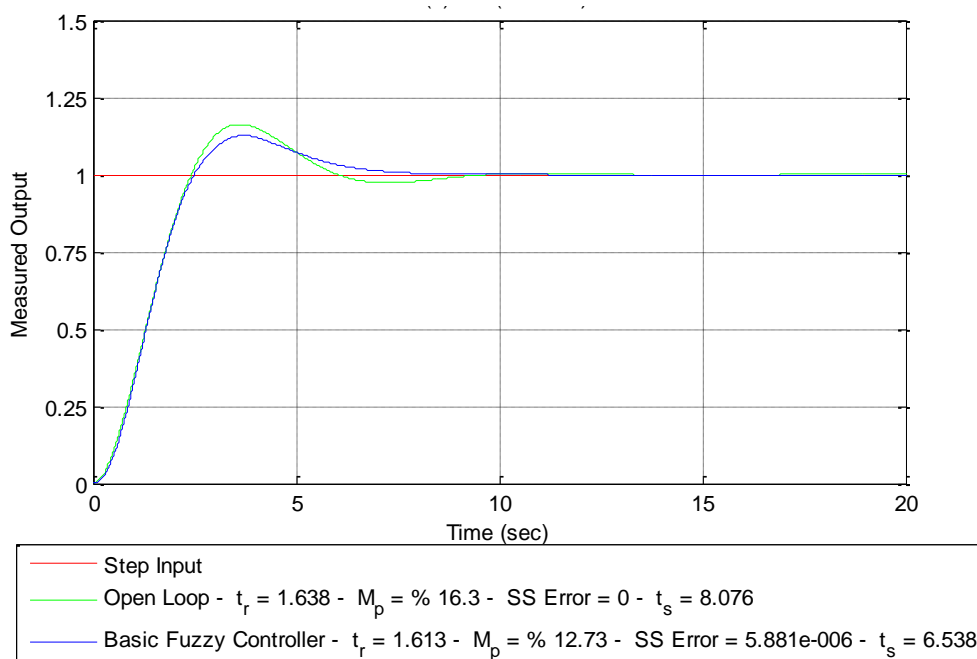
Figure 5-1: Closed-loop step response of Case 3 using the basic FPD+I controller.

As it can be noted from above figure, the basic FPD+I controller provided an improved step response. The maximum percentage overshoot was reduced from 16.3% to 12.73%. The settling time was reduced from 8.07 seconds to 6.53 seconds. The results are encouraging, despite the fact that the controller gains are not yet tuned.

5.1.2 The Auto-tuning Algorithm

To test the auto-tuning algorithm the model presented in Section 3.5.1 was used. A unit pulse input, with a period of 30 seconds and a duty cycle of 50%, was applied to all the systems with the transfer functions listed in Table 3.2. The **Auto-tuning Algorithm Script Code**, listed in Appendix B, was used to simulate the model and to generate the required responses. The controller gain values were initially set to 1, and the simulation time was set to 30 seconds with a fixed 0.01 second sampling interval. The step sizes of the changes in G_{CE} and G_{IE} gains were set as follows: G_{CE} was set to decrease in a step size of 0.1 and G_{IE} was set to increase in a step size equal to twice the value of the initial value of G_{IE} and then in each subsequent step this step size was doubled again.

Figure 5.2 shows closed-loop responses of Case 3, where the responses are iteratively shown from left-to-right top-to-bottom. The first iteration response includes the open-loop step response alongside the closed-loop step response. The step responses of Case 1, Case 2, Case 4 and Case 5 are shown in Figure C-17 – Figure C-20 in Appendix C.
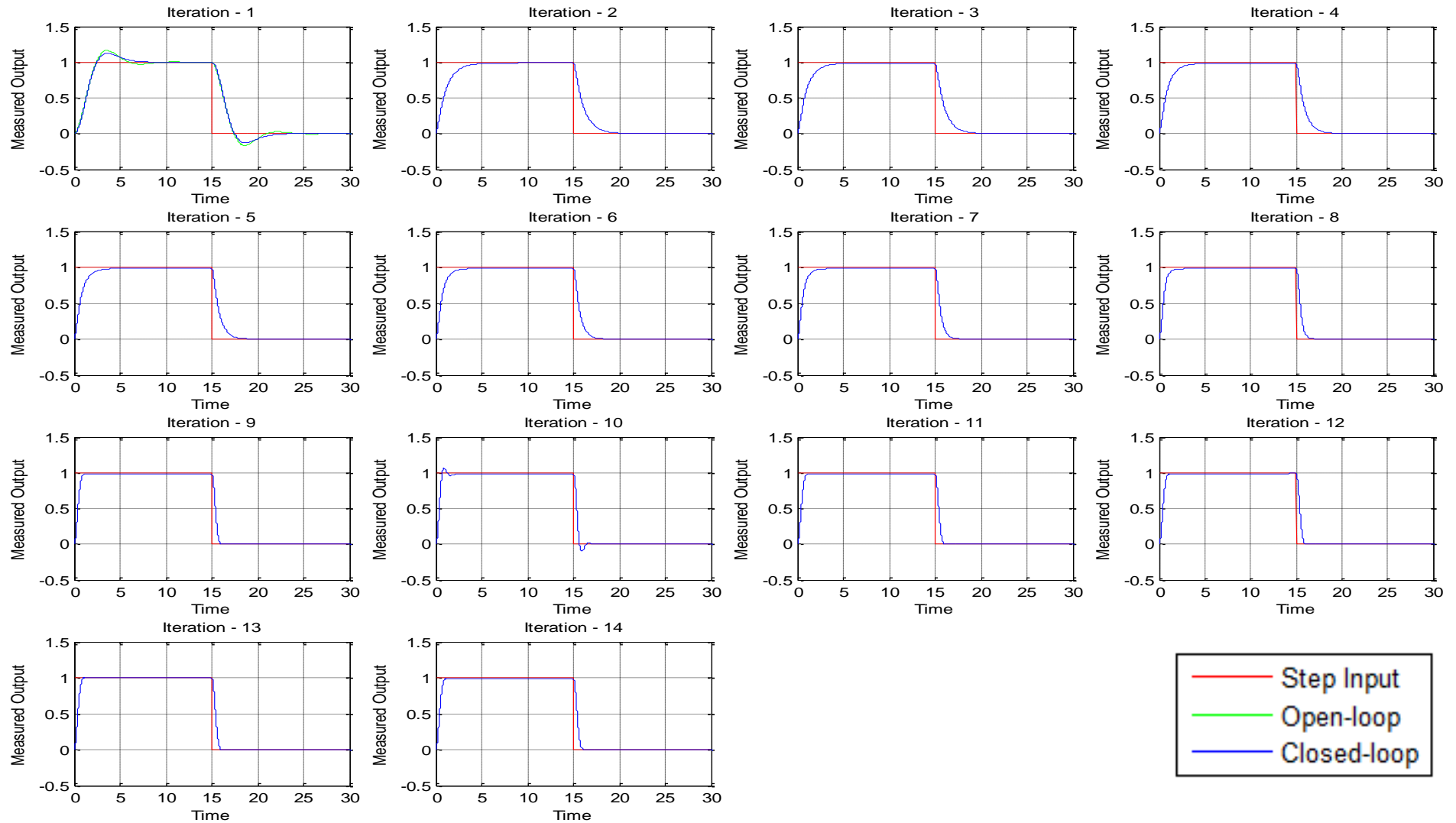
Figure 5-2: Closed-loop step responses for Case 3 using the auto-tuning algorithm for step sizes of G_{CE} = 0.1 and G_{IE} = twice of the initial value of G_{IE}.

94

The closed-loop step responses from Figure 5.2 depict substantial improvements in the transient and the steady-state responses. The performance was considerably improved from the second iteration by eliminating the overshoot and then in an iterative manner the rise time and the settling time improved. The auto-tuning process was accomplished in 14 iterations.

Table 5.1 lists the controller gains' values, the ISE values and the closed-loop characteristics of Case 3 for each iteration. Similar data of Case 1, Case 2, Case 4 and Case 5 are listed in Table C-1 – Table C-4 in Appendix C.

Table 5-1: Controller gains' values and closed-loop characteristics of Case 3 using the auto-tuning algorithm for step sizes of $G_{CE} = 0.1$ and G_{IE} = twice of the initial value of G_{IE}.

Iteration	Controller Gain				ISE	Closed-loop Characteristics			
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p	t_s	SSE
1	1	1	1	1	1.017	1.613	12.730	6.538	0
2	1	1	0.039	12.730	0.644	2.419	0	5.070	0.009
3	1	0.9	0.039	12.730	0.592	2.187	0	4.750	0.011
4	1	0.8	0.039	12.730	0.540	1.949	0	4.420	0.012
5	1	0.7	0.039	12.730	0.489	1.702	0	4.083	0.013
6	1	0.6	0.039	12.730	0.440	1.444	0	3.738	0.014
7	1	0.5	0.039	12.730	0.394	1.171	0	3.406	0.016
8	1	0.4	0.039	12.730	0.351	0.878	0	3.328	0.017
9	1	0.3	0.039	12.730	0.316	0.611	0	6.771	0.018
10	1	0.2	0.039	12.730	0.296	0.446	6.332	10.186	0.019
11	1	0.3	0.039	12.730	0.316	0.611	0	6.771	0.018
12	1	0.3	0.079	12.730	0.312	0.598	0	1.013	0.010
13	1	0.3	0.157	12.730	0.306	0.574	0.178	0.910	0.002
14	1	0.3	0.079	12.730	0.312	0.598	0	1.013	0.010

The table above represents relationships between the controller gains' values and the closed-loop characteristics of the system where changes in the gains resulted in improvement of the characteristics. It can be seen from the data that the maximum percentage overshoot, M_p, was completely eliminated and became zero from the second iteration. This was as a result of a rapid change in the value of G_{IE} from 1 to 0.039.

The data also show improvements in the rise time and the settling time, t_r and t_s respectively, beginning from iteration 3 where the value of G_{CE} started to decrease while retaining the values of G_{IE} and G_U.

The algorithm utilises ISE and Mp to monitor the overall performance of the controller and to indicate further changes in the values of G_{CE} and G_{IE}. For instance, in iteration 10, due to a continuous decreasing of the value of G_{CE}, the overshoot increased from 0 to 6.332. Therefore, to preserve a better performance, in iteration 11 the algorithm reversed to the previous value of G_{CE}, iteration 9, and then started to increase the value of G_{IE}. A similar situation happened again when the overshoot changed from 0 to 0.178 in iteration 13, and then accordingly in iteration 14 the algorithm reversed to the previous value of G_{IE}, iteration 12.

In order to establish whether the step sizes of G_{CE} and G_{IE} have any effects on the time required to tune the controller and on the performance of the controller, two other tests were carried out with different step sizes. In the first test with smaller steps, the gains were set as: $G_{CE} = 0.05$ and GIE = 1.5 fold of the initial value of G_{IE}, and in the second test with larger step sizes. The gains were set as: $G_{CE} = 0.2$ and $G_{IE} = 3$ fold of the initial value of G_{IE}.

Figure 5.3 and Figure 5.4 show the closed-loop responses for the first test, and Figure 5.5 shows the closed-loop responses for the second test. Table 5.2 and Table 5.3 list the controller gains' values, ISE and the closed-loop characteristics of both tests.
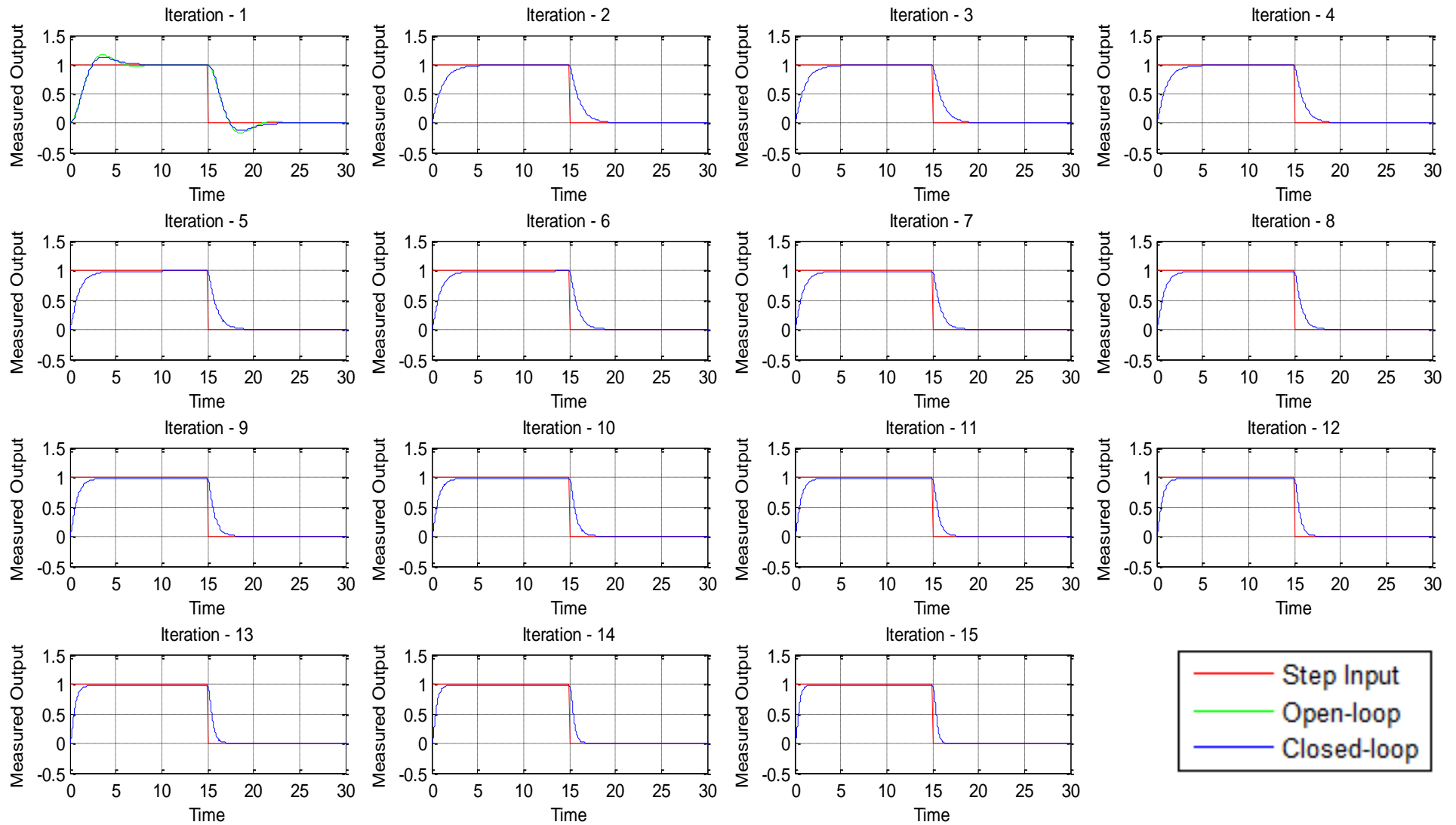
Figure 5-3: Closed-loop step response for Case 3, iteration 1 – iteration 15, using the auto-tuning algorithm for step sizes of G_{CE} = 0.05 and

G_{IE} = 1.5 fold of initial value of G_{IE}.

97

Figure 5-4: Closed-loop step response, iteration 16 – iteration 23, for Case 3 using the auto-tuning algorithm for step sizes of G_{CE} = 0.05 and G_{IE} = 1.5 fold of initial value of G_{IE}.

Table 5-2: Controller gains' values and closed-loop characteristics of Case 3 using the auto-tuning algorithm for step sizes of G_{CE} = 0.05 and G_{IE} = 1.5 fold of initial value of G_{IE}.

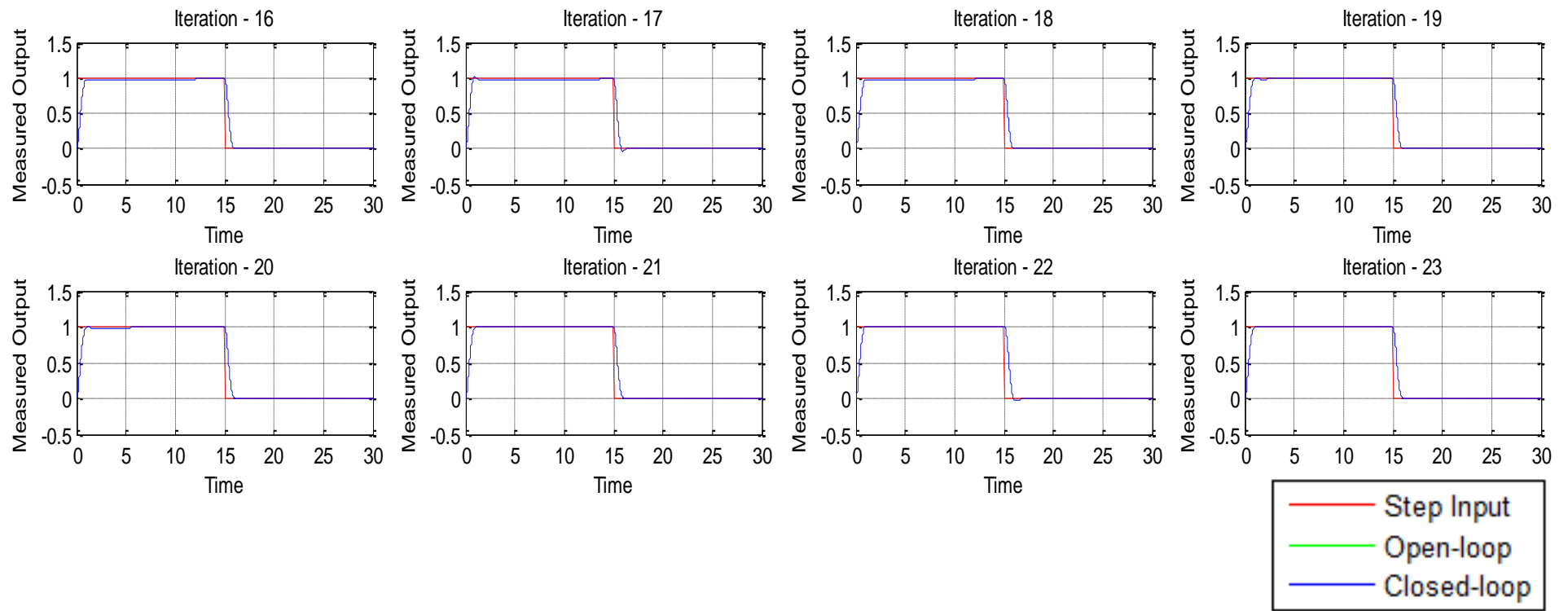| Iteration	Controller Gain				ISE	Closed-loop Characteristics			
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p	t_s	SSE
1	1	1	1	1	1.017	1.613	12.730	6.538	0
2	1	1	0.039	12.730	0.644	2.419	0.000	5.070	0.009
3	1	0.95	0.039	12.730	0.618	2.304	0.000	4.911	0.010
4	1	0.90	0.039	12.730	0.592	2.187	0.000	4.750	0.011
5	1	0.85	0.039	12.730	0.565	2.069	0.000	4.587	0.011
6	1	0.80	0.039	12.730	0.540	1.949	0.000	4.420	0.012
7	1	0.75	0.039	12.730	0.514	1.827	0.000	4.252	0.013
8	1	0.70	0.039	12.730	0.489	1.702	0.000	4.083	0.013
9	1	0.65	0.039	12.730	0.464	1.575	0.000	3.910	0.014
10	1	0.60	0.039	12.730	0.440	1.444	0.000	3.738	0.014
11	1	0.55	0.039	12.730	0.416	1.310	0.000	3.566	0.015
12	1	0.50	0.039	12.730	0.394	1.171	0.000	3.406	0.016
13	1	0.45	0.039	12.730	0.372	1.026	0.000	3.281	0.016
14	1	0.40	0.039	12.730	0.351	0.878	0.000	3.328	0.017
15	1	0.35	0.039	12.730	0.332	0.734	0.000	4.701	0.017
16	1	0.30	0.039	12.730	0.316	0.611	0.000	6.771	0.018
17	1	0.25	0.039	12.730	0.304	0.516	0.710	8.355	0.018
18	1	0.30	0.039	12.730	0.316	0.611	0.000	6.771	0.018
19	1	0.30	0.059	12.730	0.314	0.604	0.000	1.848	0.014
20	1	0.30	0.088	12.730	0.311	0.595	0.000	0.994	0.009
21	1	0.30	0.133	12.730	0.308	0.581	0.000	0.934	0.004
22	1	0.30	0.199	12.730	0.305	0.563	1.086	0.878	-0.001
23	1	0.30	0.133	12.730	0.308	0.581	0.000	0.934	0.004
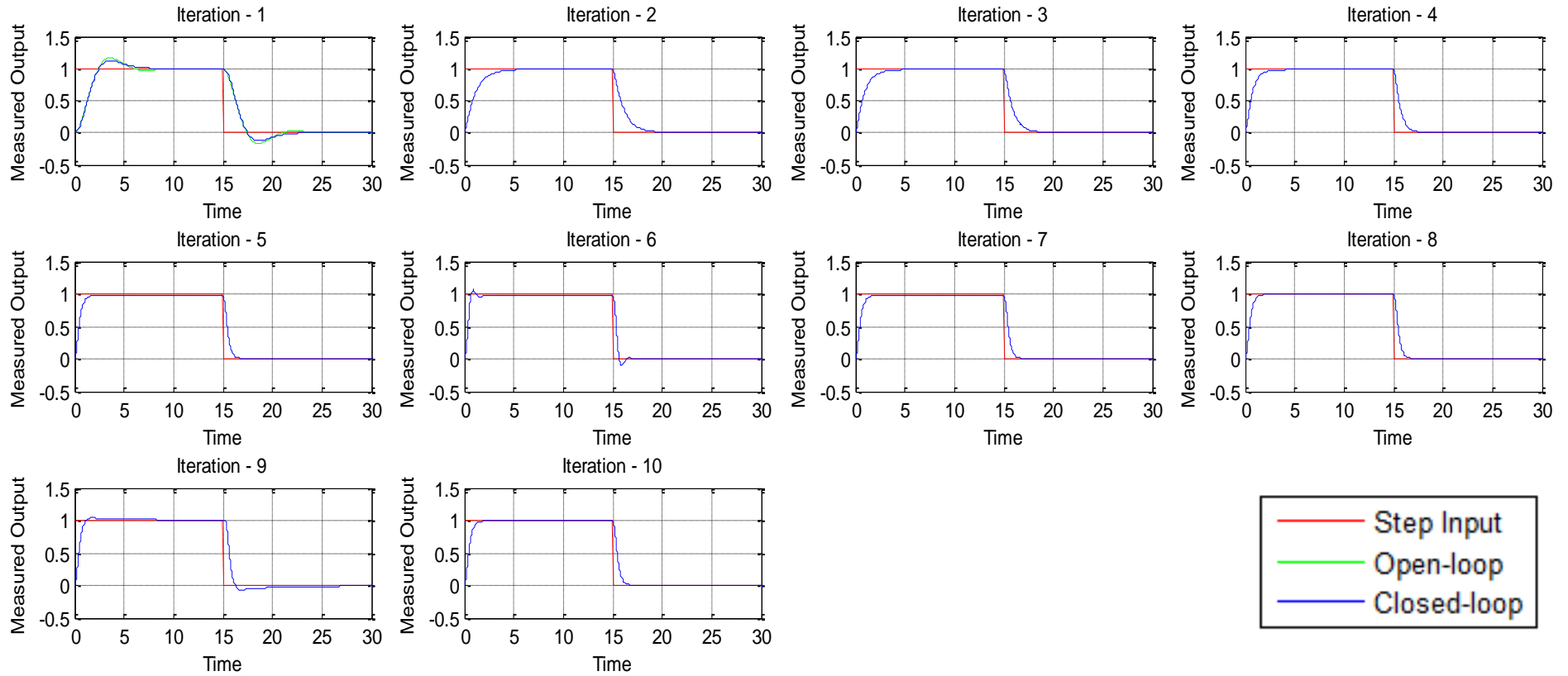
Figure 5-5: Closed-loop step response for Case 3 using the auto-tuning algorithm for step sizes of G_{CE} = 0.2 and G_{IE} = 3 fold of initial value of G_{IE}.

Table 5-3: Controller gains' values and closed-loop characteristics of Case 3 using the auto-tuning algorithm for step sizes of G_{CE} = 0.2 and G_{IE} = 3 fold of initial value of G_{IE}.

Iteration	Controller Gain				ISE	Closed-loop Characteristics			
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p	t_s	SSE
1	1	1	1	1	1.017	1.613	12.730	6.538	0
2	1	1	0.039	12.730	0.644	2.419	0.000	5.070	0.009
3	1	0.8	0.039	12.730	0.540	1.949	0.000	4.420	0.012
4	1	0.6	0.039	12.730	0.440	1.444	0.000	3.738	0.014
5	1	0.4	0.039	12.730	0.351	0.878	0.000	3.328	0.017
6	1	0.2	0.039	12.730	0.296	0.446	6.332	10.186	0.019
7	1	0.4	0.039	12.730	0.351	0.878	0.000	3.328	0.017
8	1	0.4	0.118	12.730	0.341	0.813	0.000	1.582	0.003
9	1	0.4	0.353	12.730	0.333	0.699	4.239	4.802	-0.005
10	1	0.4	0.118	12.730	0.341	0.813	0.000	1.582	0.003

It can be seen from the data in above tables that decreasing the step sizes led to an increase in the number of iterations and vice versa. The number of iterations became 23 and 10 for both tests respectively. However, performance was improved in the first test while it was reduced in the second test. This is also shown in Figure 5.3, Figure 5.4 and Figure 5.5.

5.1.3 Disturbance Reduction Tests

One of the most important advantages of any feedback controller is to reduce the effects of disturbances that may cause the output of the controlled system to drift from the setpoint. The significance of tuned controllers becomes apparent when disturbances are applied, as tuned controllers are expected to perform better than their non-tuned or poorly-tuned counterpart controllers.

Two disturbance reduction tests were conducted. In the first test the basic FPD+I controller with its default gain values, set to 1, was used and applied to Case 3. While in the second test the same controller was used and applied to Case 3 with the tuned gains' values obtained from the application of the auto-tuning algorithm. The values of the tuned gains were obtained from Table 5.1, iteration 14, and they were as follows: $G_E = 1$, $G_{CE} = 0.3$, $G_{IE} = 0.079$ and $G_U = 12.730$.

In both tests, the simulation were setup to run for 100 seconds, then a step input was applied and a sequence of step disturbances were forced on to the system: one at the time = 40 seconds and the second disturbance at the time = 70 seconds. Both persisted for 10 seconds with the magnitude of +1 and -1 respectively. The design model shown in Figure 5.6 and the **Disturbance Test Script Code** listed in Appendix B were used to generate the test responses.
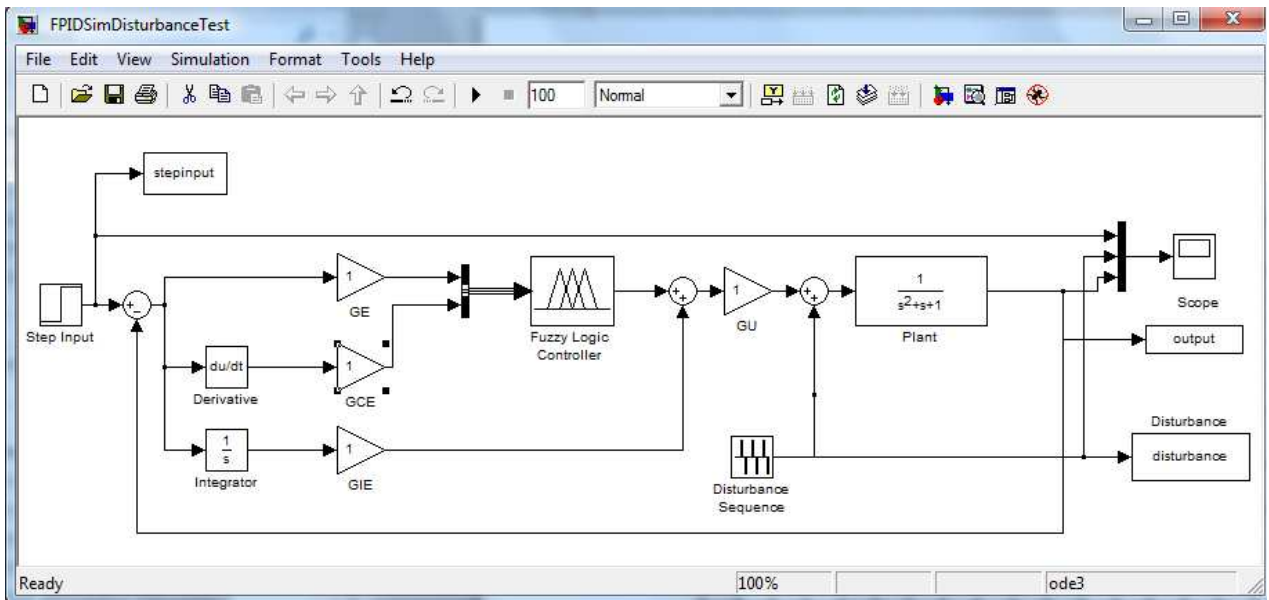
Figure 5-6: Disturbance test simulation model design.

Figure 5.7 and Figure 5.8 illustrate the responses for the basic FPD+I controller using different gains' values.
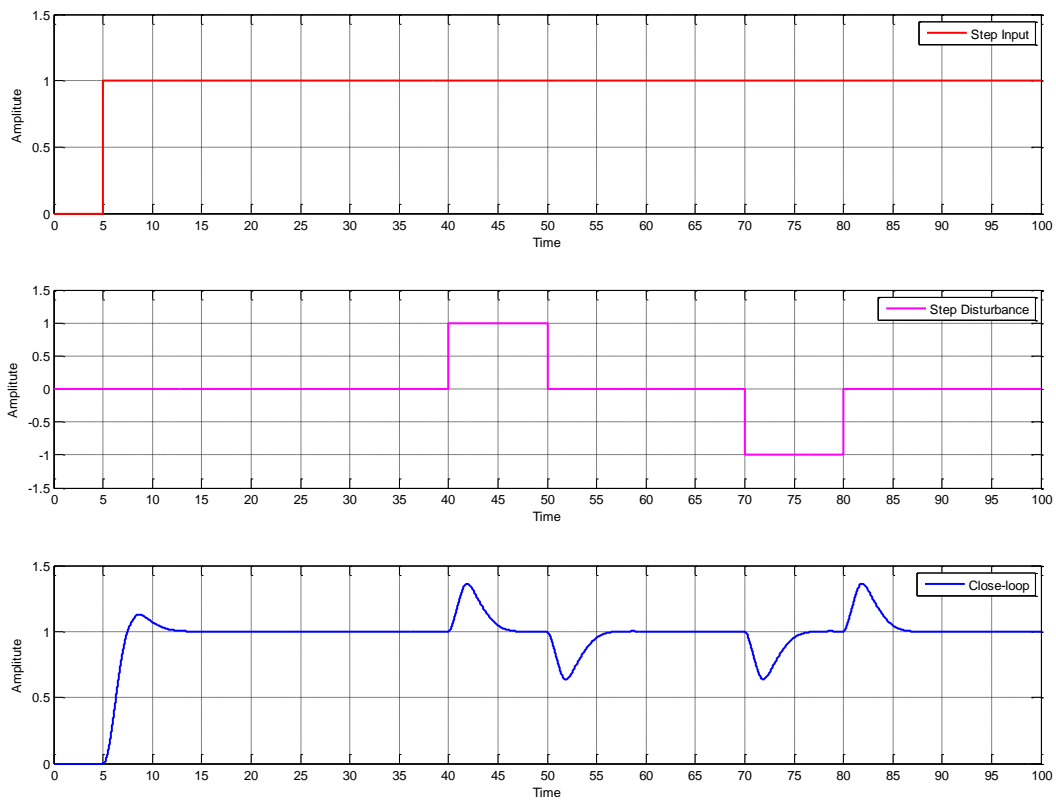
Figure 5-7: Step disturbance response of Case 3 using the basic FPD+I controller with default gains' values.

103
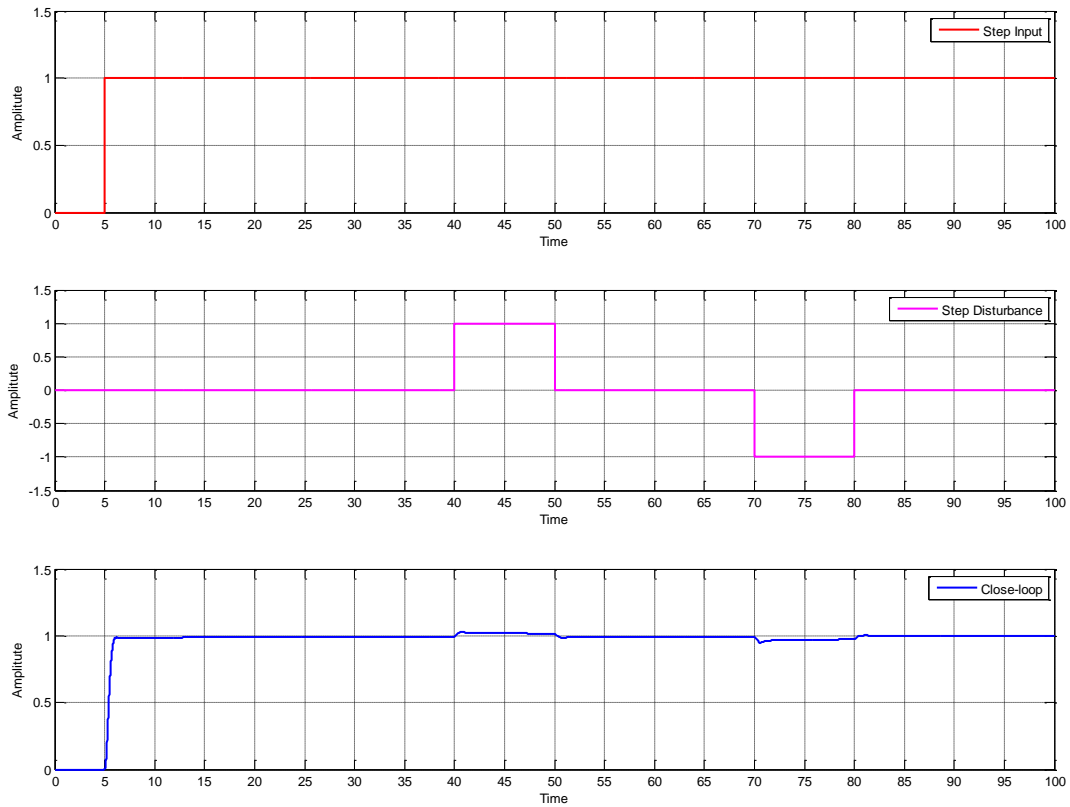
Figure 5-8: Step disturbance response of Case 3 using the basic FPD+I controller with tuned gains' values.

As it can be seen from Figure 5.7, the closed-loop response, that the disturbances caused an overshoot of 39% in the rising edge and the falling edge of each disturbance signals and the overshoot persisted for about 6 seconds.

Whilst, the overshoot was reduced to only 5.7%, but lasted for 10 seconds when the tuned gains' values were used. This is shown in the closed-loop response of Figure 5.8.

104

5.2 LabVIEW-Based Tests

In this section the basic fuzzy controller and the auto-tuning algorithm are tested using LabVIEW. The wireless intelligent fuzzy controller network system functionalities and capabilities are also tested.

5.2.1 The Basic Fuzzy Controller

Using the basic fuzzy controller designed and presented in Section 4.3, all the systems with the transfer functions listed in Table 3.2 were simulated. The user interface was used to set the transfer function parameters and the controller gains, in which the gains were set to 1. A unit step input was applied and the simulation time was set to 20 seconds with a fixed 0.01 second sampling interval.

Figure 5.9 shows the user interface where the settings to simulate Case 3 are shown. The step response of the closed-loop system and the open-loop system are also shown alongside with their characteristic parameters, such as the maximum percentage overshoot (M_p), the rise time (t_r), the settling time (t_s) and the steady state error (SSE). The closed-loop step responses of Case 1, Case 2, Case 4 and Case 5 are shown in Figure D-1 – Figure D-4 in Appendix D.
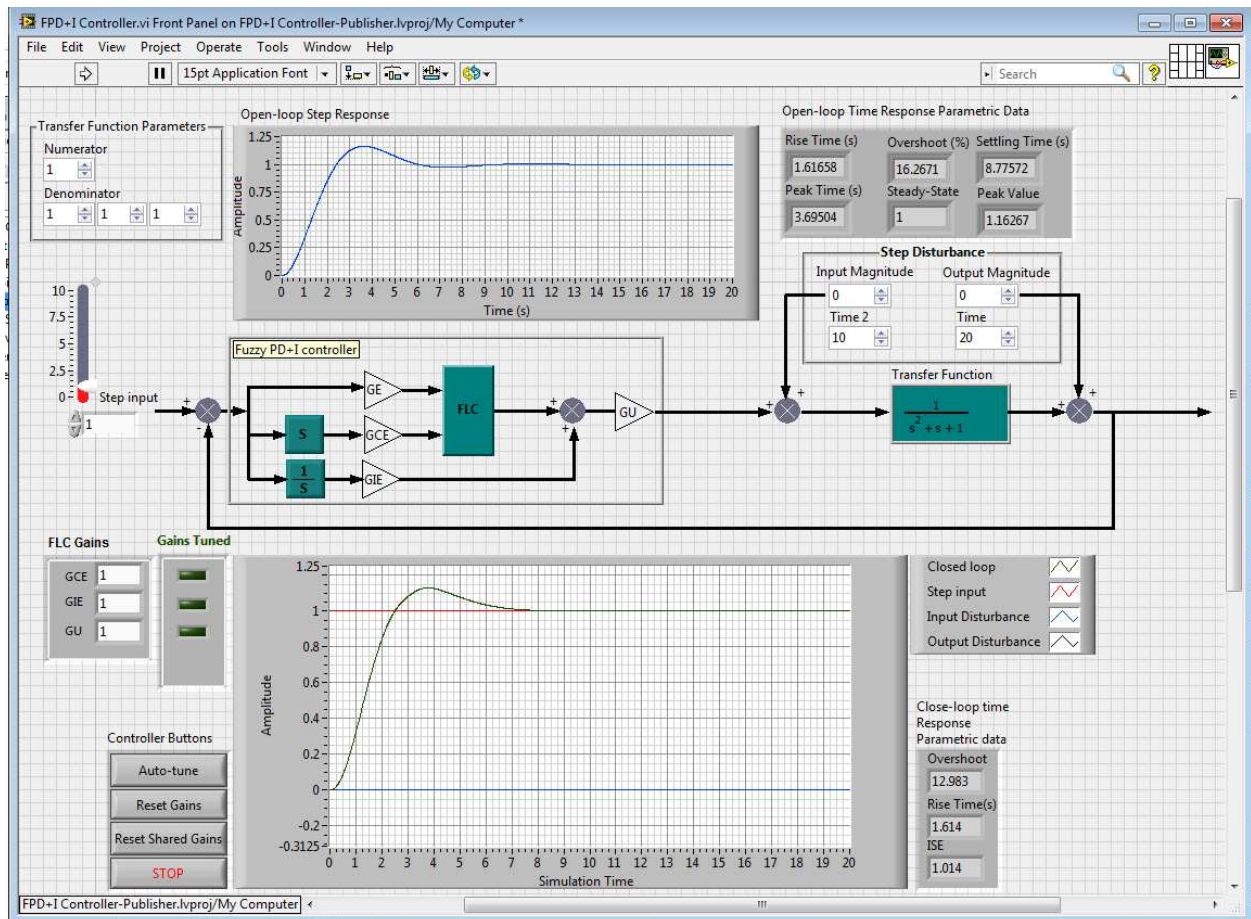
Figure 5-9: The basic FPD+I controller settings and step response of Case 3.

The closed-loop step response above shows that a stable response was achieved with improvements in both the maximum percentage overshoot and the settling time. The overshoot was reduced from 16.26% to 12.983%. The settling time was reduced from 8.77 seconds to 6.60 seconds.

5.2.2 The Auto-tuning Algorithm

Using the same settings shown in Figure 5.9, the auto-tuning algorithm was simulated. The controller input was changed to a unit pulse input, with a period of 30 seconds and a duty cycle of 50%. The controller gains' values were initially set to 1, and the simulation time was set to 30 seconds with a fixed-size (0.01 second) of sampling interval. Finally, the controller was set to auto-tune mode through the auto-tune button on the user interface.

Figure 5.10 shows the closed-loop responses of Case 3, where responses are iteratively shown. The step responses of Case 1, Case 2, Case 4 and Case 5 are shown in Figure D-5 – Figure D-8 in Appendix D.
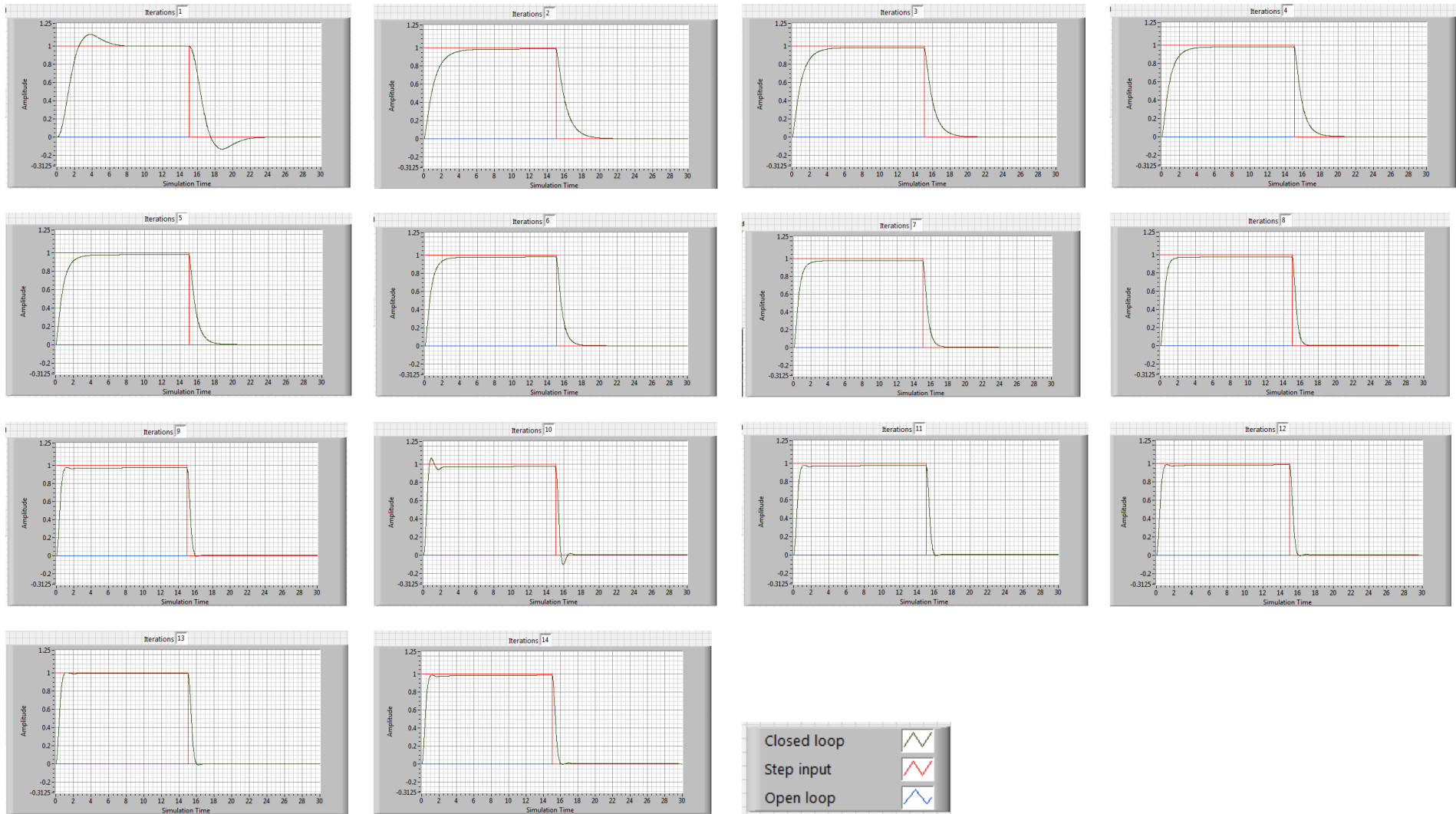
Figure 5-10: Closed-loop step response for Case 3 using the auto-tune algorithm.

As can be seen from the figures above, substantial improvement achieved in the transient and the steady-state responses. The overshoot was eliminated from the second iteration, and then in an iterative manner the rise time and the settling time were improved. In this test, the tuning process was accomplished in 14 iterations.

Table 5.4 lists the controller gains, ISE and the closed-loop characteristics of Case 3 for each iteration. Similar data of Case 1, Case 2 , Case 4 and Case 5 are listed in Table D-1 – Table D-4 in Appendix D.

Table 5-4: Controller gains' values and closed-loop characteristics of Case 3 using the auto-tuning algorithm.

Iteration	Controller Gain				ISE	Closed-loop Characteristics	
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p
1	1	1	1	1	1.014	1.614	12.983
2	1	1	0.039	12.983	0.643	2.414	0
3	1	0.9	0.039	12.983	0.591	2.184	0
4	1	0.8	0.039	12.983	0.539	1.946	0
5	1	0.7	0.039	12.983	0.488	1.699	0
6	1	0.6	0.039	12.983	0.438	1.442	0
7	1	0.5	0.039	12.983	0.391	1.168	0
8	1	0.4	0.039	12.983	0.348	0.871	0
9	1	0.3	0.039	12.983	0.312	0.604	0
10	1	0.2	0.039	12.983	0.293	0.439	6.69
11	1	0.3	0.039	12.983	0.312	0.604	0
12	1	0.3	0.077	12.983	0.306	0.591	0
13	1	0.3	0.154	12.983	0.209	0.570	0.432
14	1	0.3	0.077	12.983	0.306	0.591	0

It can be noted from the data above that the overshoot eliminated and became 0 from the second iteration. This was on account of a rapid change in the value of G_{IE} where it was changed from 1 to 0.039 from the second iteration. The data also show improvements in the rise time starting in iteration 3 where the value of G_{CE} started to decrease while retaining the values of G_{IE} and G_U.

The algorithm uses ISE and *Mp* to monitor the overall performance of the controller and to indicate further changes in the values of G_{CE} and G_{IE}. For instance, in iteration 10, due to a continuous decreasing of the value of G_{CE} the overshoot increased from zero to 6.69. Therefore, to preserve a better performance, in iteration 11 the algorithm reversed to the previous value of G_{CE}, iteration 9, and then started to increase the value of G_{IE}. A similar situation happened again when the overshoot changed from 0 to 0.432 in iteration 13, and then accordingly in iteration 14 the algorithm reversed to the previous value of G_{IE}, iteration 12.

5.2.3 The Wireless Intelligent Fuzzy Controller Network System

To test the wireless intelligent fuzzy controller network system, a network of five wireless-enabled PCs was established through a dedicated wireless router to ensure there is no other network traffic to disturb the communication of the PCs. The structure is shown in Figure 5.11 where instances of the auto-tuning algorithm applied to the basic FPD+I controller were running on PC1-PC4 denoted as ABFPD+I 1 - ABFPD+I 4 (auto-tuned basic FPD+I controller). The PC5 were used to monitor and show the response of each PC and this is denoted as Monitor. As mentioned in Section 4.2, the standard wireless local area network (WLAN) Wi-Fi was used for the communication.

Wireless Router

| ABFPD+I 1 | ABFPD+I 2 | ABFPD+I 3 | ABFPD+I 4 | Monitor |

Figure 5-11: The wireless fuzzy logic controller network structure.

Also in this test, all the second order systems with the transfer functions listed in Table 3.2 were simulated, but only the settings of Case 3 are shown in Figure 5.12. The controllers were setup to simulate Case 3. For each controller, the input was a unit pulse input with a period of 30 seconds and a duty cycle of 50%. The gains were initially set to 1, and the simulation time was set to 30 seconds with a fixed 0.01 second of sampling interval.

A VI was designed to aggregate and show the responses on PC5. A normal response is shown in Figure 5.13.
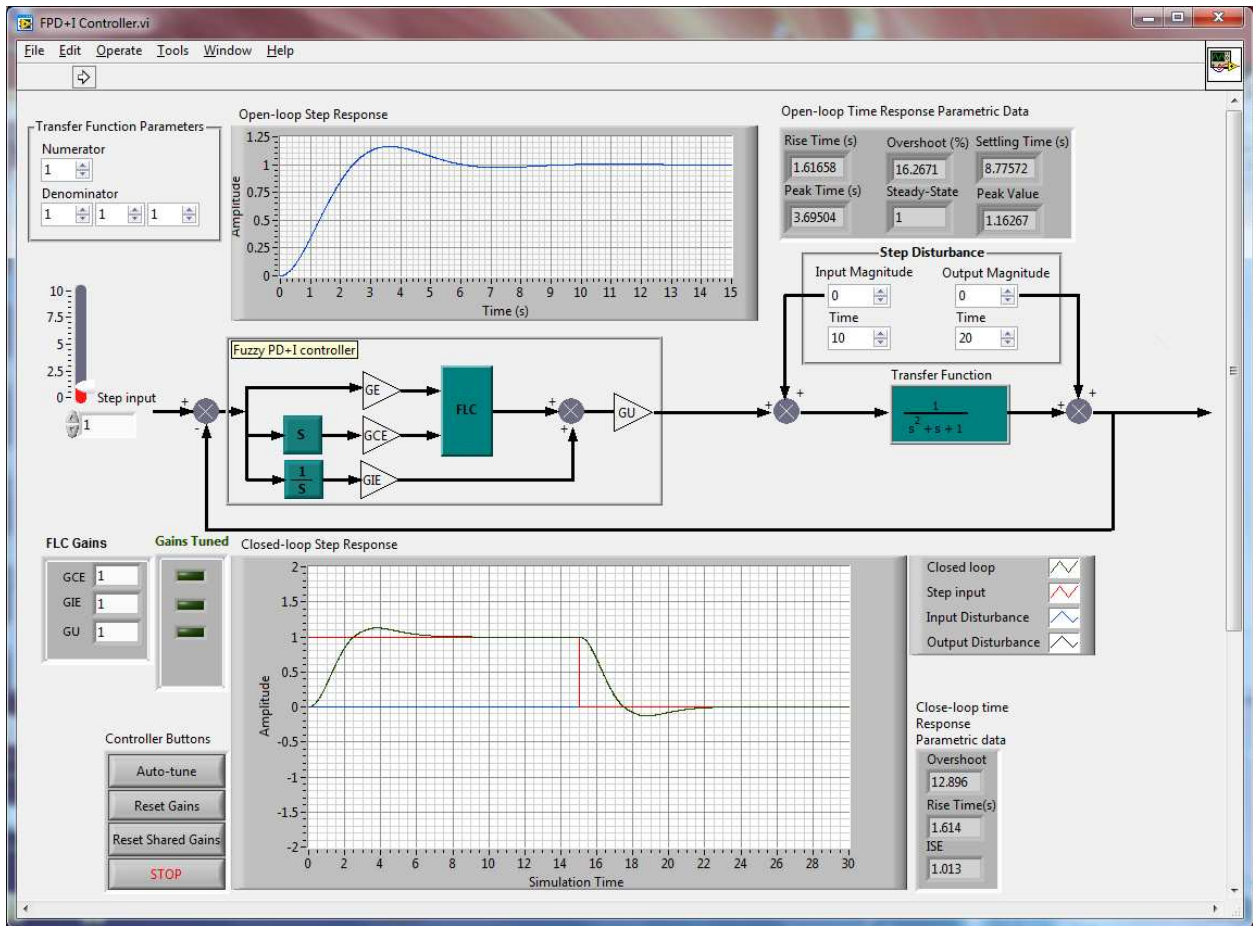
Figure 5-12: The basic FPD+I controller with auto-tuning algorithm settings and step response of Case 3.
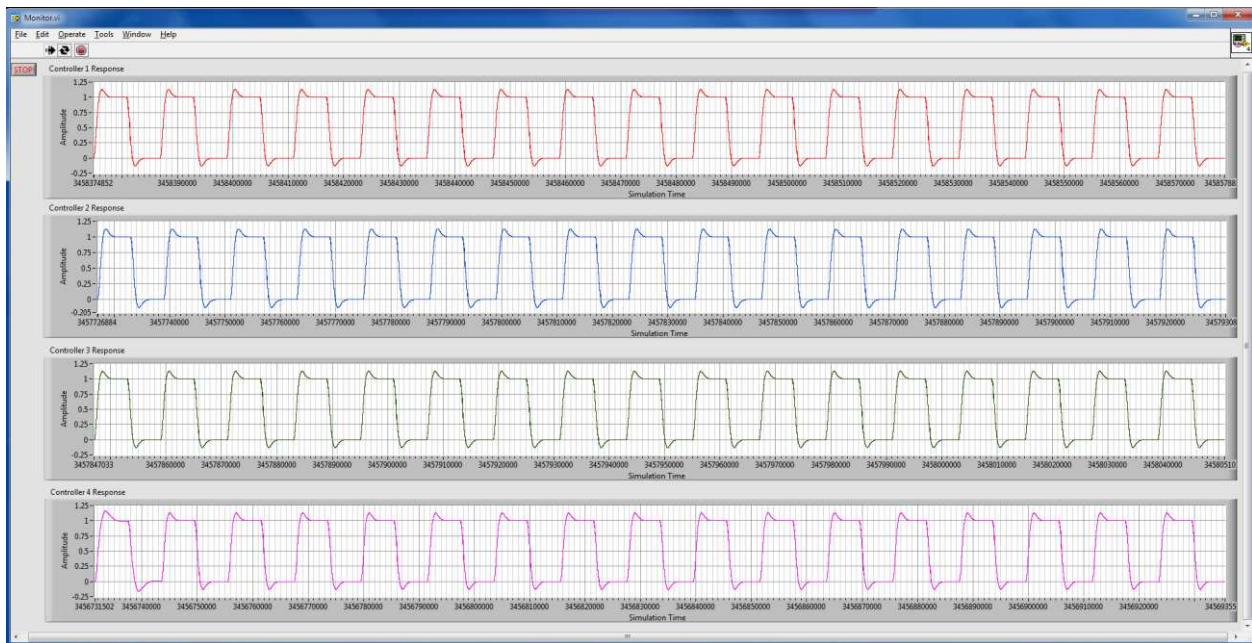
Figure 5-13: Aggregated normal controller responses of Case 3.

After setting up the system, two tests were conducted as follows:

5.2.3.1 *Sharing Capability Test*

This test illustrates the sharing capability of the system. One of the controllers automatically accomplishes the tuning process and then shares the knowledge, the tuned gains values, with the other controllers. The other controllers, which are set to auto-tuning mode, use the tuned gains made available to them by the first controller.

In this test, all the controllers were initiated to operate continuously and then ABFPD+I 1 was set to auto-tune mode. After the completion of the auto-tuning process, ABFPD+I 1 was halted and then ABFPD+I 2, ABFPD+I 3 and ABFPD+I 4 were set to auto-tune mode sequentially. The iterative closed-loop responses are shown in Figure 5.14, where row 1, row 2, row 3 and row 4 show the responses of ABFPD+I 1, ABFPD+I 2, ABFPD+I 3 and ABFPD+I 4 respectively.

Figure 5-14: Controller responses for sharing capability test.

Table 5.5 shows the number of iterations were required by the controllers to tune their gains. Also the values of the gains and the closed-loop characteristics for each iteration are listed.

Table 5-5: Controller gains' values and closed-loop characteristics for sharing capability test.

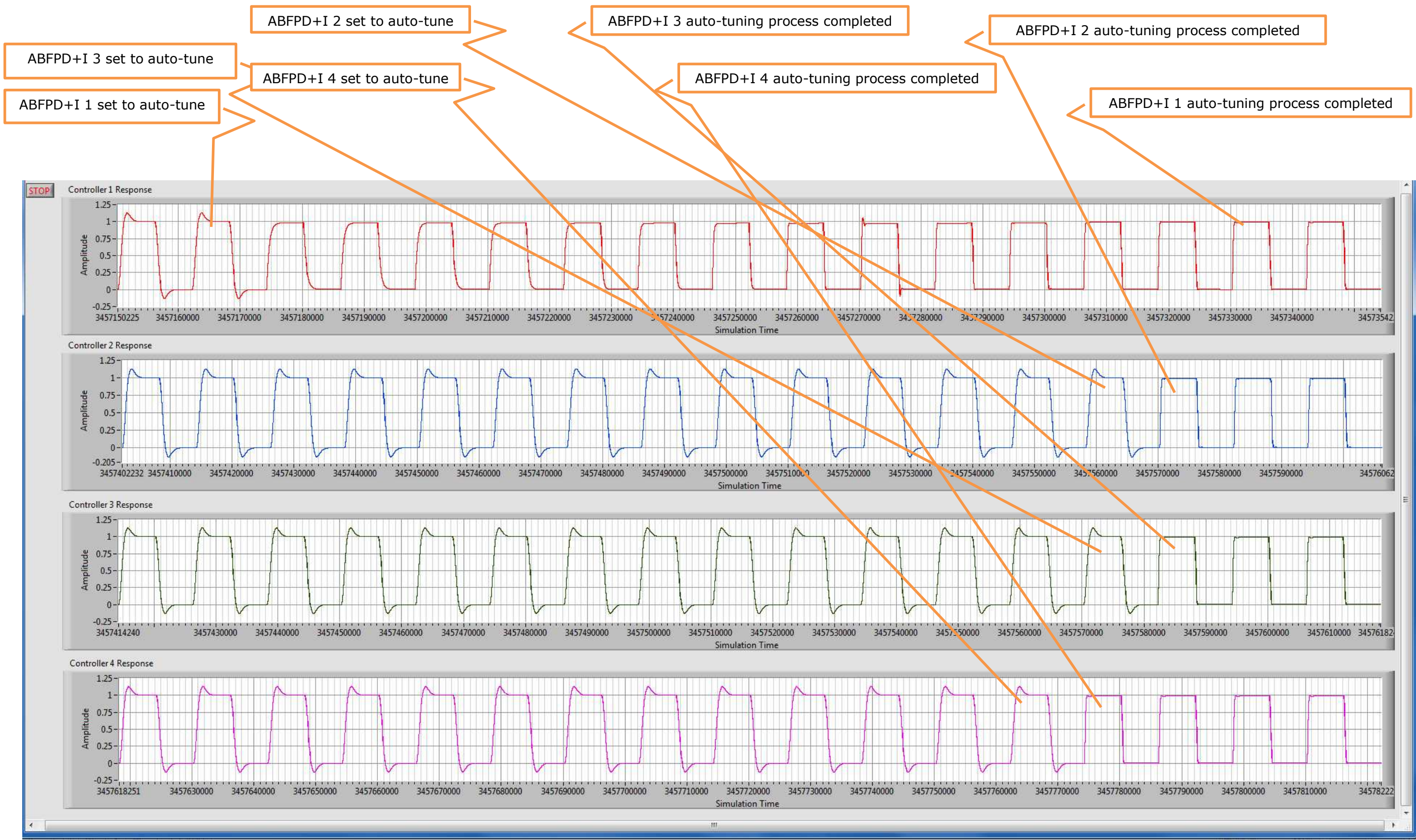| Controller	Iteration	Controller Gain				ISE	Closed-loop Characteristics	
		G_E	G_{CE}	G_{IE}	G_U		t_r	M_p
ABFPD+I 1	1	1	1	1	1	1.013	1.614	12.896
	2	1	1	0.039	12.895	0.642	2.429	0
	3	1	0.9	0.039	12.895	0.59	2.199	0
	4	1	0.8	0.039	12.895	0.538	1.961	0
	5	1	0.7	0.039	12.895	0.487	1.715	0
	6	1	0.6	0.039	12.895	0.438	1.457	0
	7	1	0.5	0.039	12.895	0.391	1.184	0
	8	1	0.4	0.039	12.895	0.348	0.887	0
	9	1	0.3	0.039	12.895	0.313	0.608	0
	10	1	0.2	0.039	12.895	0.293	0.439	6.174
	11	1	0.3	0.039	12.895	0.313	0.608	0
	12	1	0.3	0.078	12.895	0.306	0.595	0
	13	1	0.3	0.116	12.895	0.302	0.583	0
	14	1	0.3	0.155	12.895	0.299	0.572	0.122
	15	1	0.3	0.116	12.895	0.302	0.583	0
ABFPD+I 2	1	1	1	1	1	1.013	1.614	12.896
	2	1	0.3	0.116	12.895	0.302	0.583	0
ABFPD+I 3	1	1	1	1	1	1.013	1.614	12.896
	2	1	0.3	0.116	12.895	0.302	0.583	0
ABFPD+I 4	1	1	1	1	1	1.013	1.614	12.896
	2	1	0.3	0.116	12.895	0.302	0.583	0

As it can be seen from Figure 5.14 and the table above, that ABFPD+I 1 completed the auto-tuning process in 15 iterations, while ABFPD+I 2, ABFPD+I 3, ABFPD+I 4 completed the tuning in 2 iterations and this was as the result of sharing capability of the system. From the tuning of ABFPD+I 1, the gains' values of 0.3, 0.116 and 12.895 for G_{CE}, G_{IE} and G_U respectively were obtained, and then these values were shared with ABFPD+I 2, ABFPD+I 3 and ABFPD+I 4.

When these controllers were set to auto-tune mode, they used the shared gains' values and completed their respective tuning processes in 2 iterations. Therefore, similar performance was achieved for all the controllers. This can be noted from data of columns t_r and M_p, iterations 15 of ABFPD+I 1, and from iteration 2 of ABFPD+I 2, ABFPD+I 3 and ABFPD+I 4 respectively where $t_r = 0.583$ and $M_p = 0$.

5.2.3.2   Cooperative Sharing Capability Test

This test demonstrates the completion of the auto-tuning process by all the collaborating controllers where they, in turn, involve in the process. Part of the tuning is performed by one of the controllers, and then the tuned gains are made available on the network. A next controller resumes the process using the gains received from its predecessor as starting gains. The new tuned gains are then made available again by the controller to its successor. The process continues until the gains are optimally tuned.

To carry out the test, initially the controllers were started to operate continuously. Then each controller was set to auto-tune mode for a period of time and then halted one after another as follows: ABFPD+I 1 was set to auto-tuning mode for 5 cycles and then was halted; ABFPD+I 2 was set to auto-tune mode for 5 cycles and then was halted; ABFPD+I 3 was set to auto-tuning mode for 5 cycles and then was halted; and finally ABFPD+I 4 was set to auto-tune mode for 6 cycles and then was halted. The closed-loop responses of the controllers are shown in Figure 5.15.
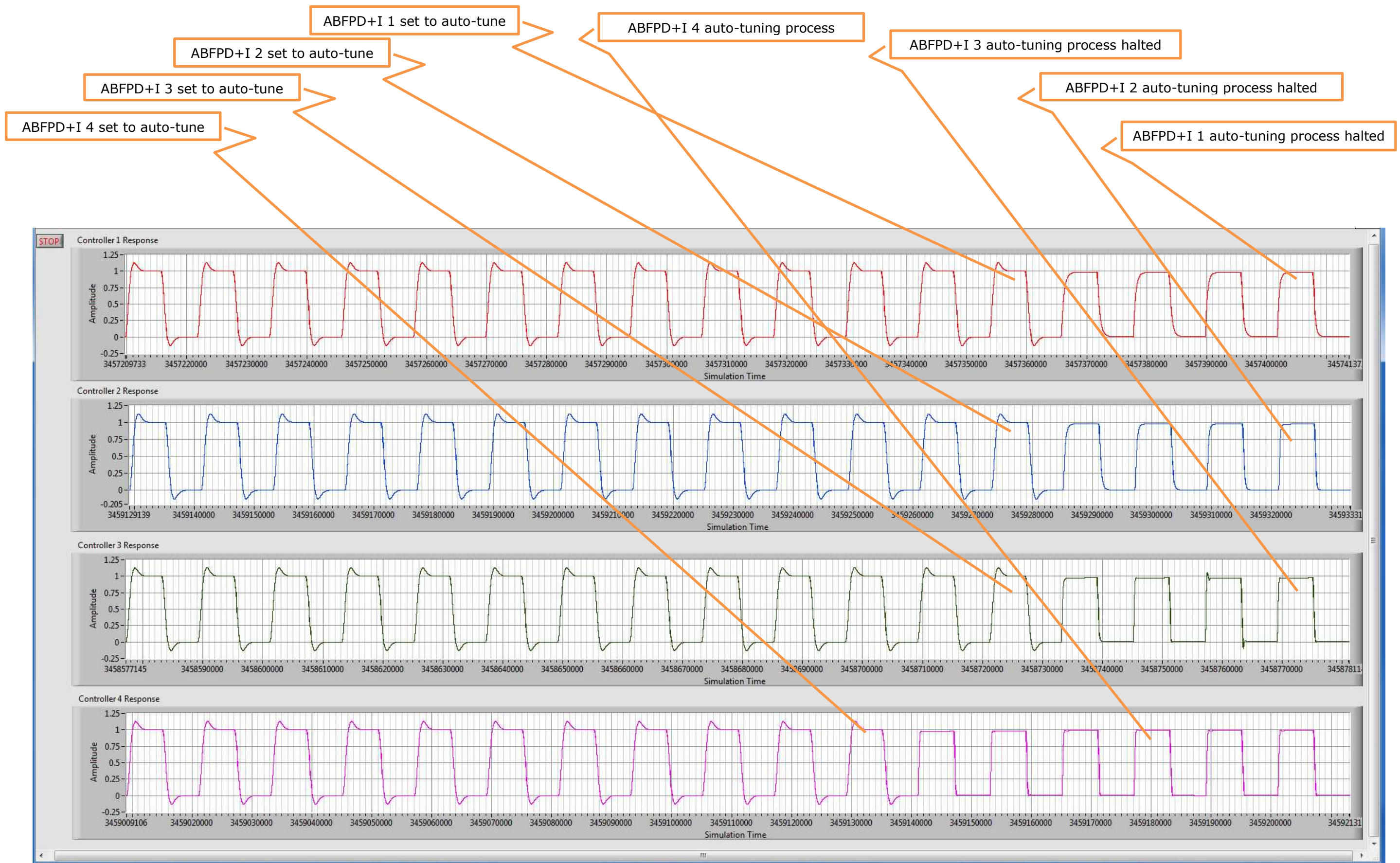
Figure 5-15: Controller responses for cooperative sharing capability test.

117

Table 5.6 shows the number of iterations required by the controllers to tune their gains. Also the values of the gains and the closed-loop characteristics for each iteration are shown.

Table 5-6: Controller gains' values and closed-loop characteristics for cooperative sharing capability test.

Controller	Iteration	Controller Gain				ISE	Closed-loop Characteristics	
		G_E	G_{CE}	G_{IE}	G_U		t_r	M_p
ABFPD+I 1	1	1	1	1	1	1.013	1.614	12.896
	2	1	1	0.039	12.895	0.642	2.429	0
	3	1	0.9	0.039	12.895	0.59	2.199	0
	4	1	0.8	0.039	12.895	0.538	1.961	0
	5	1	0.7	0.039	12.895	0.487	1.715	0
ABFPD+I 2	1	1	1	1	1	1.013	1.614	12.896
	2	1	0.7	0.039	12.895	0.487	1.715	0
	3	1	0.6	0.039	12.895	0.438	1.457	0
	4	1	0.5	0.039	12.895	0.391	1.184	0
	5	1	0.4	0.039	12.895	0.348	0.887	0
ABFPD+I 3	1	1	1	1	1	1.013	1.614	12.896
	2	1	0.4	0.039	12.895	0.348	0.887	0
	3	1	0.3	0.039	12.895	0.313	0.608	0
	4	1	0.2	0.039	12.895	0.293	0.439	6.174
	5	1	0.3	0.039	12.895	0.313	0.608	0
ABFPD+I 4	1	1	1	1	1	1.013	1.614	12.896
	2	1	0.3	0.039	12.895	0.313	0.608	0
	3	1	0.3	0.078	12.895	0.306	0.595	0
	4	1	0.3	0.116	12.895	0.302	0.583	0
	5	1	0.3	0.155	12.895	0.299	0.572	0.122
	6	1	0.3	0.116	12.895	0.302	0.583	0

As it can be seen from Figure 5.15 and the table above, that each controller participated in the auto-tuning process, and therefore the process was completed in 21 iterations. The gain values of ABFPD+I 2, ABFPD+I 3 and ABFPD+I 4 in their second iterations were set to the values obtained from iteration 5 of their respective predecessors ABFPD+I 1, ABFPD+I 2 and ABFPD+I 3 respectively.

Although the process required more time, 21 iterations compared to 15 in the previous test, the test still illustrates the significance of the sharing capabilities of the controllers.

5.3 Hardware-in-the-loop Tests

In this section the results of the application of the basic FPD+I and the auto-tuning controllers to real-time systems are presented. The real-time systems were second order systems constructed using op-amps and RC components presented in detail in Section 4.8.

For comparison purposes the hardware implementation of Case 3 was chosen and henceforth it is referred to as the hardware circuit. The implemented and constructed system shown in schematic diagram in Figure 4.17 and the values of the RCs presented in Table 4.2 for Case 3 were used.

5.3.1 Open-loop Step Response

A VI was designed to conduct an open-loop step response test on the hardware circuit and to measure the maximum percentage overshoot (M_p) and the rise time (t_r). The result is shown in Figure 5.16.
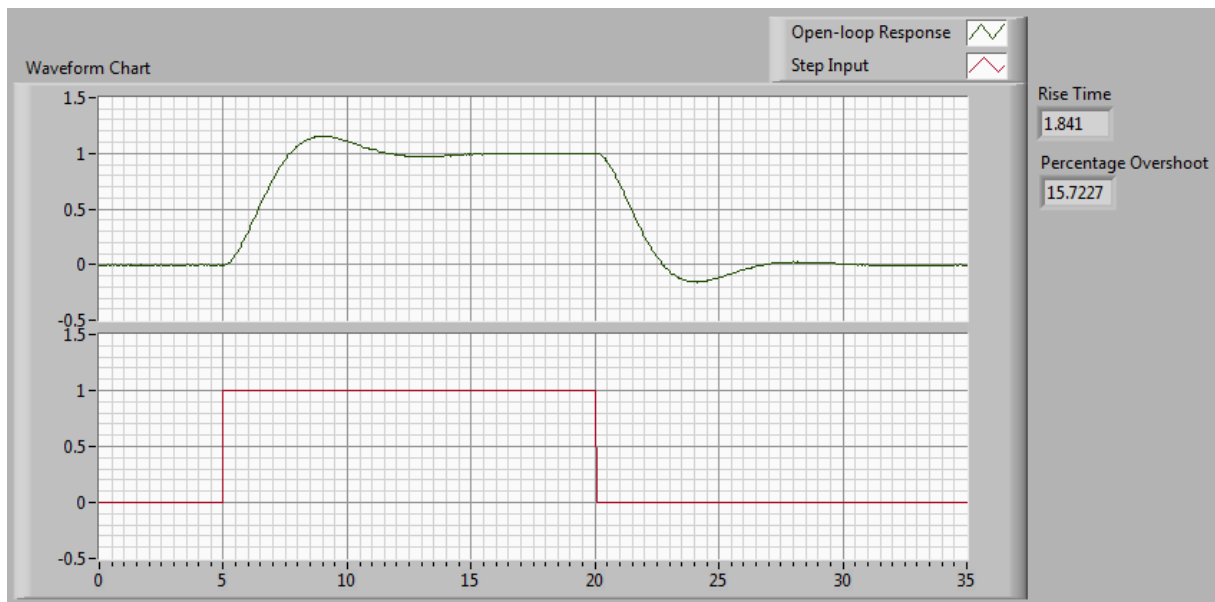
Figure 5-16: Open-loop step response of the hardware circuit using the basic FPD+I controller.

As can be seen from the above figure that the open-loop step response of the hardware circuit is stable with 15.72% overshoot and 1.841 seconds rise time.

5.3.2 The Basic FPD+I Controller

The basic FPD+I controller VI presented in Figure 4.18 was used to evaluate the output response of the hardware circuit. The controller input was a unit pulse input. The controller gains were set to 1 and the sampling interval was set to 0.05 second. The closed-loop response is shown in Figure 5.17.
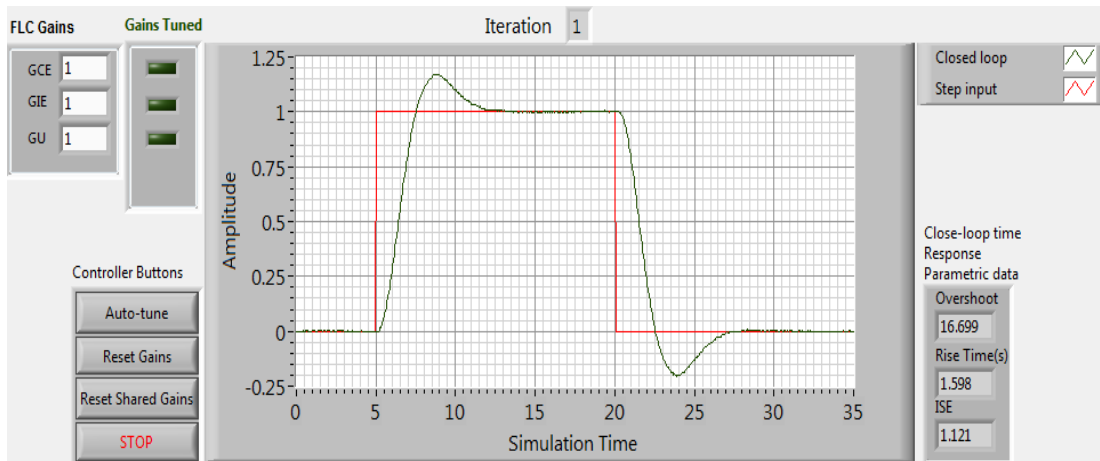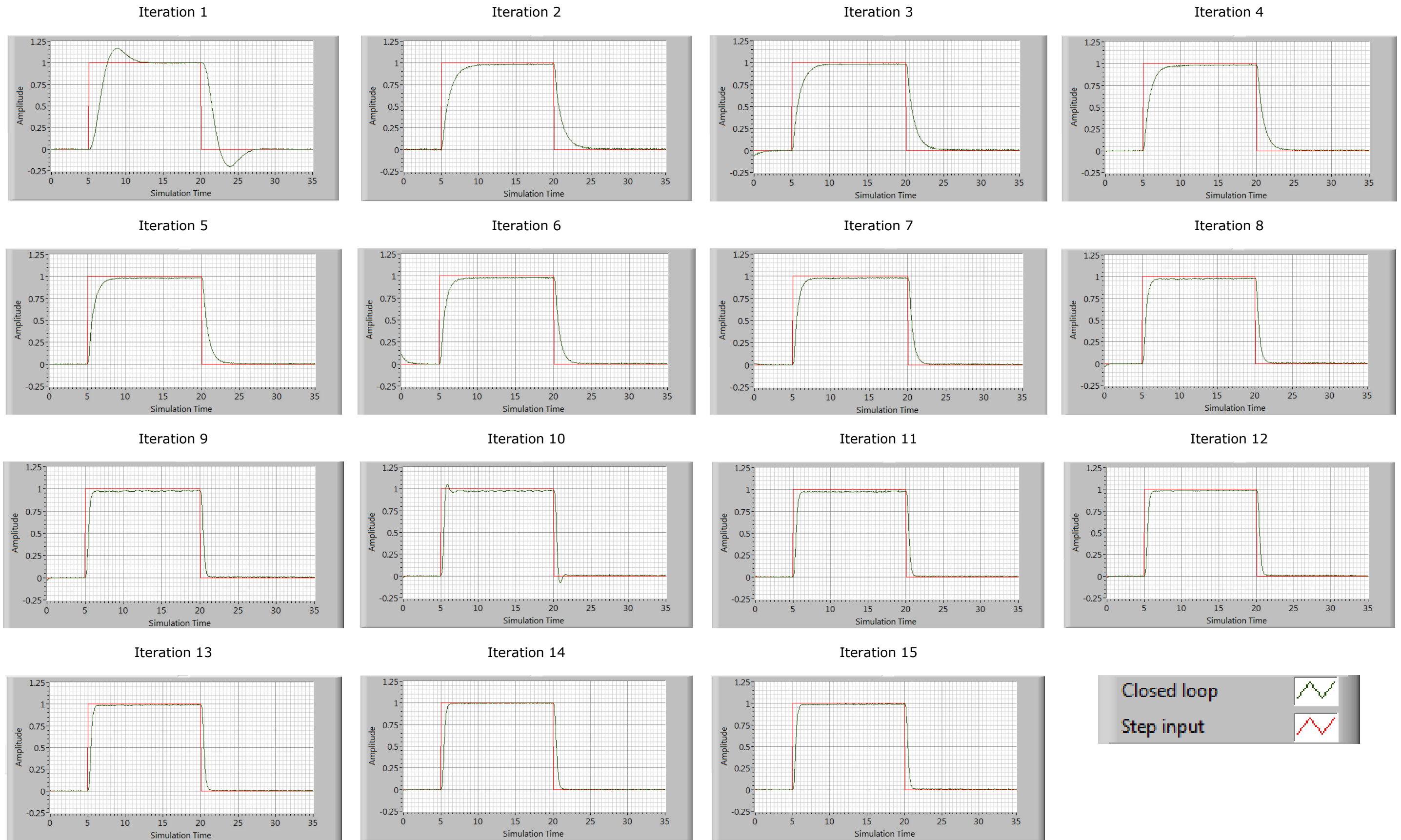
Figure 5-17: Closed-loop step response of the hardware circuit using the basic FPD+I controller.

Figure above shows a stable closed-loop step response with improvement in the rise time where it was reduced from 1.84 seconds to 1.59 seconds. However, the overshoot increased from 15.72% to 16.69%.

5.3.3 The Auto-tuning Algorithm

To evaluate the output response of the hardware circuit, the same controller VI in the previous section with the auto-tuning algorithm settings was used. The controller gains were initially set to 1 and then the controller was set to auto-tuning mode.

Figure 5.18 shows the closed-loop responses, where responses of 15 iterations are presented from left-to-right top-to-bottom. Table 5.7 lists the controller gains, ISE and the closed-loop characteristics of each iteration.

Figure 5-18: Closed-loop step response of the hardware circuit using the auto-tuning algorithm.

From the figures above it can be seen that the application of the auto-tuning algorithm to the hardware circuit also resulted in substantial improvements of the transient and the steady-state responses. The performance significantly improved from the second iteration by eliminating the overshoot and then in an iterative manner, within 15 iterations, the rise time and the settling time were also improved.

Table 5-7: Controller gains' values and closed-loop characteristics from the application of the auto-tuning algorithm to the hardware circuit.

Iteration	Controller Gain				ISE	Closed-loop Characteristics	
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p
1	1	1	1	1	1.121	1.598	16.67
2	1	1	0.03	16.67	0.578	2.363	0
3	1	0.9	0.03	16.67	0.563	2.168	0
4	1	0.8	0.03	16.67	0.504	1.889	0
5	1	0.7	0.03	16.67	0.466	1.674	0
6	1	0.6	0.03	16.67	0.457	1.44	0
7	1	0.5	0.03	16.67	0.409	1.189	0
8	1	0.4	0.03	16.67	0.366	0.903	0
9	1	0.3	0.03	16.67	0.333	0.594	0
10	1	0.2	0.03	16.67	0.317	0.408	5.469
11	1	0.3	0.03	16.67	0.336	0.596	0
12	1	0.3	0.06	16.67	0.327	0.588	0
13	1	0.3	0.09	16.67	0.325	0.573	0
14	1	0.3	0.12	16.67	0.324	0.556	0.586
15	1	0.3	0.09	16.67	0.321	0.574	0.098

It can be seen from the data that the overshoot, M_p, was eliminated and became zero from the second iteration. This was due to a rapid change in the value of GIE where it changed from 1 to 0.03 from the second iteration. The data also show improvements in the rise time,

t_r, beginning in iteration 3 where the value of G_{CE} started to decrease while retaining the values of G_{IE} and G_U.

5.4 Discussion of Results

The system transfer functions listed in Table 3.2 represent various real-time control systems which include critically damped, overdamped, underdamped and unstable systems.

From the MATLAB simulation results of the application of the basic FPD+I controller to abovementioned systems, Figure 5.1 and Figures C13-C16 in Appendix C, it is apparent that the controller was successful in stabilising all the systems. This is more clearly evident in Case 4 and Case 5 unstable systems. In addition, the maximum percentage overshoot reductions were proportional to the damping radio of these systems.

However, a significant overshoot is noted for the critically damped and overdamped systems, Case 1 and Case 2, where these systems are stable in their open-loop responses and exhibit no overshoots. The addition of the integral action in the structure of the controller as shown in Figure 3.4 in Section 3.5.1, has such an effect and might cause the system susceptible to more oscillation and possibly instability as is the case in conventional PID controllers (Åström & Murray, 2009).

The responses obtained from the LabVIEW based tests of the basic FPD+I controller are similar to the results obtained from MATLAB based tests. These are shown in Figure 5.9 and Figures D1-D4 in Appendix D.

Furthermore, the practical application of the FPD+I controller to the hardware circuit representing Case 3 system as is shown in in Figure 5.17, validates the results obtained from MATLAB and LabVIEW platforms. This can be compared in Figure 5.17, Figure 5.1 and Figure 5.9.

These results above indicate that the basic FPD+I controller is capable of controlling and stabilising most of the second order systems.

In the case of the auto-tuning algorithm, the MATLAB results presented in Figure 5.2 and Figures C17-C20 in Appendix C, indicate considerable improvements in the transient and the steady-state responses of all the systems. The improvements were achieved progressively in approximately 15 iterations. The final iteration resulted in a zero-overshoot response and reduced the rise time and the settling time. The number of the iterations is subject to change depending on the step sizes of G_{CE} and G_{IE}. For instance, the smaller the step size, the longer time is required by the algorithm to accomplish the tuning process.

Similar results from LabVIEW, Figure 5.10 and Figures D5-D8 in Appendix D, and from the practical application of the auto-tuned fuzzy controller to a hardware circuit, Figure 5.18, validates the abovementioned findings.

For the comparison purposes, the results obtained from the application of the auto-tuning algorithm to Case 3 using MATLAB, LabVIEW and the hardware circuit are presented in Table 5.8.

Table 5-8: Number of iterations, controller gains' values and closed-loop characteristics of Case 3 using three different platforms.

Test	No. of iterations	Controller Gain			ISE	Closed-loop Characteristics	
		G_{CE}	G_{IE}	G_U		t_r	M_p
MATLAB	14	0.3	0.079	12.730	0.312	0.598	0
LabVIEW	14	0.3	0.077	12.983	0.306	0.591	0
Hardware Circuit	15	0.3	0.09	16.67	0.321	0.574	0.098

When evaluating the sharing capability of the wireless intelligent fuzzy controller network, it is shown how the gained knowledge in tuning a controller can be shared using the network. The recipient controllers can benefit from the knowledge and apply it to auto-tune their gains and achieve a desirable response in much less time than it is normally required. This can be noted in Figure 5.14, where ABFPD+I 1 completed the tuning process in 15 iterations whilst each of ABFPD+I 2, ABFPD+I 3 and ABFPD+I 4 completed the same process in 2 iterations.

5.5 Summary

The chapter has presented details of various tests and showed the results of the application of the basic FPD+I controller and the auto-tuning algorithms to several second order systems using MATLAB and LabVIEW platforms. The results have been encouraging and indicate the validity of the technique where the performance of the system response progressively improves. The results have also shown that the auto-tuning algorithm is highly effective in achieving a zero overshoot response and produced a faster transient response.

In addition, the algorithm has been tested on real systems using hardware circuits modelling various processes.

Furthermore, the sharing capability and the functionalities of the wireless intelligent fuzzy controller network system have been successful in exchanging data whenever required.

CHAPTER 6

Conclusions and Further Work

In the coming sections, some conclusions are drawn on the work undertaken in this thesis and the main contributions are presented. Also some potential suggestions are outlined.

6.1 Conclusions

As discussed in Section 2.3 in the Literature Review chapter, fuzzy logic systems have a main role in the design of intelligent controllers and this is due to the fact that the human knowledge in understanding, modelling and controlling the dynamics and complex characteristics of a process is systematically integrated into a controller.

Furthermore, the discussion has also given an account of the limitations of the static representation of knowledge in such controllers. In Sections 2.4 and 2.5, a detailed attention has been paid to various difficulties associated with the design of a general purpose fuzzy controller and also with an auto-tuning algorithm. Consequently, the relationship between the controller parameters and the controller performance is not a straightforward task. Even with the existence of suitable algorithms, the advantage of the efforts made in tuning process in a controller is limited, as the knowledge is not retained locally for later use, and neither shared with other identical controllers in a networked environment.

In this context, the present thesis was set out to develop and implement an intelligent algorithm for FCs. Through memory and extensive communication attributes, the algorithm

enables a controller to auto-tune its gains, as well as to retain and share knowledge. The work also focused on the design and establishment of a framework for learning in a wireless network of FCs.

The application domain of the algorithms was a standard second order system, as many real-time applications exhibit oscillation and overshoot in their step responses, and these characteristics can be easily modelled using a second order transfer function. Parts of the systems were constructed using electronic components and then the algorithms were tested on the produced systems.

The auto-tuning algorithm was tested in a simulated wireless and wired network environment. The results demonstrate the effectiveness of the auto-tuning algorithm where the performance of the controller is progressively improved, and the algorithm produced a step response system with a zero overshoot. The results of the simulation-based experiments and the hardware-circuits also show the robustness of the algorithm in terms of step input tracking response and reduction of disturbances.

The proposed structure designed and presented in Chapter 3 and implemented in Chapter 4, showed the feasibility of the system and illustrated the capabilities of the controllers with regard to retaining and sharing the knowledge.

Based on the results and discussions presented in Chapter 5, the following conclusions and findings are drawn:

- Fuzzy logic controllers have a set of parameters, such as those associated with the membership functions, the rule-base and the gains. Each set might have complex or conflicting effects in terms of achieving various time-domain characteristics, such as overshoot, rise time and settling time. The effect of these additional variables on the system is twofold. On the one hand, the number of variables is higher in comparison

with PID controllers, where normally there are only three main parameters to adjust. However, there are alternative possibilities that utilise different techniques to devise various tuning algorithms. On the other hand, the tuning process becomes more complicated as relationships between these parameters and controller performance characteristics become more complex.

It is worth noting that concentrating on a few parameters in the tuning process minimises the complexity of the process and, consequently, less resources such as controller time and memory are required. It is also easier to determine relationships between a small number of parameters and the closed-loop characteristics. In addition, an algorithm with lesser degrees of complexity becomes more practical.

The auto-tuning algorithm devised in this research, presented in Chapter 3, possesses three significant features. Firstly, the tuning process involves only three parameters, namely G_{CE}, G_{IE} and G_U, and it is performed systematically. Secondly, relationships between the controller gains; G_{IE} and G_U, and M_p and t_r are determined. The values of G_{IE} and G_U are related to overshoot and they have significant effects on reducing the overshoot and the steady state error. The value of G_{CE} defines the speed of the response and also affects the stability of the system. Finally, as the PID controller gains have a clear and physical meaning, the auto-tuning algorithm has also given such meanings to the FPD+I controller gains.

- Determining appropriate values of controller gains, to achieve possible optimal responses, can be addressed as a search problem, whereby the gains are adjusted to minimise the value of a performance index, such as integral of squared error or integral of absolute error. However, minimising a single performance index does not always achieve an optimal response that satisfies all the required closed-loop characteristics, such as overshoot, rise time and settling time. Therefore a new

performance index is composed from all the closed-loop characteristics, each with a weight factor proportionate to their importance in a specific control system.

In the development of the auto-tuning algorithm, the values of G_{CE} and G_{IE} gains were determined by minimising the integral of squared error while monitoring the system output for possible overshoot and this is presented in Section 3.5.2. Although a single performance index was not formulated, the search was indirectly based on the combination of the integral of squared error and the overshoot. This finding would empirically help to compose a new performance index. A comparison process based on only one factor is more straightforward and is performed much faster than those based on two or more variables.

- The step sizes of the changes in G_{CE} and G_{IE} gains have a significant impact on the number of iterations required to tune the controller. In the design of the auto-tuning algorithm, after initial determination of the values of G_U and G_{IE} gains, the step sizes are set as follows: G_{CE} is set to decrease in a step size of 0.1, one tenth of the G_{CE} value, and G_{IE} is set to increase in a step size equal to twice the value of G_{IE} and then in each subsequent steps this step size is doubled again. This can be seen in the result of the auto-tuning algorithm test in Table 5.1. Decreasing the step sizes of G_{CE} and G_{IE} leads to an increase in the number of iterations required to tune the parameters, but it also achieves a better performance. However, increasing the step sizes has the opposite effect. This can be seen from that data in Table 5.2 and Table 5.3 where the number of iterations became 23 and 10 respectively.

The current step sizes of the gains were extensively investigated so that a trade-off is achieved between the number of iterations needed to tune the controller and the demanded closed-loop characteristics in order to achieve the required response in a reasonable time.

6.2 Contributions of the Thesis

This section discusses the main contributions of this research work outlined in the Introduction chapter and presents the significance of the research to the relevant fields.

6.2.1 A Systematic Auto-tuning Algorithm

The main contribution of this research work is the development of a new auto-tuning algorithm. The algorithm is performed systematically and it is based on the tuning of a few parameters. Using this approach ensures its straightforwardness and consequently increases the scope of its application.

More importantly, achieving a closed-loop step response with a zero overshoot provides potential virtues to control systems where such a feature is demanded. In some industrial process control systems, it is desirable not to allow an overshoot beyond the setpoint or a threshold and this could be a safety constraint or the requirement of the system.

6.2.2 A New Structure for an Intelligent Fuzzy Logic Controller

Whilst incorporating human knowledge into a controller might not be sufficient in designing an intelligent controller, the main feature of an intelligent controller still remains in the capability of learning from past experience. Through the design of a multi-layer structure algorithm, discussed and presented in Section 3.3 and implemented in Section 4.4, the new fuzzy controller encapsulates the abilities of self-learning and knowledge retaining. The acquired experience and knowledge in the tuning process is retained and then utilised whenever a similar situation arises.

This new structure allows fuzzy controllers to integrate extra knowledge to their existing rule-base knowledge system.

6.2.3 A New Knowledge Sharing Framework

Using LabVIEW, the thesis introduced and implemented a framework for networked FCs. This new framework provides a learning facility through enabling effective sharing of knowledge between identical FCs. The structure could be used as a new paradigm for implementing practical multiple fuzzy controller systems.

6.3 Recommendations for Further Work

The following are some recommendations that could provide directions for further work.

- As the main feature of fuzzy logic control systems is to incorporate control strategies in the form of linguistic terms, it is also suggested to use fuzzy logic to integrate the tuning procedure in the form of the 'If-then' expressions. Although this might add further computation time for the overall control system, it is worth researching the point. In the new case, fuzzy logic can be utilised in the control strategy and in the tuning procedure of the controller which results in a multiple layer of fuzzy logic control systems.

- A wide range of physical control systems can be modelled using the standard classes of second order systems, such as critically damped, overdamped and underdamped classes. For instance, the first two forms are often found when two first order systems are connected in a series. The underdamped forms exist in some mechanical and fluid systems (Edgar et al., 2004). The algorithm was successfully tested and produced satisfactory responses for the abovementioned systems. In addition, unstable and undamped cases were included.

  It is recommended that further research needs to be undertaken in the application of the algorithm to higher order systems and systems with dead time. This widens the scope of the application of the algorithm.

- Since the amount of data exchanged between the controllers is small, mainly the controllers share the gains' values, hence embedded-microcontrollers interfaced to ZigBee modules can be utilised to incorporate the algorithm. Recently, some microcontrollers have been marketed with built-in ZigBee modules (Amtel, 2013; NXP, 2013; Texas Instruments, 2013) and sufficient internal program memory. In addition, these microcontrollers have built-in Analogue to Digital Converters (ADC), Digital to Analogue Converters (DAC) and Pulse Width Modulation (PWM) interfaces to realise a control loop. These microcontrollers are also physically small enough to be embedded in the sensor enclosures. This will provide a new generation of applications that include versatile and cost-effective solutions in wireless sensor networks and control applications.

- The principle of distributed agents is well-established in the field of computer science and it has also been recognised in control engineering. There are several software platforms (Fabio, Giovanni, & Dominic, 2007) to develop and simulate multiple controllers or multi-agent systems, such as Java Agent DEvelopment Framework (JADE) (JADE, 2013), Intelligent Agents known as JACK (AOS, 2013), and Agent Development Kit (ADK) (Tryllian, 2013). Whilst in practical implementation their usages are usually limited to manufacturing systems, a standard platform for implementing such scenarios does not exist. As the proposed and implemented structure presented in this thesis has also shown the feasibility of LabVIEW for simulating and implementing multi-controller structures, it is suggested that further investigation be carried out to explore the capabilities of LabVIEW becoming the standard as a development and implementation platform for multi-controller systems. The burden of developing communication protocols in LabVIEW is enormously reduced as the main communication protocol suit TCP/IP is efficiently provided.

- The proposed framework cannot be limited to fuzzy logic controllers and therefore it could also be adapted to incorporate other control strategies such as conventional PID controllers.

Finally, in spite of what has often been reported about the difficulties of developing an intelligent auto-tuning algorithm for fuzzy logic controllers, the findings in this research have demonstrated the ability of a controller to learn from its own and others' past experiences.

As learning from past experience has always been a significant quality for human beings, likewise it would be for controllers.

Appendices

A. A Design Example of a Fuzzy Logic Controller

To illustrate different terms related to fuzzy control system, a basic design example is given. It is assumed that a standard second order process with the open loop step response shown in Figure A.1 is intended to be controlled by a fuzzy controller.
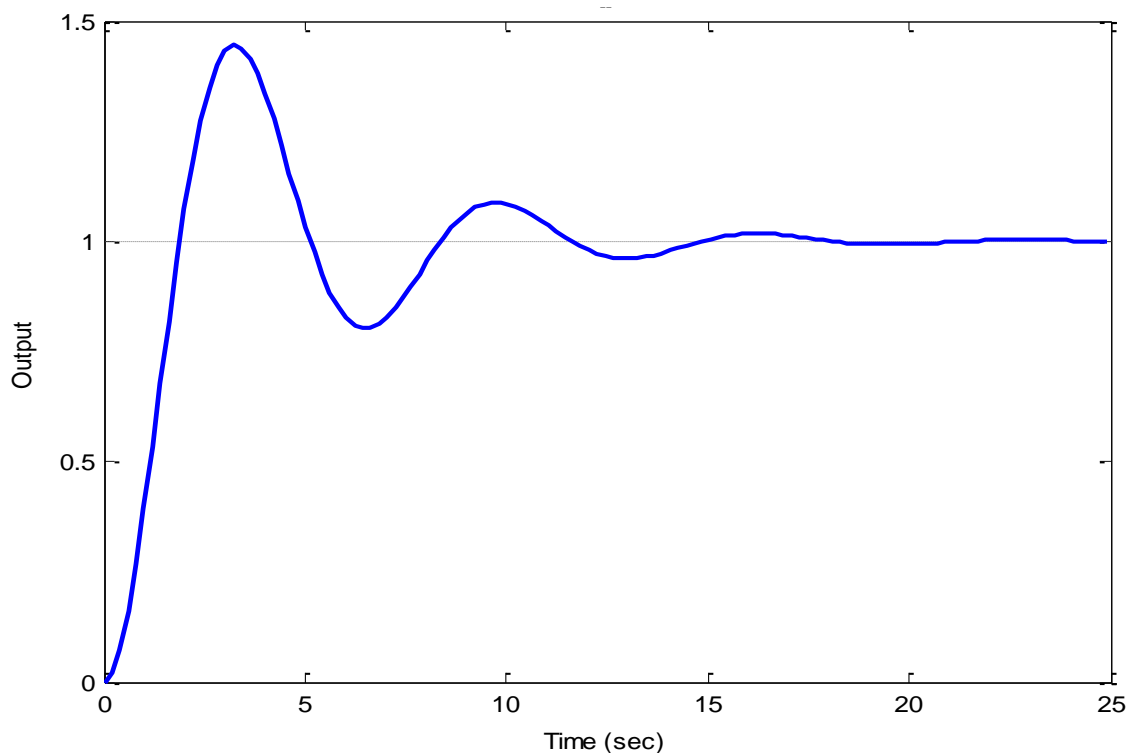
Figure A-1: Open-loop step response of a second order system.

Figure A.2 shows the block diagram of a closed-loop system, where a fuzzy controller is in direct control of a process. The system has one reference input r(t) and one output y(t). The aim is to maintain the output of the process as close as possible to the reference input. This is achieved by producing a suitable control signal u(t) from the controller based on the error e(t) and the change of error de(t) (derivative of error). As the control signal is proportional and related to two input signals, the controller is referred to as a fuzzy proportional-derivative (FPD) controller (Mamdani, 1999). The controller has three gains: gain of the

error (G_E), gain of the change of error (G_{CE}) and output gain (G_U). By adding these gains, the two inputs of the controller become E(t) and dE(t) and its output becomes u(t). The gains are also known as scaling factors (SFs) (Jantzen, 2007; Passino & Yurkovich, 1998), because they are used to scale or transform the real controller inputs and outputs to predefined and normalised universes of discourse of the controller. The scaling gains are supposed to be, theoretically, constant parameters, but normally are used to tune the performance of the controller analogous to the tuning of PID controller gains (Yung-Yaw & Chiy-Ferng, 1994). Throughout the remainder of this thesis, the scaling gains are referred to as gains.
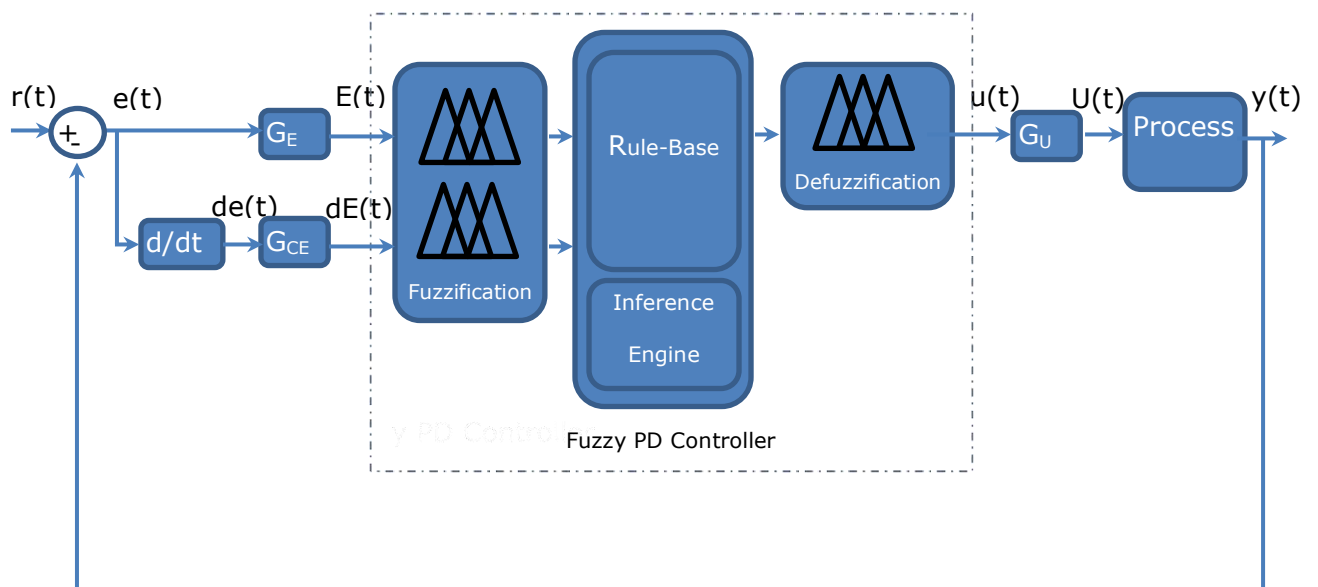
Figure A-2: A fuzzy controller in a closed-loop control system.

The fuzzy PD controller comprises of the following four components (Driankov, Hellendoorn, & Reinfrank, 1996; Passino, 2005; Passino & Yurkovich, 1998):

    1- Fuzzification.

This unit converts, fuzzifies, the inputs: E(t) and dE(t) and the output u(t) to information in the form of linguistic terms, known as fuzzy variables (Zadeh, 1996), where later on they

can be used by the rule-base unit to produce the required rules to control the process. The fuzzy variables take a set of values also in the form of linguistic terms, called fuzzy values or fuzzy sets. For example, the error signal E(t) can be represented by a set of three fuzzy sets such as Negative, Zero and Positive. Each fuzzy set can take a range of values of the represented variable. In this example, the same fuzzy sets are used for E(t), dE(t) and u(t).

To quantify the range, the fuzzy sets are mathematically represented through membership functions (MFs). There are various types of MFs that can be used such as triangular, trapezoidal, Gaussian and singleton (Engelbbrecht, 2007). As symmetric shapes are computationally efficient for calculating their areas they are the most commonly used in the design of fuzzy controllers (Altas & Sharaf, 2007; Passino & Yurkovich, 1998). Figure A.3 shows one triangular and two trapezoidal MFs assigned to each of the inputs, and three symmetric triangular MFs assigned to the output.

As can be seen from Figure A.3, (a) and (b), the Negative terms can take values from -1 to 0, the Zero terms can take values between -0.5 and 0.5, and the Positive terms can take values from 0 to 1. For the output, Figure A.3 (c), the Negative term can take values from -1 to 0, the Zero term can take values between -0.5 and 0.5, and the Positive term can take values from 0 to 1.

Also from the same figure, (a) and (b), the x-axes represent the values of the inputs in a range known as the universe of discourse (UoD) (Passino & Yurkovich, 1998). Assume that the reference input is 1, therefore the range of the universes of discourse of the inputs and the output is, in this case, [-1 1]. This is because the maximum possible values of the error or the change of error is -1 or 1. The universes of discourse of the inputs and the output are normalised and therefore, the gains are set to 1. The y-axes represent the degree of the membership functions (μ) (Passino & Yurkovich, 1998) and, as can be seen, can take any value from 0 to 1.

137
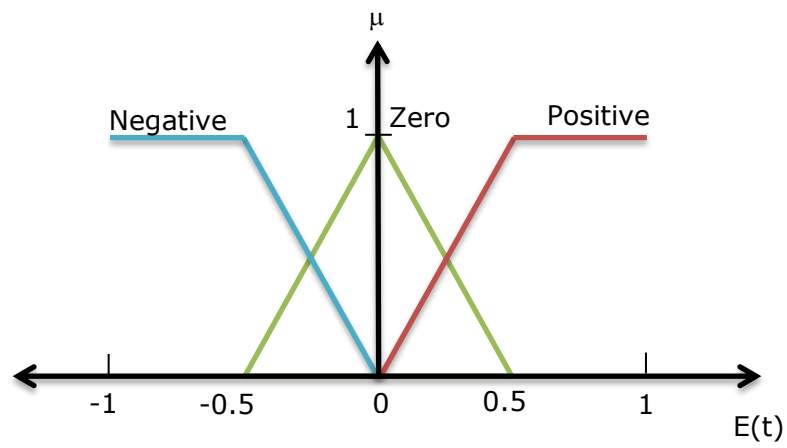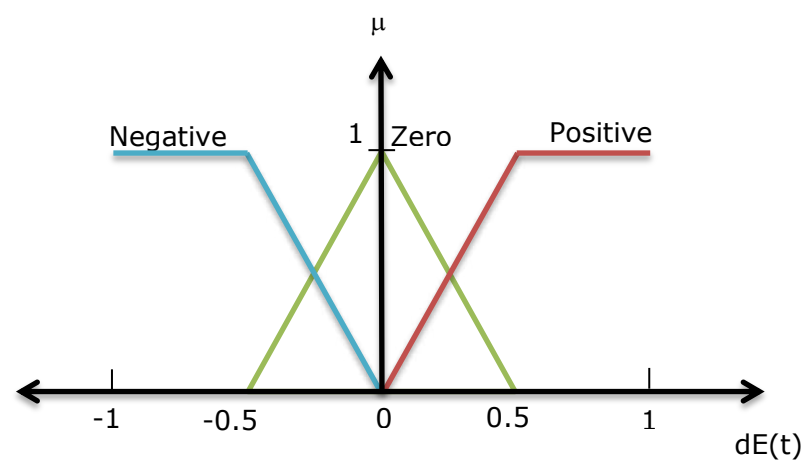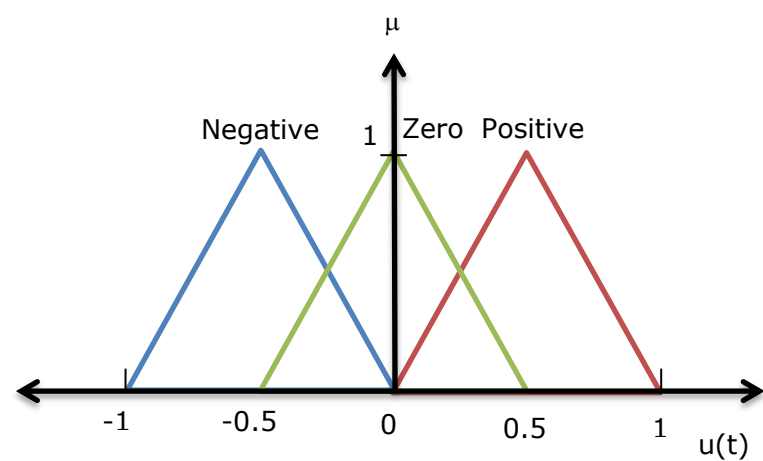
Figure A-3: Membership functions of: (a) Error E(t). (b) Change of error dE(t). (c) Control signal u(t).

2- Rule-Base.

This unit contains the control strategy in the form of rules that instructs the controller on how to control the process. The rules are normally constructed from the knowledge of a skilled operator and are linguistically represented (Mamdani, 1999; Mamdani & Assilian, 1975). A general form of the rules takes the following format:

If input is ..... **Then** output is .....

The **If** part, also known as premises or antecedents, accommodates the input fuzzy variables and their fuzzy values, while the **Then** part, also called conclusions or consequents, accommodates the output fuzzy variables and their fuzzy values (Passino, 2005). The inputs and the output, fuzzified in the previous section, are used to construct the rules. As the controller has two inputs, the logical operator 'and' is normally used to combine them (Passino & Yurkovich, 1998). The inputs and the output have three fuzzy variables; therefore in total, nine rules are possible and they are listed in Table A.1.

Table A-1: Fuzzy Rules.

Rules	Fuzzy Rules
1	**If** E(t) is **Negative and** dE(t) is **Negative Then** u(t) is **Negative**
2	**If** E(t) is **Negative and** dE(t) is **Zero Then** u(t) is **Negative**
3	**If** E(t) is **Negative and** dE(t) is **Positive Then** u(t) is **Zero**
4	**If** E(t) is **Zero and** dE(t) is **Negative Then** u(t) is **Negative**
5	**If** E(t) is **Zero and** dE(t) is **Zero Then** u(t) is **Zero**
6	**If** E(t) is **Zero and** dE(t) is **Positive Then** u(t) is **Positive**
7	**If** E(t) is **Positive and** dE(t) is **Negative Then** u(t) is **Zero**
8	**If** E(t) is **Positive and** dE(t) is **Zero Then** u(t) is **Positive**
9	**If** E(t) is **Positive and** dE(t) is **Positive Then** u(t) is **Positive**

For the purpose of clarity, the rules are listed in a tabular format in Table A.2. The vertical and horizontal headings represent all possible values of the inputs *E(t)* and *dE(t)* respectively, and the cells contain the required control output actions.

Table A-2: A normal Fuzzy PD rule-base.

		dE(t)		
		Negative	*Zero*	*Positive*
E(t)	*Negative*	Negative	Negative	Zero
	Zero	Negative	Zero	Positive
	Positive	Zero	Positive	Positive

3- Inference Engine.

Through using the rule-base, this unit maps the fuzzified inputs to fuzzified outputs for each rule. For example, during the operation of the controller, at a particular time assume that *E(t)* = 0.5 and *dE(t)* = - 0.25. As shown in Figure A.4, the degrees of the membership functions of *E(t)* are as follows: $\mu_{Negative}(0.5) = 0$, $\mu_{Zero}(0.5) = 0$ and $\mu_{Positive}(0.5) = 1$. For *dE(t),* the degrees of the membership functions are $\mu_{Negative}(- 0.25) = 0.5$, $\mu_{Zero}(- 0.25) = 0.5$ and $\mu_{Positive}(-0.25) = 0$.
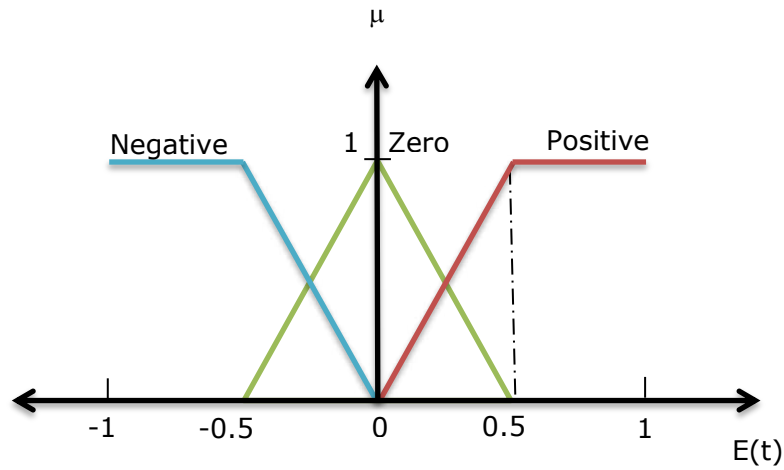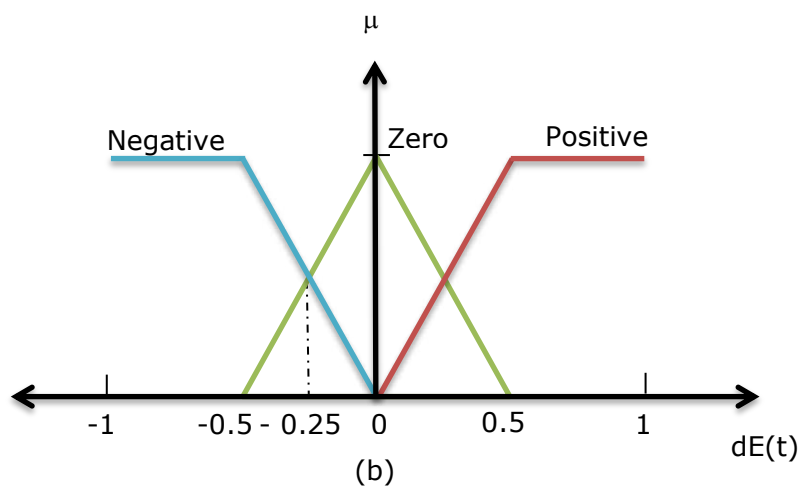
Figure A-4: The degree of the membership functions of: (a) Error E(t).

(b) Change of error dE(t).

From the above calculation, only μ_{Positive} of *E(t)*, μ_{Negative} and μ_{Zero} of *dE(t)* are active. Therefore, only the rules contain Positive term for *E(t)*, and Negative term or Zero term for *dE(t)* are selected. Consequently, only Rules 7 and 8 are applied and they are:

Rule 7: **If** E(t) is **Positive and** dE(t) is **Negative Then** u(t) is **Zero**

Rule 8: **If** E(t) is **Positive and** dE(t) is **Zero Then** u(t) is **Positive**

In fuzzy sets, the 'min' (minimum) operation is normally used to implement the logical 'and' operator in the antecedent part of the rules (Lilly, 2011). By substituting the membership function values and 'min' operator, the strength of active rules are determined as follows:

Rule 7: min{μ_{Positive}(0.5), μ_{Negative}(- 0.25)} = **min**{1, 0.5} = 0.5

Rule 8: min{μ_{Positive}(0.5), μ_{Zero}(- 0.25)} = **min**{1, 0.5} = 0.5

For Rule 7, as the consequent part has Zero term, then Zero MF on the output premise will be truncated at the height of 0.5. Whilst Rule 8 has Positive MF in its consequent part, then Positive MF will be truncated at the height of 0.5. These are shown in Figure A.5.
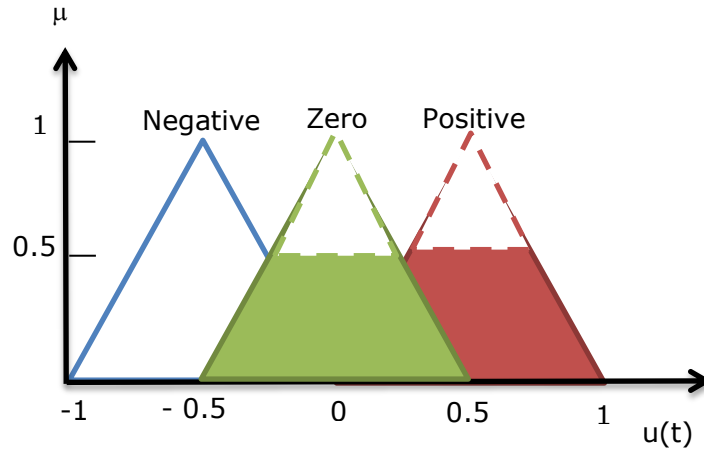
Figure A-5: Truncated output membership functions.

4- Defuzzification.

This unit combines the results of the active rules, Rules 7 and 8 in the previous section, to produce the final control signal u(t). As shown in Figure 5, the shaded areas of Zero and Positive MFs are obtained. The final output value can be achieved by finding the average weight of the shaded areas. There are various methods to compute the average weight, and the most popular is the centre of gravity (CoG) (Passino, 2005).This can be calculated as follows:

$$CoG = \frac{\sum_i Ci\ Ai}{\sum_i Ai} \qquad\qquad (A.1)$$

Where C is the centre of the MFs, A denotes the area of the MF, and i is the number of the MFs. From Figure A.6, C_1= 0 and C_2 = 0.5 for Zero and Positive MFs respectively. As the shaded areas have trapezoidal shapes, their areas are calculated as follows:

$$Area = h * \left(\frac{a+b}{2}\right) \qquad\qquad (A.2)$$

Where, h is the height, a is the length of top, and b is the length of bottom of the trapezoidal shape. As the two shapes are identical, hence;

$$A1 = A2 = 0.5 * \left(\frac{0.5+1}{2}\right) = 0.375 \qquad\qquad (A.3)$$

By substituting the values of C_1, C_2, A_1 and A_2 in Equation (A.1)

$$CoG = \frac{(0*0.375+0.5*0.375)}{(0.375+0.375)} = 0.25$$

Therefore, for $E(t) = 0.5$ and $dE(t) = -0.25$, the calculated control signal u(t) = 0.25.
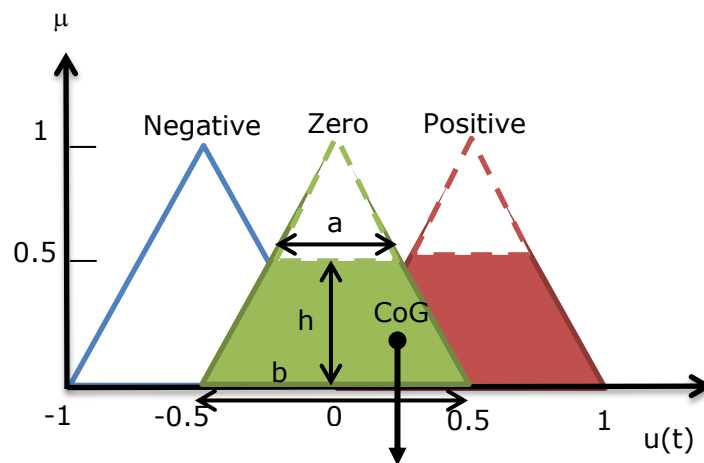
Figure A-6: CoG of the active output membership functions.

The controller applies this new value to the process and consequently new output is produced, thus another control signal is calculated and these cycles are continued until the controller remains in operation.

B. MATLAB script codes and simulation models
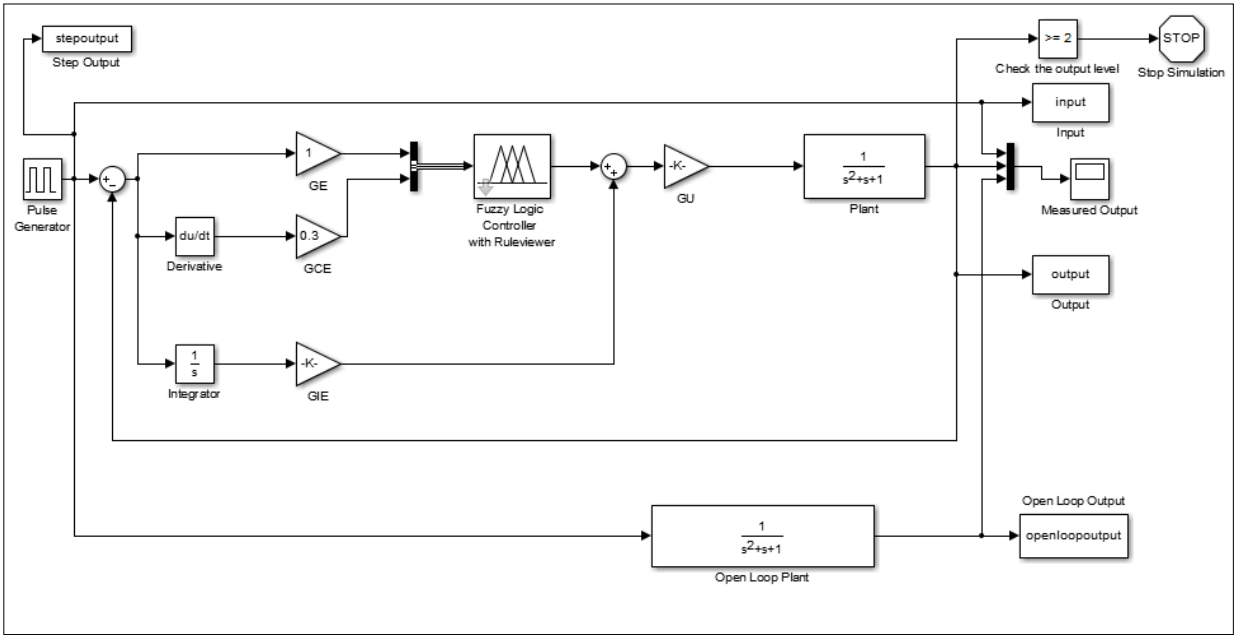
Figure B-1: Simulation model of the closed-loop and open-loop FPD+I controller.

Basic FPD+I Controller Script Code

```matlab
% ---------- BasicFuzzyController.m File -----------
% ------------- Basic Fuzzy Controller -------------
%       Load FPIDSim Model and plot step response
% -------------------------------------------------

% Load simulation block

open_system('FPIDSim');

% Read FLC inference system

flcManual = readfis('flcManual');

% Run simulation

sim('FPIDSim');

% Initializations

format short;

% Calculate open-loop step response characterstics
% rist time, overshoot, settling time and steady state error

open_loop_step_sinfo = stepinfo(openloopoutput, time, 1);
open_loop_rise_time = open_loop_step_sinfo.RiseTime;
open_loop_overshoot = open_loop_step_sinfo.Overshoot;
open_loop_settling_time = open_loop_step_sinfo.SettlingTime;
open_loop_ss_error = 1 - openloopoutput(2001);

if open_loop_ss_error < 0
    open_loop_ss_error = 0;
end

% Convert Numbers to Strings

open_loop_rise_time_str = num2str(open_loop_rise_time,4);
open_loop_overshoot_str = num2str(open_loop_overshoot,4);
open_loop_settling_time_str = num2str(open_loop_settling_time,4);
open_loop_ss_error_str = num2str(open_loop_ss_error,4);

% Calculate closed-loop step response characterstics
% rist time, overshoot, settling time and steady state error

closed_loop_step_input_info = stepinfo(output,time, 1);
closed_loop_rise_time = closed_loop_step_input_info.RiseTime;
closed_loop_overshoot = closed_loop_step_input_info.Overshoot;
closed_loop_settling_time = closed_loop_step_input_info.SettlingTime;
closed_loop_ss_error = 1 - output(2001);

if closed_loop_ss_error < 0
    closed_loop_ss_error = 0;
end

% Convert Numbers to Strings
closed_loop_rise_time_str = num2str(closed_loop_rise_time,4) ;
```

```matlab
closed_loop_overshoot_str = num2str(closed_loop_overshoot,4);
closed_loop_settling_time_str = num2str(closed_loop_settling_time,4);
closed_loop_ss_error_str = num2str(closed_loop_ss_error,4);

% Create figure window

figure;

% Plot Step input

plot(time,stepoutput,'r');

% Plot other responses on the same graph

hold on;

% Plot open-loop step response

plot(time,openloopoutput,'g');

% Plot closed-loop step response

plot(time,output,'b');

% Add graph tags

tagCollect = [];
tag = ['Step Input'] ;
tagCollect = [tagCollect,{tag}] ;

tag = ['Open Loop',' - {\it t}{_r} = ' , open_loop_rise_time_str ,' - {\it M}{_p} = % ',
open_loop_overshoot_str ,' - {\it SS Error} = ', open_loop_ss_error_str  ,' - {\it t}{_s} =
', open_loop_settling_time_str] ;
tagCollect = [tagCollect,{tag}] ;

tag = ['Basic Fuzzy Controller', ' - {\it t}{_r} = ' , closed_loop_rise_time_str, ' - {\it
M}{_p} = % ', closed_loop_overshoot_str, ' - {\it SS Error} = ', closed_loop_ss_error_str, '
- {\it t}{_s} = ', closed_loop_settling_time_str] ;
tagCollect = [tagCollect,{tag}] ;

legend(tagCollect,'Location', 'SouthOutside');

% Put axies labels

xlabel('Time (sec)') ;
ylabel('Measured Output');
axis([0,20,0,1.5]);

set (gca,'XTick', 0:5:20);
set (gca,'YTick', 0:0.25:1.5);

grid on;
hold off;

%--------------- END ----------------------
```

146

Mp and tr 3D Plot Script Code

```matlab
% ------------------- MpandTr3DPlot.m File -------------------
% --- 3D Plot of Maximum percentage overshoot and Rise time ---
% ------------------- against GIE and GU --------------------
% -----------------------------------------------------------

% Load simulation block
open_system('FPIDSim');

% Read FLC inference system
flcManual = readfis('flcManual');

% Initializations

z1=[];            % Overshoot 3D plot array
z2=[];            % Rise time 3D plot array
format long;

% Set minimum, maximum and step size of GIE

gieStepSize = 0.1;
gieMin = 0;
gieMax = 1;

% Set minimum, maximum and step size of GU

guStepSize = 0.1;
guMin = 1;
guMax = 50;

gieLoop = 0;
rtime = 0;
ovshoot = 0;

for gie = gieMin:gieStepSize:gieMax
    gieLoop = gieLoop+1;

    % Clear overshoot and rtime variables

    rtime = 0;
    ovshoot = 0;

    % Set GIE value
    set_param('FPIDSim/GIE','Gain',num2str(gie));

    guLoop = 0;

    for gu = guMin:guStepSize:guMax
        guLoop = guLoop+1;

        % Set GU value
        set_param('FPIDSim/GU','Gain',num2str(gu));

        % Run simulation for the new values of GIE and GU
        sim('FPIDSim');
        % Calculate closed-loop overshhot and ris-time
        stpinfo = stepinfo(output, time, 1);
```

```matlab
        rtime = stpinfo.RiseTime;
        ovshoot = stpinfo.Overshoot;

        % if overshoot and rise time are not calculated then set them to Not a Number

        if (length(output) < 2001)
        ovshoot = NaN;
        rtime = NaN;
        end


        z1(guLoop, gieLoop) = ovshoot;
        z2(guLoop, gieLoop) = rtime;

    end

end

% Set back the values of GIE and GU

set_param('FPIDSim/GIE','Gain',num2str(1)) ;
set_param('FPIDSim/GU','Gain',num2str(1)) ;

% Creat a grid for x and y
[x,y] = meshgrid(gieMin:gieStepSize:gieMax,guMin:guStepSize:guMax);

% 3D plot of Overshoot

figure
surf(x,y,z1,'EdgeColor','none');
xlabel('GIE');
ylabel('GU');
zlabel('{\it M}{_p}(%)');
shading interp
alpha(.6);
view(-45,45);
colorbar
axis([0 gieMax 0 guMax]);

% 3D plot of Rise time

figure
surf(x,y,z2, 'EdgeColor','none');
xlabel('GIE');
ylabel('GU');
zlabel('{\it t}{_r}(s)');
shading interp
alpha(.6);
view(45,45);
colorbar
axis([0 gieMax 0 guMax]);

%--------------- END ----------------------
```

Transient Response 3D Plot Script Code

```matlab
% ------------- TransientResponse3DPlot.m File -----------
% --- 3D Plot of Transient Response against GIE and GU ---
% -------------------------------------------------------


% Load simulation block
open_system('FPIDSim');

% Read FLC inference system
flcManual = readfis('flcManual');

% Initializations

z = []; % Transient Response 3D array

% Set initial and step size of GIE

gieStepSize = 0.1;
gie = 1.1;

% Set initial and step size of GU

guStepSize = 3;
gu = -2;

format long;

% Change value of GE

for i = 1:1:11
    gie = gie - gieStepSize;
    gu = gu + guStepSize;

    % Set GIE value
    set_param('FPIDSim/GIE','Gain',num2str(gie));
    % Set GU value
    set_param('FPIDSim/GU','Gain',num2str(gu));

    % Run simulation
    sim('FPIDSim');

    % if transient response is not obtained set it to
    % Not a Number

    if (length(output) < 2001)
        output = NaN;
    end

    z(1:1:2001, i) = output;

end

% Set back the values of GIE and GU

set_param('FPIDSim/GIE','Gain',num2str(1)) ;
set_param('FPIDSim/GU','Gain',num2str(1)) ;
```

```matlab
% Creat a grid for x and y
[x,y] = meshgrid(1:1:11,1:1:2001);

% 3D plot of Transient Response

surf(x,y,z,'FaceColor','red','EdgeColor','none')
shading interp

camlight left; lighting phong

xlabel('Iteration');
ylabel('Time(s)');
zlabel('Output');

axis([1 11 0 2001]);
set (gca,'XTick', 0:1:11);
set (gca,'YTickLabel', {'0','5','10','15','20'});
alpha(.6);
view(45,45);
colorbar;

%--------------- END ----------------------
```

Auto-tuning Algorithm Script Code

```matlab
% -------------- AutoTuningAlgorithm.m File --------------
% --------------- Auto Tuning Algorithm ----------------
% -----------------------------------------------------

global iteration_data; % Closed-loop characterstics for each iteration

open_system('FPIDSim'); % Load simulation block

% Set the gains to 1

set_param('FPIDSim/GE','Gain','1');
set_param('FPIDSim/GCE','Gain','1');
set_param('FPIDSim/GIE','Gain','1');
set_param('FPIDSim/GU','Gain','1');

flcManual = readfis('flcManual'); %  Read FLC inference system

sim('FPIDSim'); % Run simulation

iteration = 1;

% Plot initial step response and calculate the closed-loop characterstics

[ISE, GCE, GIE, ovshoot] = PlotResponse(output, openloopoutput, stepoutput, time, iteration);

measuredovershoot = ovshoot;

if ovshoot < 1
    measuredovershoot = 1;
end

% Define step sizes og GCE and GIE

GCEstep = 0.1;
GIEstep = 2; % times the initial value of GIE

% If there is overshoot, then set the initial values of GU and GIE

if (ovshoot > 0)
    iteration = iteration + 1;
    newGU = measuredovershoot;
    newGIE = 1/(2*newGU);
    lastISE = ISE;
    lastovshoot = ovshoot;

    % Set new values of GU and GIE

    set_param('FPIDSim/GIE','Gain',num2str(newGIE));
    set_param('FPIDSim/GU','Gain',num2str(newGU));

    sim('FPIDSim'); % Run simulation

    % Plot step response and calculate the closed-loop characterstics

    [ISE, GCE, GIE, ovshoot] = PlotResponse(output, openloopoutput, stepoutput, time,
iteration);
```

```matlab
    end

    % Tune GCE gain

    measuredGCE = str2num(GCE);

    GCEtuned = false; % Not tuned

    j = 0;

    while not (GCEtuned)

        j = j+1;
        iteration = iteration + 1;
        lastISE = ISE;
        lastovshoot = ovshoot;
        lastGCE = str2num(GCE);
        measuredGCE = str2num(GCE);

        % Set new value of GCE

        set_param('FPIDSim/GCE','Gain',num2str(measuredGCE-GCEstep));

        sim('FPIDSim');    % Run simulation

        % Plot step response and calculate the closed-loop characterstics

        [ISE, GCE, GIE, ovshoot] = PlotResponse(output, openloopoutput, stepoutput, time,
    iteration);

        if (ISE > lastISE || ovshoot >lastovshoot)

            GCEtuned = true;
            set_param('FPIDSim/GCE','Gain',num2str(lastGCE));
            GCE = get_param('FPIDSim/GCE','Gain');

        end
    end

    iteration = iteration + 1;

    sim('FPIDSim'); % Run simulation

    % Plot step response and calculate the closed-loop characterstics

    [ISE, GCE, GIE, ovshoot] = PlotResponse(output, openloopoutput, stepoutput, time, iteration);

    % Tune GIE gain

    measuredGIE = str2num(GIE);
    newGIE = measuredGIE;
    GIEtuned = false; % Not tuned

    k=0;

    while not (GIEtuned)
        k = k+1;
```

```matlab
    iteration = iteration + 1;
    lastISE = ISE;
    lastovshoot = ovshoot;
    lastGIE = str2num(GIE);
    measuredGIE = str2num(GIE);
    newGIE = GIEstep * measuredGIE;

    % Set new value of GIE

    set_param('FPIDSim/GIE','Gain',num2str(newGIE));

    sim('FPIDSim'); % Run simulation

    % Plot step response and calculate the closed-loop characterstics

    [ISE, GCE, GIE, ovshoot] = PlotResponse(output, openloopoutput, stepoutput, time,
iteration);

    if (ISE > lastISE || ovshoot >lastovshoot)
        GIEtuned = true;
        set_param('FPIDSim/GIE','Gain',num2str(lastGIE));
        GIE = get_param('FPIDSim/GIE','Gain');
    end
end

iteration = iteration + 1;

% Run simulation

sim('FPIDSim');

% Plot step response and calculate the closed-loop characterstics

PlotResponse(output, openloopoutput, stepoutput, time, iteration);

iteration_data = iteration_data (1:iteration, 1:10);

%--------------- END -----------------------
```

Disturbance Test Script Code

```matlab
% ---------- DisturbanceTest.m File ------------
% ---------- Step Disturbance Test ------------
% --------------------------------------------

% Load simulation block

open_system('FPIDSimDisturbanceTest');

flcManual=readfis('flcManual'); % Read FLC inference system

sim('FPIDSimDisturbanceTest'); % Run simulation

% Plot Step input

subplot(3,1,1);
plot(time,stepinput,'r','LineWidth',1.5);
xlabel('Time') ;
ylabel('Amplitute');
axis([0,100, 0,1.5]);
set (gca,'XTick', 0:5:100);
set (gca,'YTick', 0:0.5:1.5);

tag = 'Step Input';
legend (tag);
grid on;

% Plot open-loop step response

subplot(3,1,2);
plot(time,disturbance,'m', 'LineWidth',1.5);
xlabel('Time') ;
ylabel('Amplitute');
axis([0,100,-1.5,1.5]);
set (gca,'XTick', 0:5:100);
set (gca,'YTick', -1.5:0.5:1.5);

tag = 'Step Disturbance';
legend (tag);
grid on;

% Plot FLC step response

subplot(3,1,3);
plot(time,output,'b', 'LineWidth',1.5);
xlabel('Time') ;
ylabel('Amplitute');
axis([0,100,0,1.5]);
set (gca,'XTick', 0:5:100);
set (gca,'YTick', 0:0.5:1.5);

tag = 'Closed-loop';
legend (tag);
grid on;
%--------------- END ----------------------
```
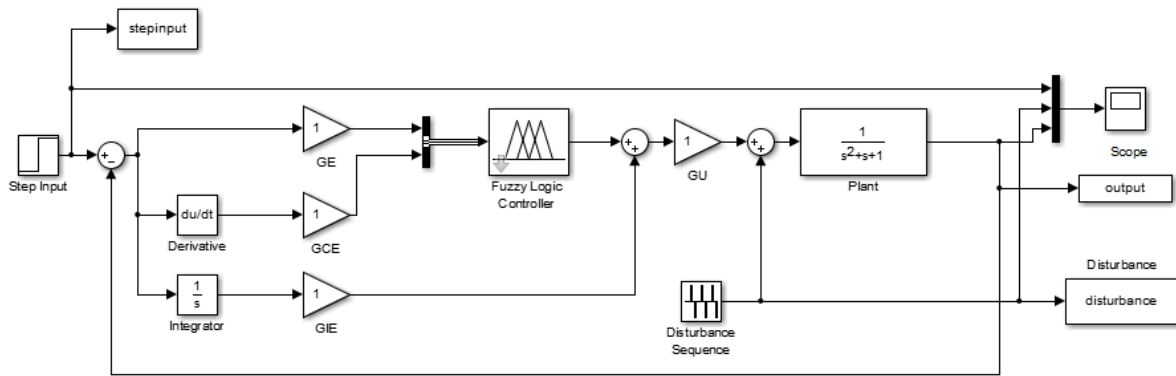
Figure B-2: Simulation model of disturbance test.

C. MATLAB Results
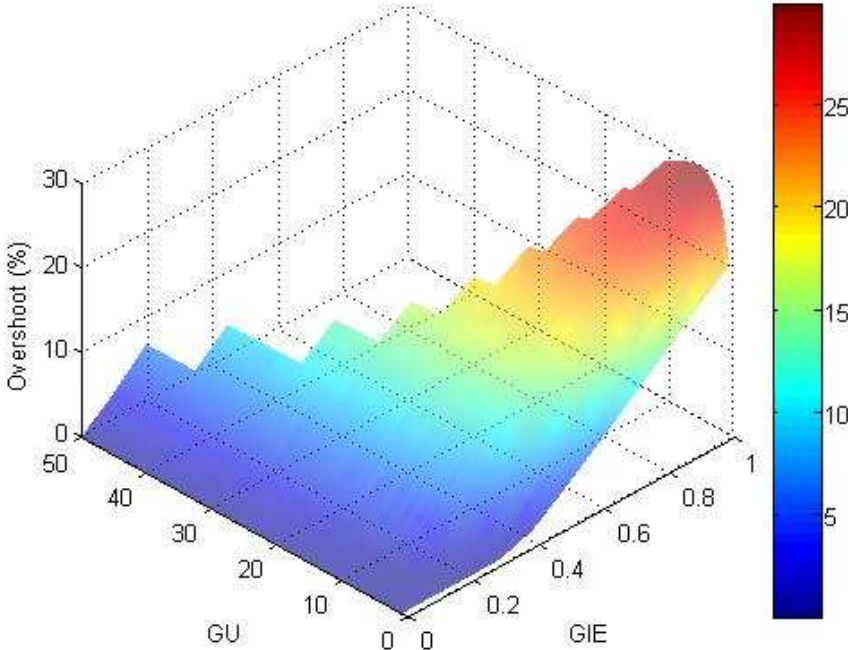
Figure C-10: The 3D plot between M_p and the basic FPD+I gains (G_{IE}, G_U) for Case 2.
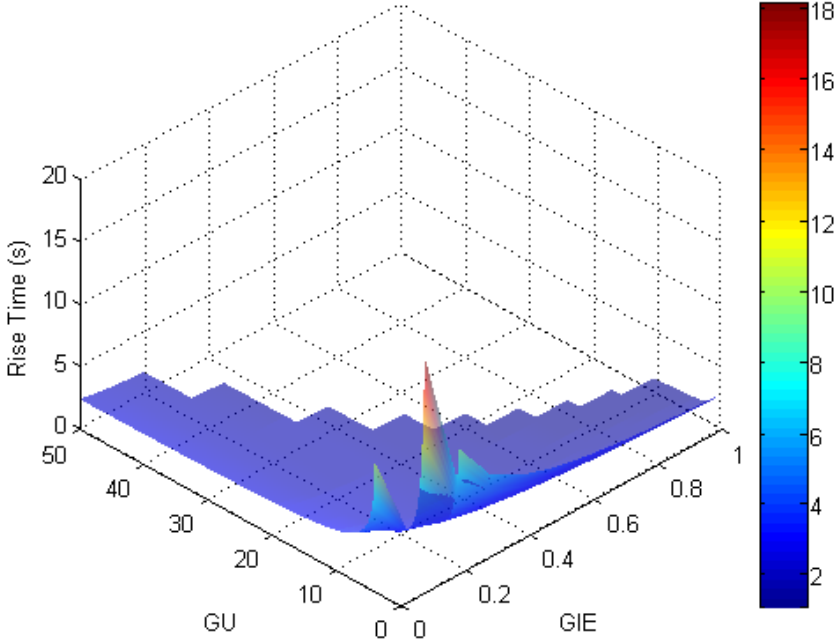
Figure C-2: The 3D plot between t_r and the basic FPD+I gains (G_{IE}, G_U) for Case 2.
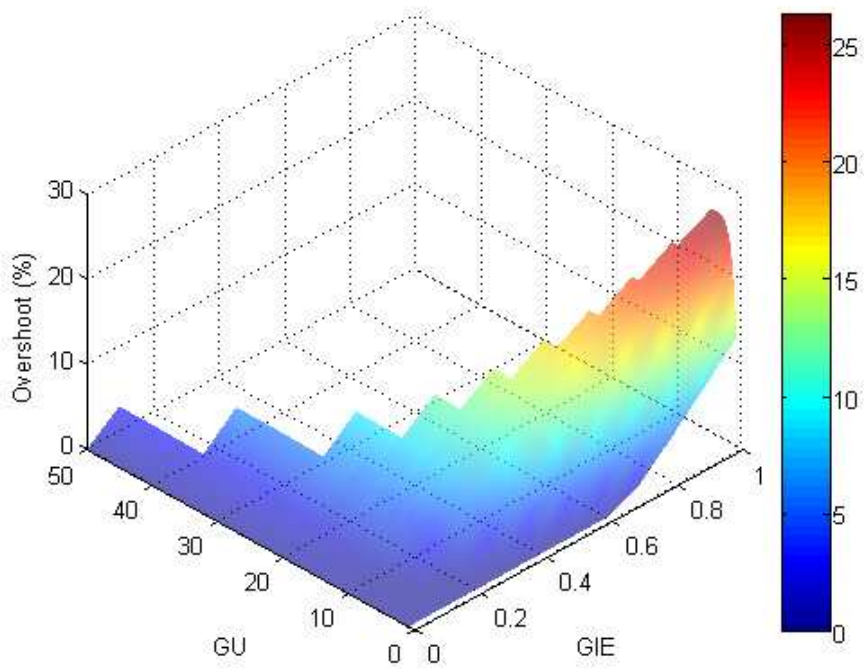
Figure C-3: The 3D plot between M_p and the basic FPD+I gains (G_{IE}, G_U) for Case 3.
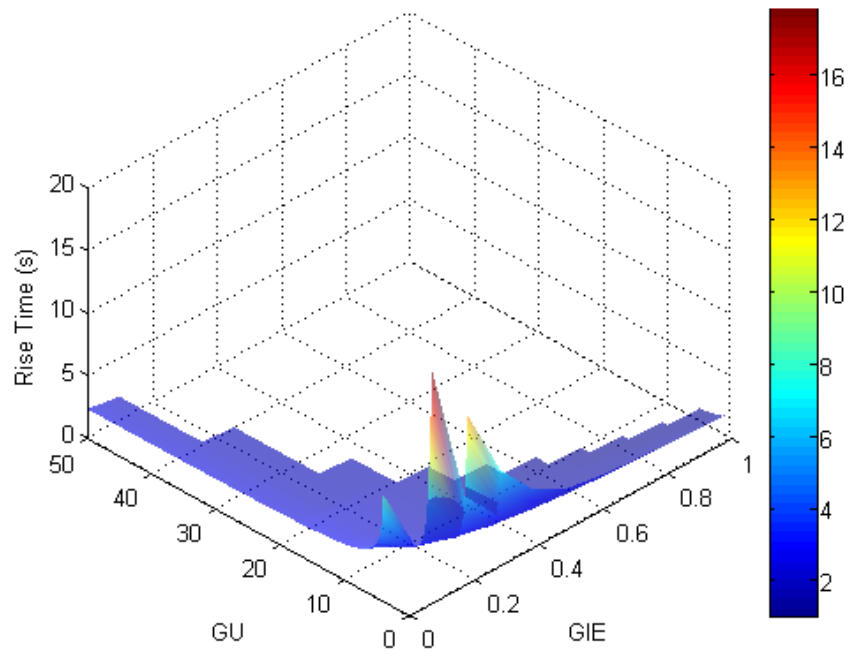
Figure C-4: The 3D plot between t_r and the basic FPD+I gains (G_{IE}, G_U) for Case 3.
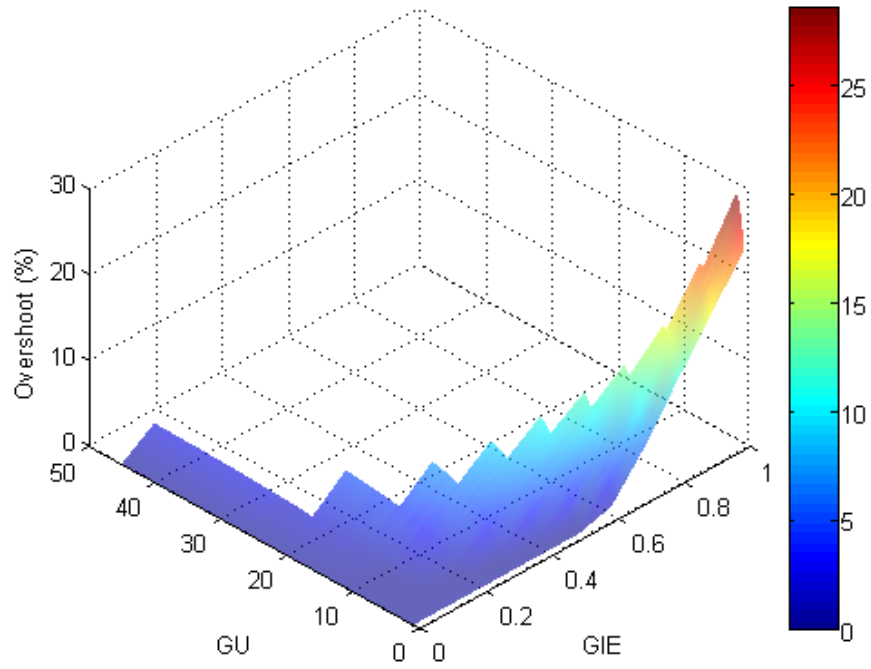
Figure C-5: The 3D plot between M_p and the basic FPD+I gains (G_{IE}, G_U) for Case 4.
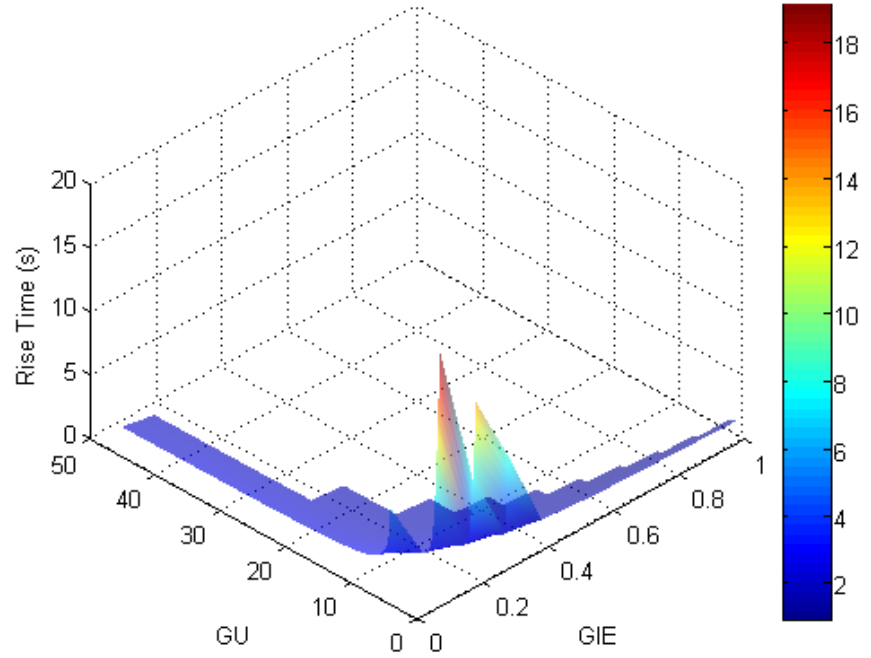
Figure C-6: The 3D plot between t_r and the basic FPD+I gains (G_{IE}, G_U) for Case 4.
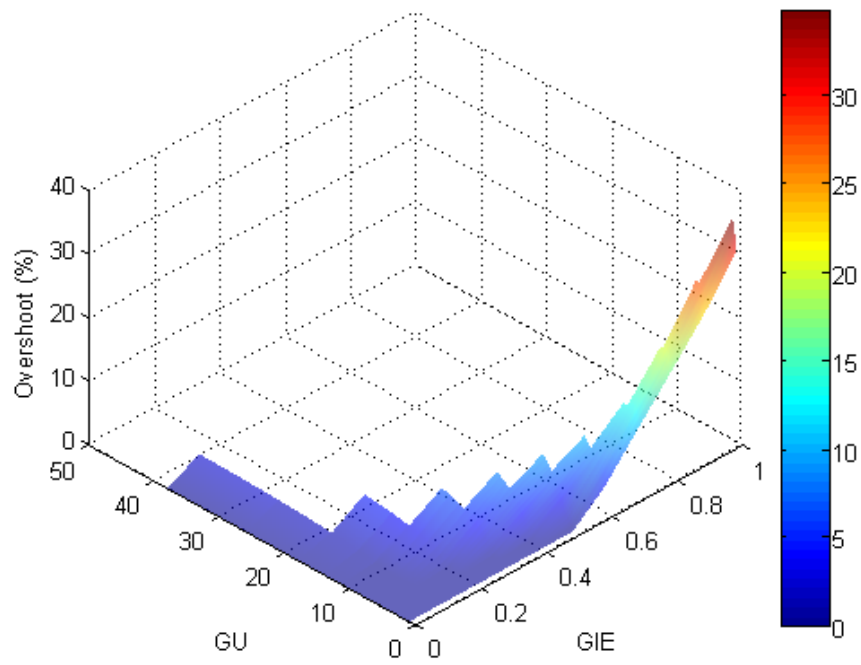
Figure C-7 The 3D plot between M_p and the basic FPD+I gains (G_{IE}, G_U) for Case 5.
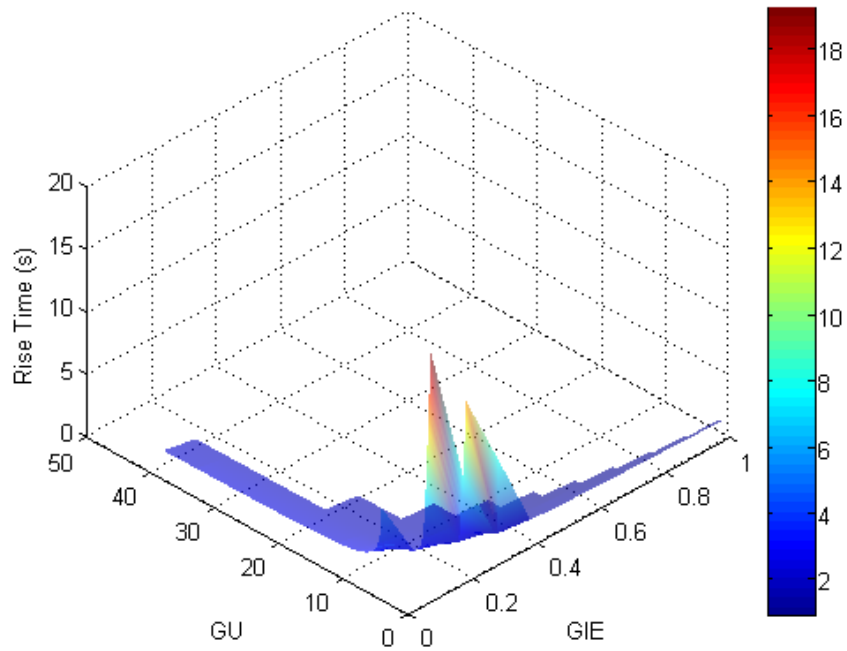
Figure C-8: The 3D plot between t_r and the basic FPD+I gains (G_{IE}, G_U) for Case 5.

159
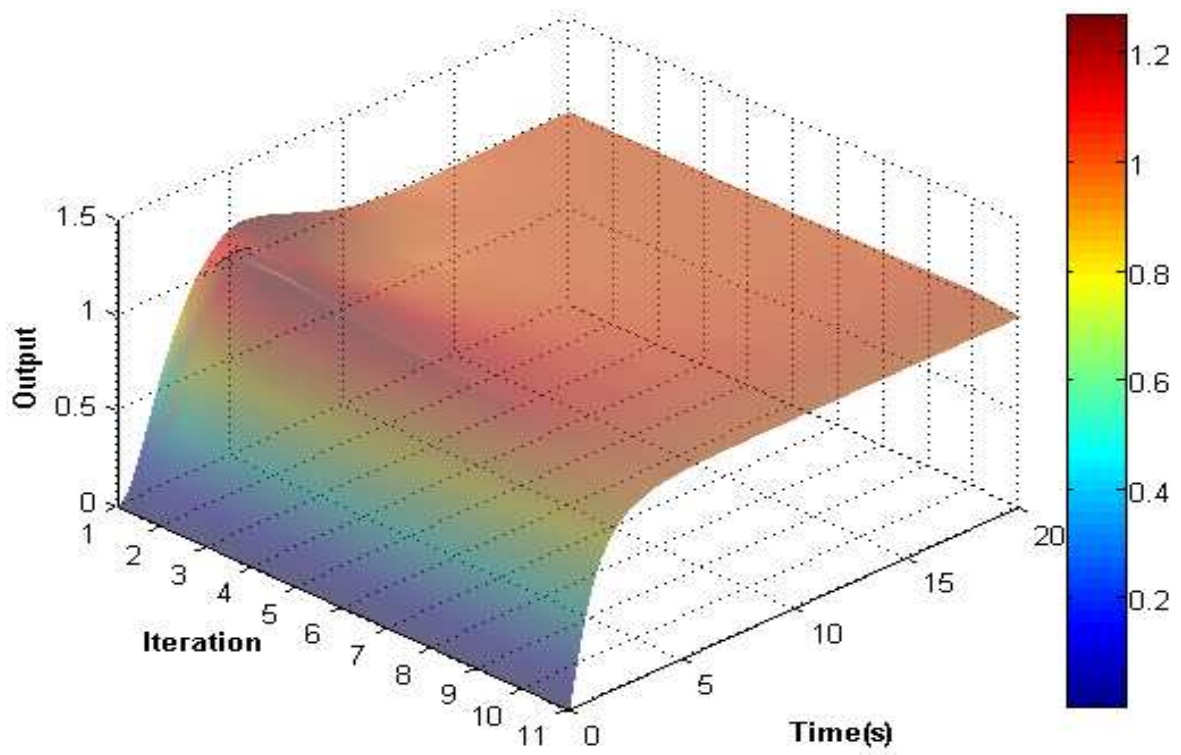
Figure C-9: The transient response of Case 2 for different values of G_{IE} and G_U.
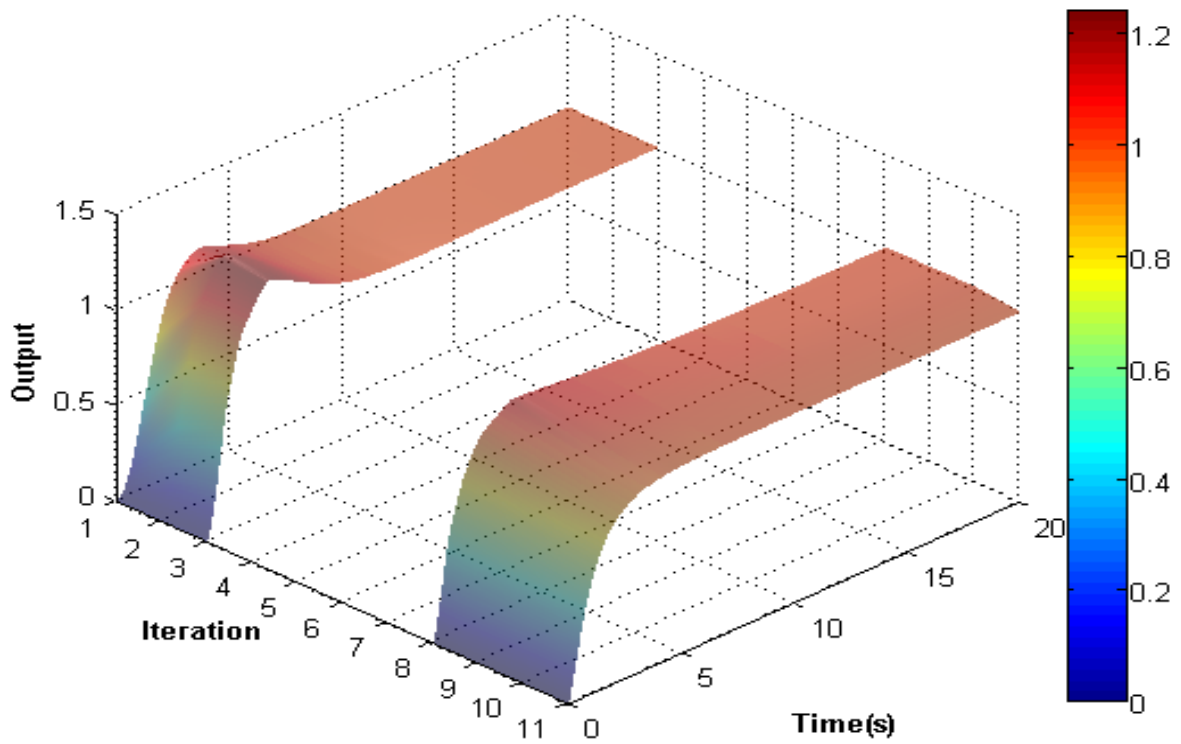
Figure C-10: The transient response of Case 3 for different values of G_{IE} and G_U.
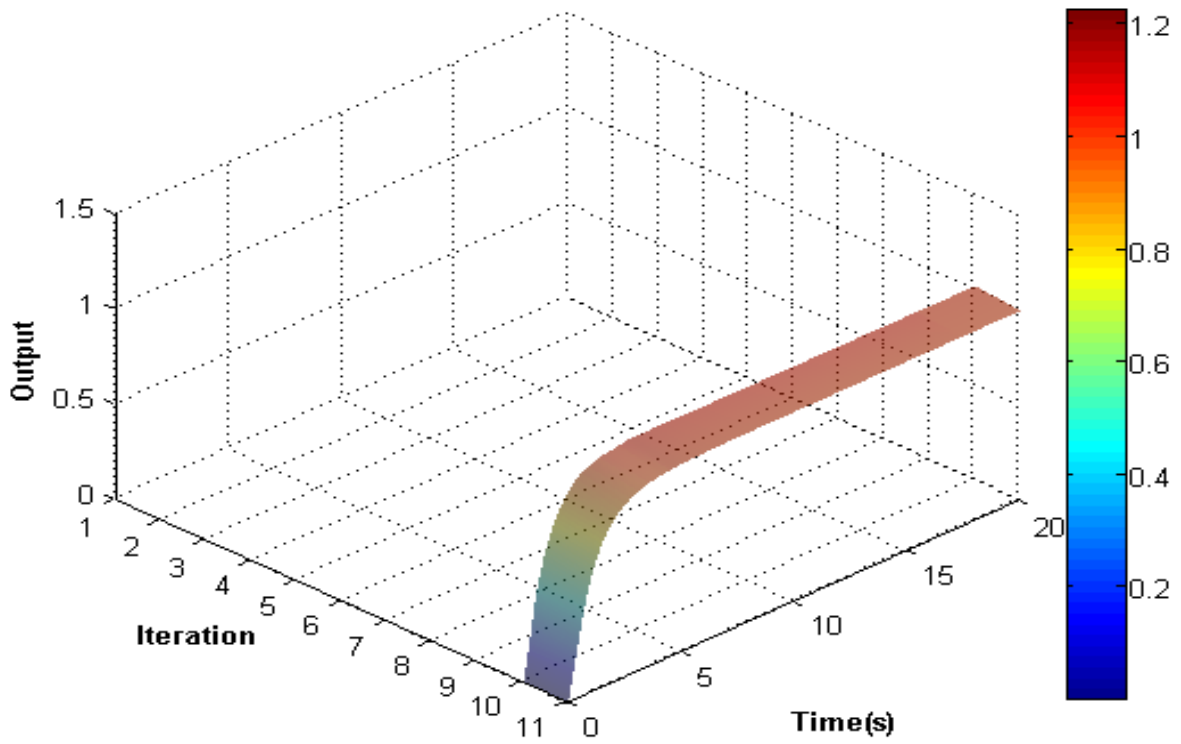
Figure C-11: The transient response of Case 4 for different values of G_{IE} and G_U.
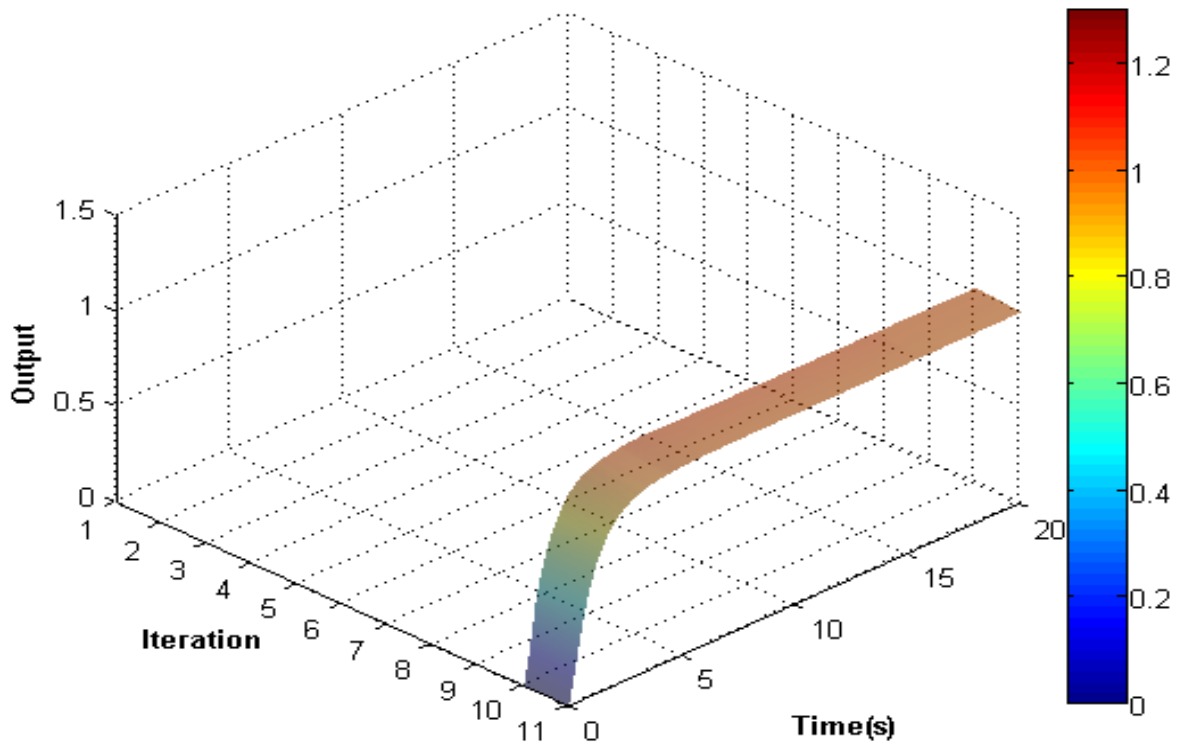
Figure C-12: The transient response of Case 5 for different values of G_{IE} and G_U.
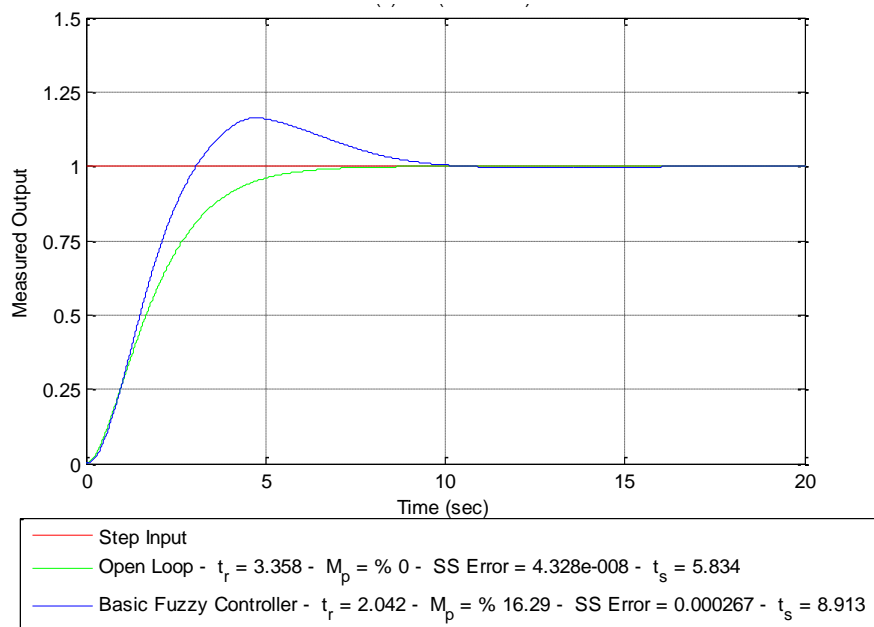
Figure C-13: Closed-loop step response of Case 1 using the basic FPD+I controller.
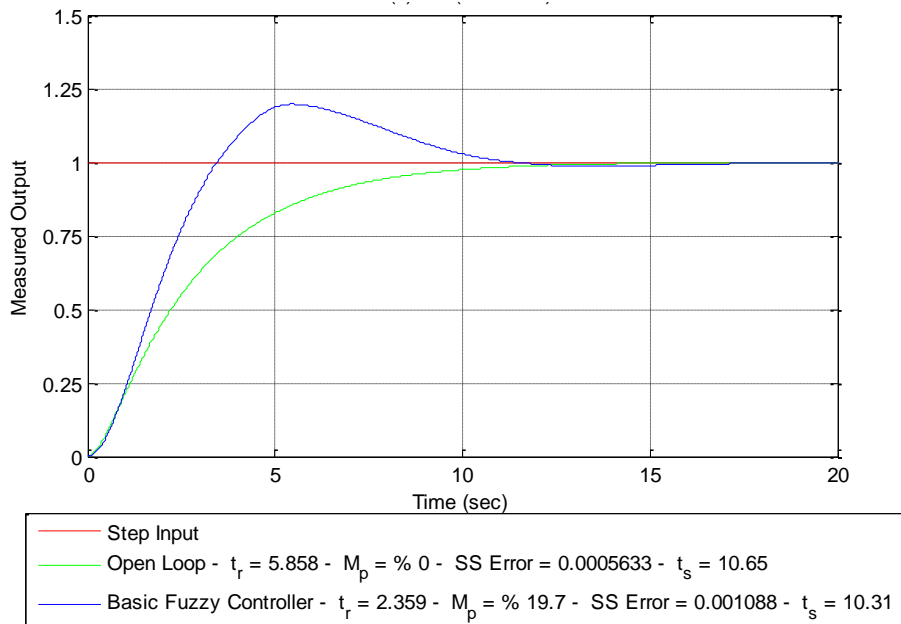
Figure C-14: Closed-loop step response of Case 2 using the basic FPD+I controller.
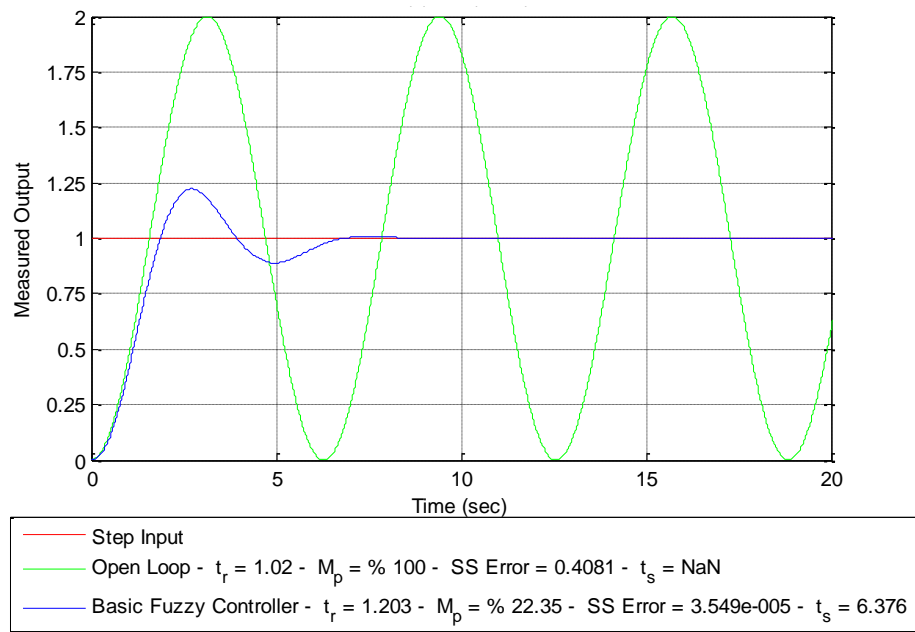
Figure C-15: Closed-loop step response of Case 4 using the basic FPD+I controller.
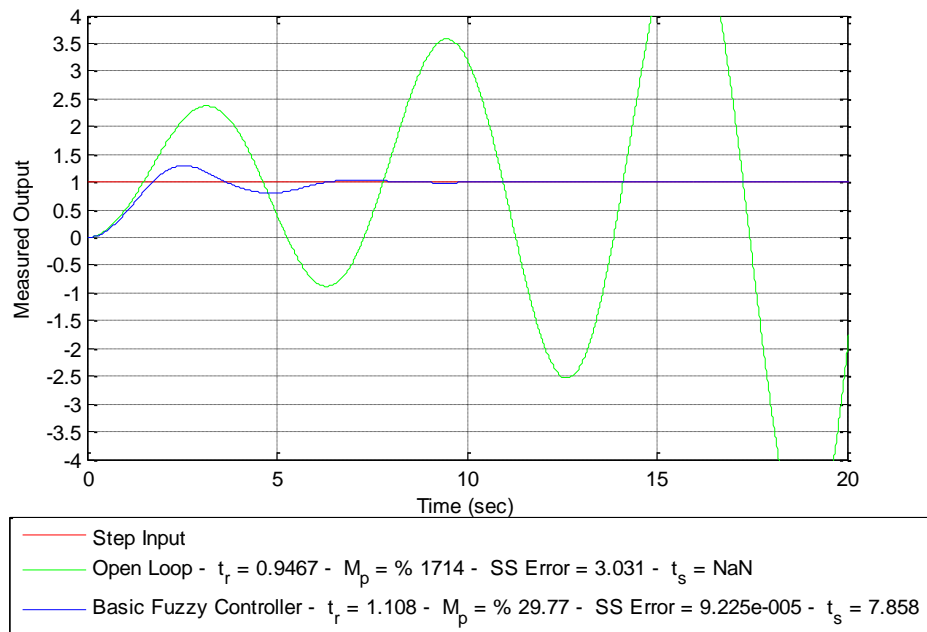
Figure C-16: Closed-loop step response of Case 5 using the basic FPD+I controller.
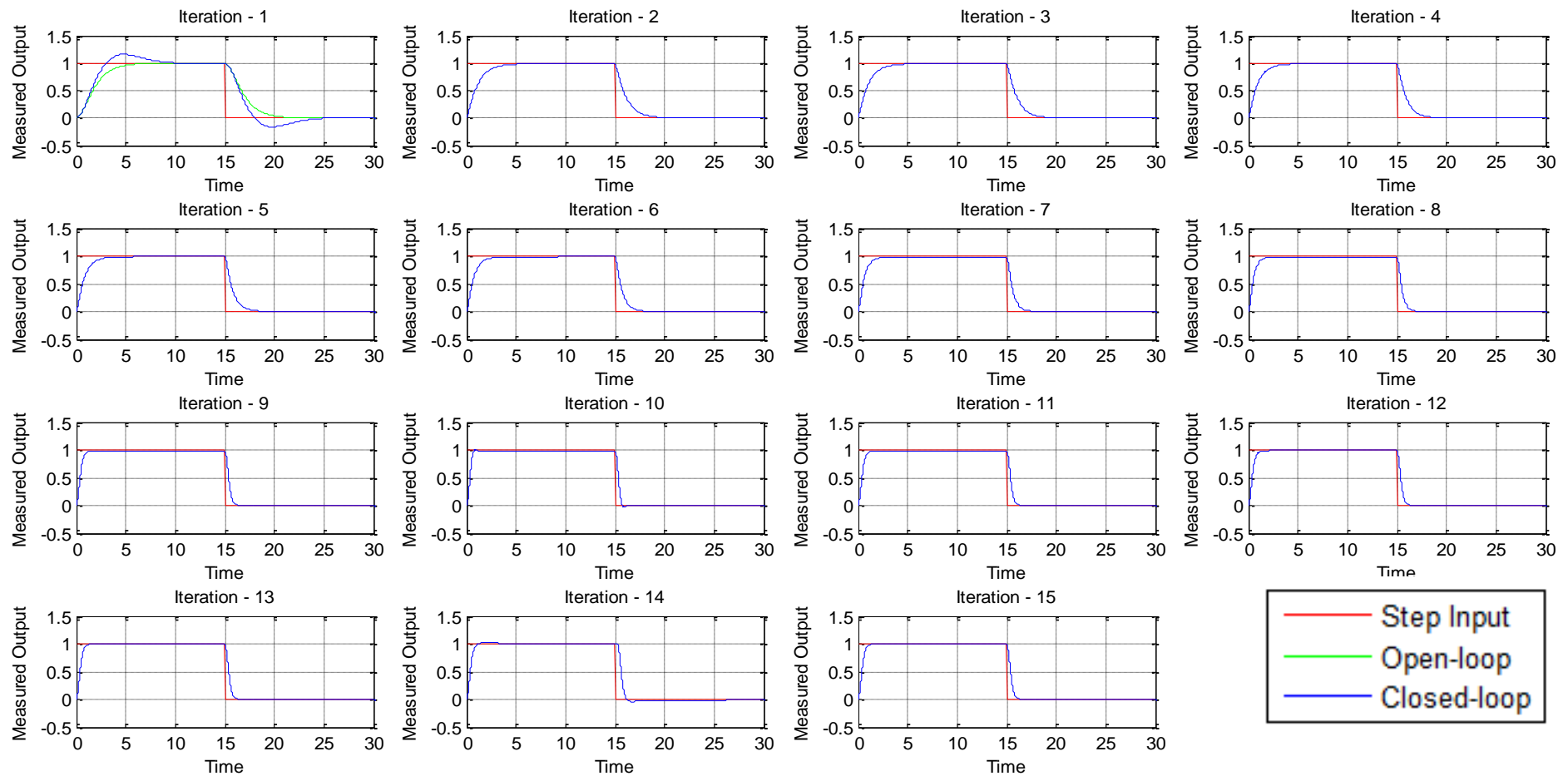
Figure C-17: Closed-loop step responses for Case 1 using the auto-tuning algorithm for step sizes of G_{CE} = 0.1 and G_{IE} = twice of initial value of G_{IE}.
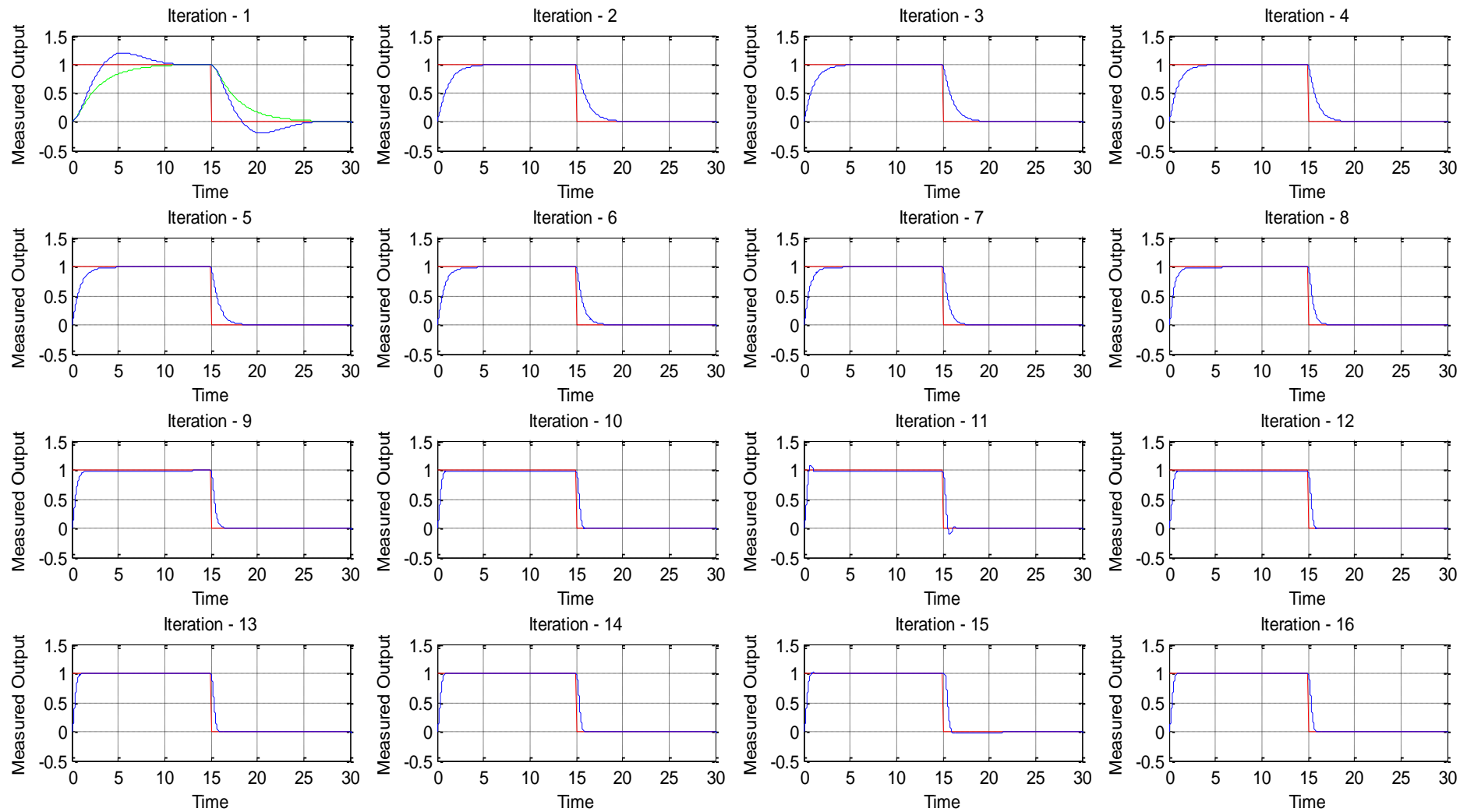
Figure C-18: Closed-loop step responses for Case 2 using the auto-tuning algorithm

for step sizes of $G_{CE} = 0.1$ and G_{IE} = twice of initial value of G_{IE}.

165
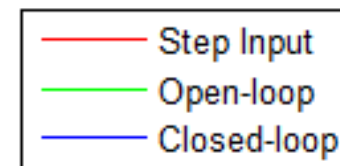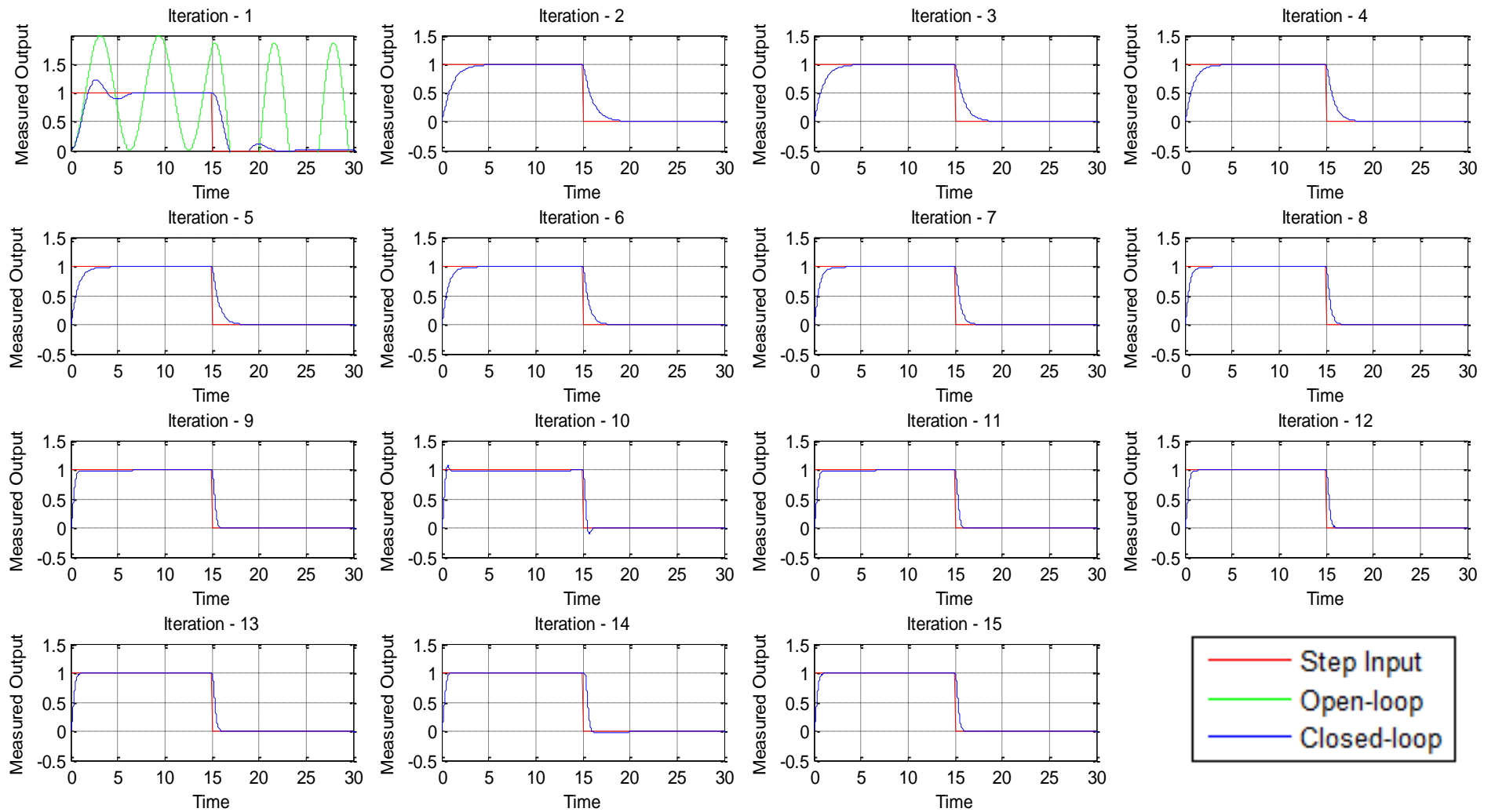
Figure C-19: Closed-loop step responses for Case 4 using the auto-tuning algorithm for step sizes of $G_{CE} = 0.1$ and G_{IE} = twice of initial value of G_{IE}.
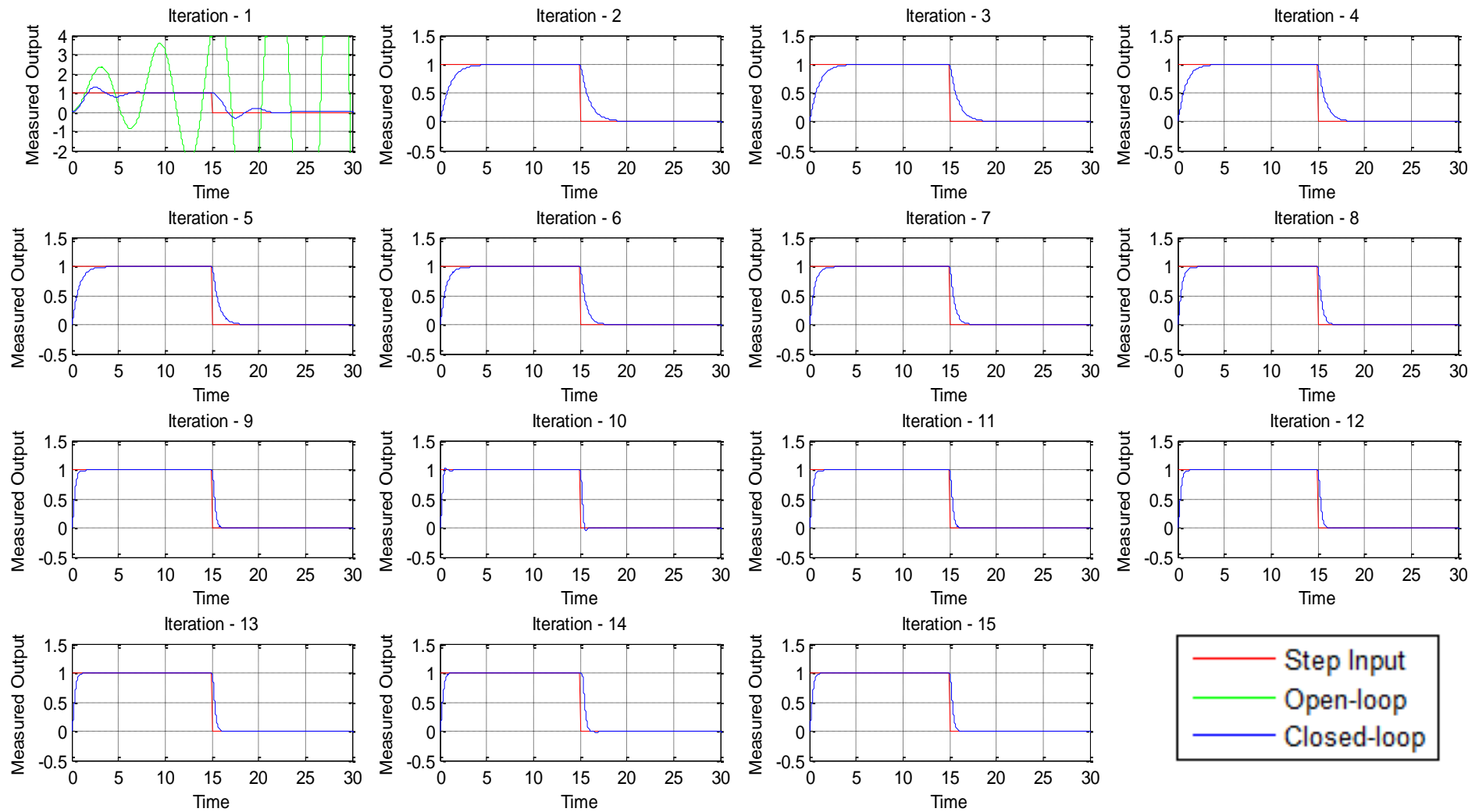
Figure C-20: Closed-loop step responses for Case 5 using the auto-tuning algorithm for step sizes of G_{CE} = 0.1 and G_{IE} = twice of initial value of G_{IE}.

Table C-1: Controller gains' values and closed-loop characteristics of Case 1 using the auto-tuning algorithm for step sizes of G_{CE} = 0.1 and G_{IE} = twice of the initial value of G_{IE}.

Iteration	Controller Gain				ISE	Closed-loop Characteristics			
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p	t_s	SSE
1	1	1	1	1	1.193	2.042	16.291	8.913	0.003
2	1	1	0.031	16.291	0.646	2.466	0	4.901	0.007
3	1	0.9	0.031	16.291	0.593	2.238	0	4.541	0.008
4	1	0.8	0.031	16.291	0.540	2.004	0	4.165	0.009
5	1	0.7	0.031	16.291	0.489	1.764	0	3.770	0.010
6	1	0.6	0.031	16.291	0.437	1.517	0	3.351	0.011
7	1	0.5	0.031	16.291	0.389	1.260	0	2.899	0.012
8	1	0.4	0.031	16.291	0.344	0.989	0	2.394	0.013
9	1	0.3	0.031	16.291	0.304	0.705	0	1.758	0.014
10	1	0.2	0.031	16.291	0.275	0.475	0.329	1.828	0.015
11	1	0.3	0.031	16.291	0.304	0.705	0	1.758	0.014
12	1	0.3	0.061	16.291	0.301	0.689	0	1.451	0.009
13	1	0.3	0.123	16.291	0.296	0.661	0	1.193	0.001
14	1	0.3	0.246	16.291	0.293	0.617	1.894	0.985	-0.004
15	1	0.3	0.123	16.291	0.296	0.661	0	1.193	0.001

Table C-2: Controller gains' values and closed-loop characteristics of Case 2 using the auto-tuning algorithm for step sizes of $G_{CE} = 0.1$ and G_{IE} = twice of the initial value of G_{IE}.

Iteration	Controller Gain				ISE	Closed-loop Characteristics			
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p	t_s	SSE
1	1	1	1	1	1.382	2.359	19.696	10.309	0.011
2	1	1	0.025	19.696	0.649	2.498	0	4.829	0.006
3	1	0.9	0.025	19.696	0.594	2.271	0	4.455	0.007
4	1	0.8	0.025	19.696	0.541	2.040	0	4.068	0.008
5	1	0.7	0.025	19.696	0.488	1.804	0	3.664	0.009
6	1	0.6	0.025	19.696	0.437	1.562	0	3.240	0.010
7	1	0.5	0.025	19.696	0.387	1.313	0	2.791	0.010
8	1	0.4	0.025	19.696	0.340	1.054	0	2.307	0.011
9	1	0.3	0.025	19.696	0.297	0.779	0	1.758	0.012
10	1	0.2	0.025	19.696	0.263	0.512	0	0.927	0.013
11	1	0.1	0.025	19.696	0.252	0.382	7.567	1.588	0.013
12	1	0.2	0.025	19.696	0.263	0.512	0	0.927	0.013
13	1	0.2	0.051	19.696	0.261	0.506	0	0.877	0.009
14	1	0.2	0.102	19.696	0.259	0.497	0	0.817	0.003
15	1	0.2	0.203	19.696	0.256	0.479	1.354	0.751	-0.003
16	1	0.2	0.102	19.696	0.259	0.497	0	0.817	0.003

Table C-3: Controller gains' values and closed-loop characteristics of Case 4 using the auto-tuning algorithm for step sizes of G_{CE} = 0.1 and G_{IE} = twice of the initial value of G_{IE}.

Iteration	Controller Gain				ISE	Closed-loop Characteristics			
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p	t_s	SSE
1	1	1	1	1	0.929	1.203	22.351	6.376	0
2	1	1	0.022	22.351	0.603	2.288	0	4.463	0.006
3	1	0.9	0.022	22.351	0.549	2.058	0	4.075	0.007
4	1	0.8	0.022	22.351	0.496	1.824	0	3.673	0.008
5	1	0.7	0.022	22.351	0.441	1.585	0	3.254	0.008
6	1	0.6	0.022	22.351	0.390	1.340	0	2.815	0.009
7	1	0.5	0.022	22.351	0.338	1.085	0	2.346	0.010
8	1	0.4	0.022	22.351	0.289	0.817	0	1.830	0.011
9	1	0.3	0.022	22.351	0.247	0.539	0	1.147	0.011
10	1	0.2	0.022	22.351	0.219	0.339	6.165	1.487	0.012
11	1	0.3	0.022	22.351	0.247	0.539	0	1.147	0.011
12	1	0.3	0.045	22.351	0.246	0.532	0	1.045	0.008
13	1	0.3	0.089	22.351	0.244	0.520	0	0.921	0.003
14	1	0.3	0.179	22.351	0.241	0.499	0.772	0.806	-0.002
15	1	0.3	0.089	22.351	0.244	0.520	0	0.921	0.003

Table C-4: Controller gains' values and closed-loop characteristics of Case 5 using the auto-tuning algorithm for step size of G_{CE} = 0.1 and G_{IE} = twice of the initial value of G_{IE}.

Iteration	Controller Gain				ISE	Closed-loop Characteristics			
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p	t_s	SSE
1	1	1	1	1	0.975	1.108	29.767	7.858	0
2	1	1	0.017	29.767	0.591	2.256	0	4.317	0.005
3	1	0.9	0.017	29.767	0.536	2.032	0	3.922	0.005
4	1	0.8	0.017	29.767	0.482	1.803	0	3.517	0.006
5	1	0.7	0.017	29.767	0.427	1.571	0	3.100	0.006
6	1	0.6	0.017	29.767	0.377	1.333	0	2.670	0.007
7	1	0.5	0.017	29.767	0.324	1.087	0	2.219	0.008
8	1	0.4	0.017	29.767	0.274	0.832	0	1.740	0.008
9	1	0.3	0.017	29.767	0.228	0.562	0	1.188	0.009
10	1	0.2	0.017	29.767	0.193	0.324	2.495	0.714	0.009
11	1	0.3	0.017	29.767	0.228	0.562	0	1.188	0.009
12	1	0.3	0.034	29.767	0.227	0.556	0	1.128	0.006
13	1	0.3	0.067	29.767	0.226	0.545	0	1.036	0.002
14	1	0.3	0.134	29.767	0.224	0.527	0.504	0.914	-0.002
15	1	0.3	0.067	29.767	0.226	0.545	0	1.036	0.002

D. LabVIEW Results
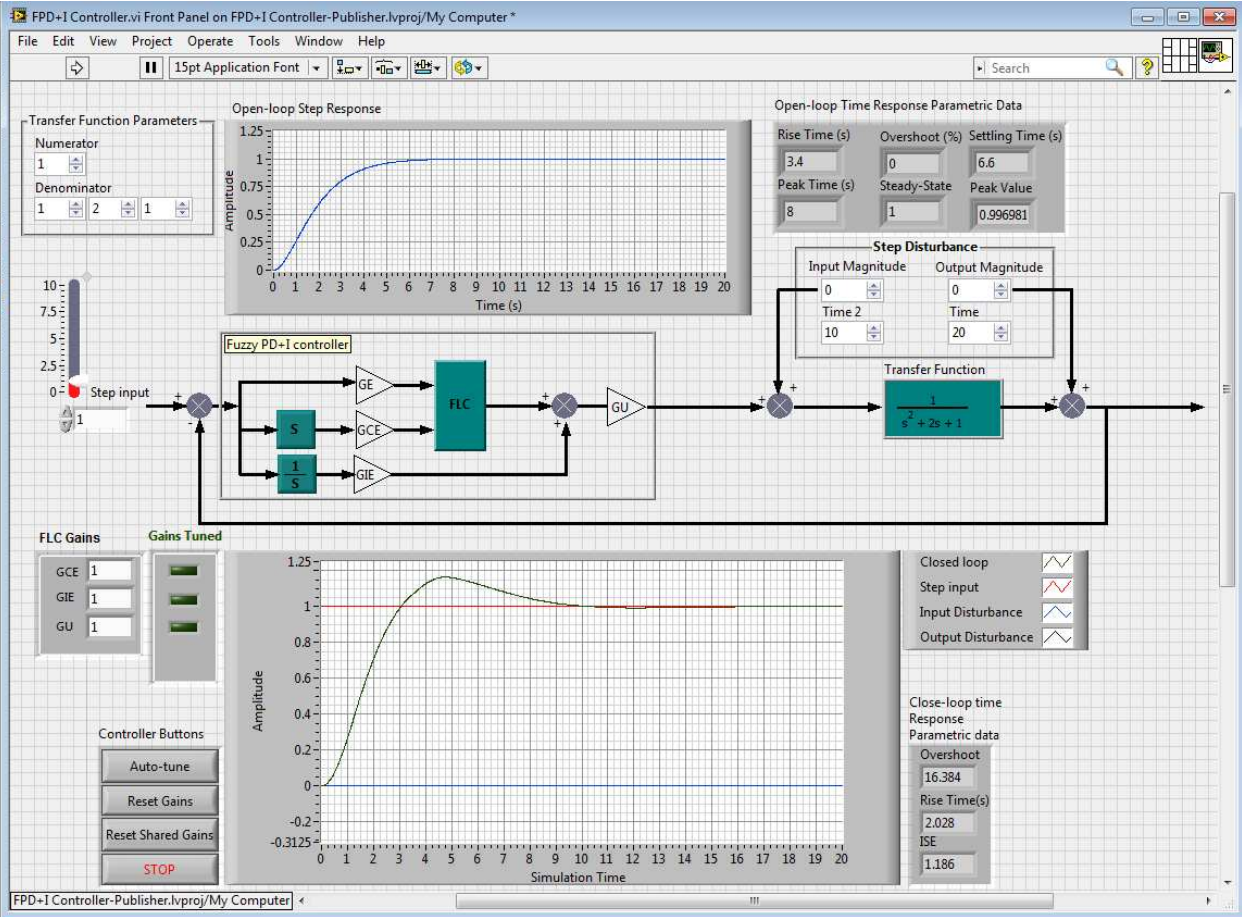
Figure D-1: The basic FPD+I controller settings and step response of Case 1.
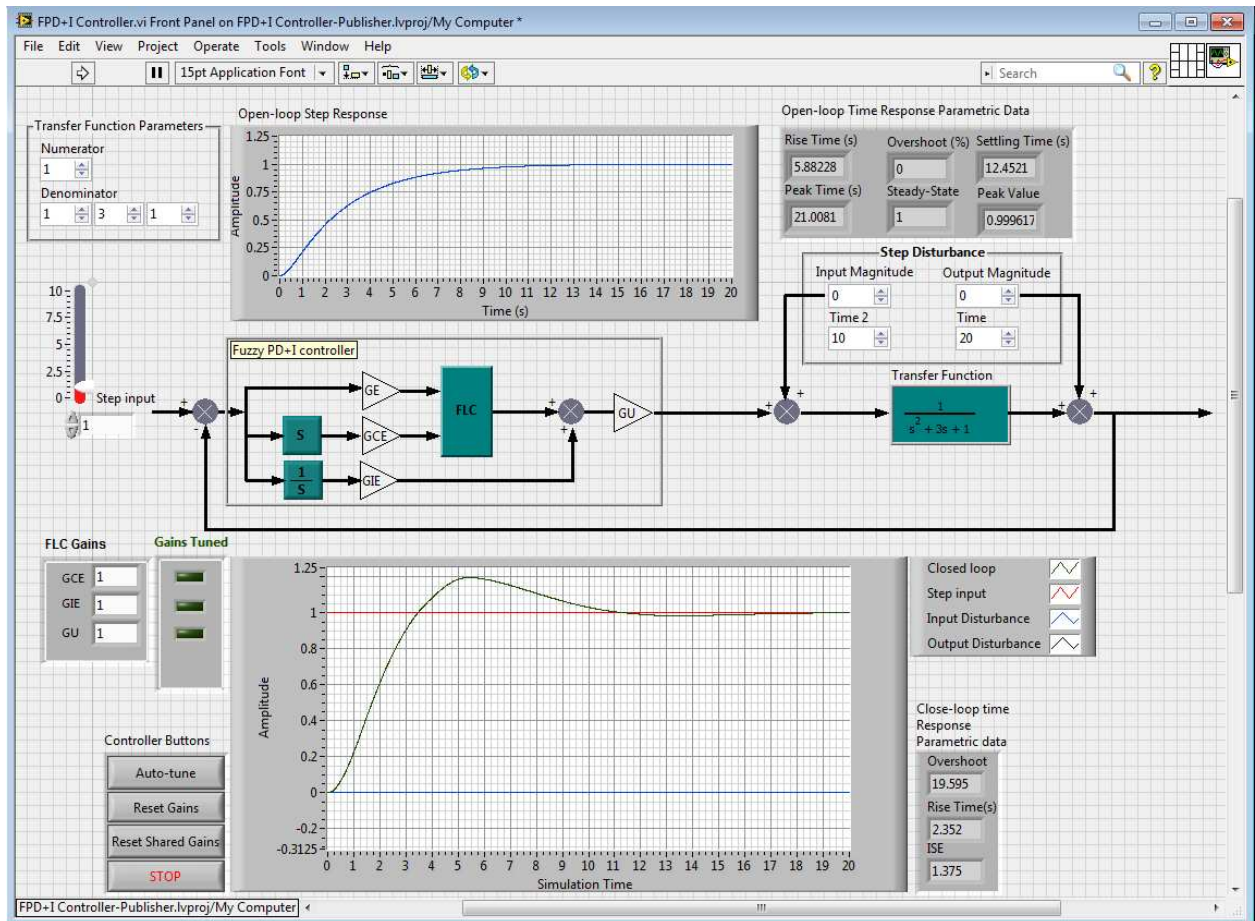
Figure D-2: The basic FPD+I controller settings and step response of Case 2.
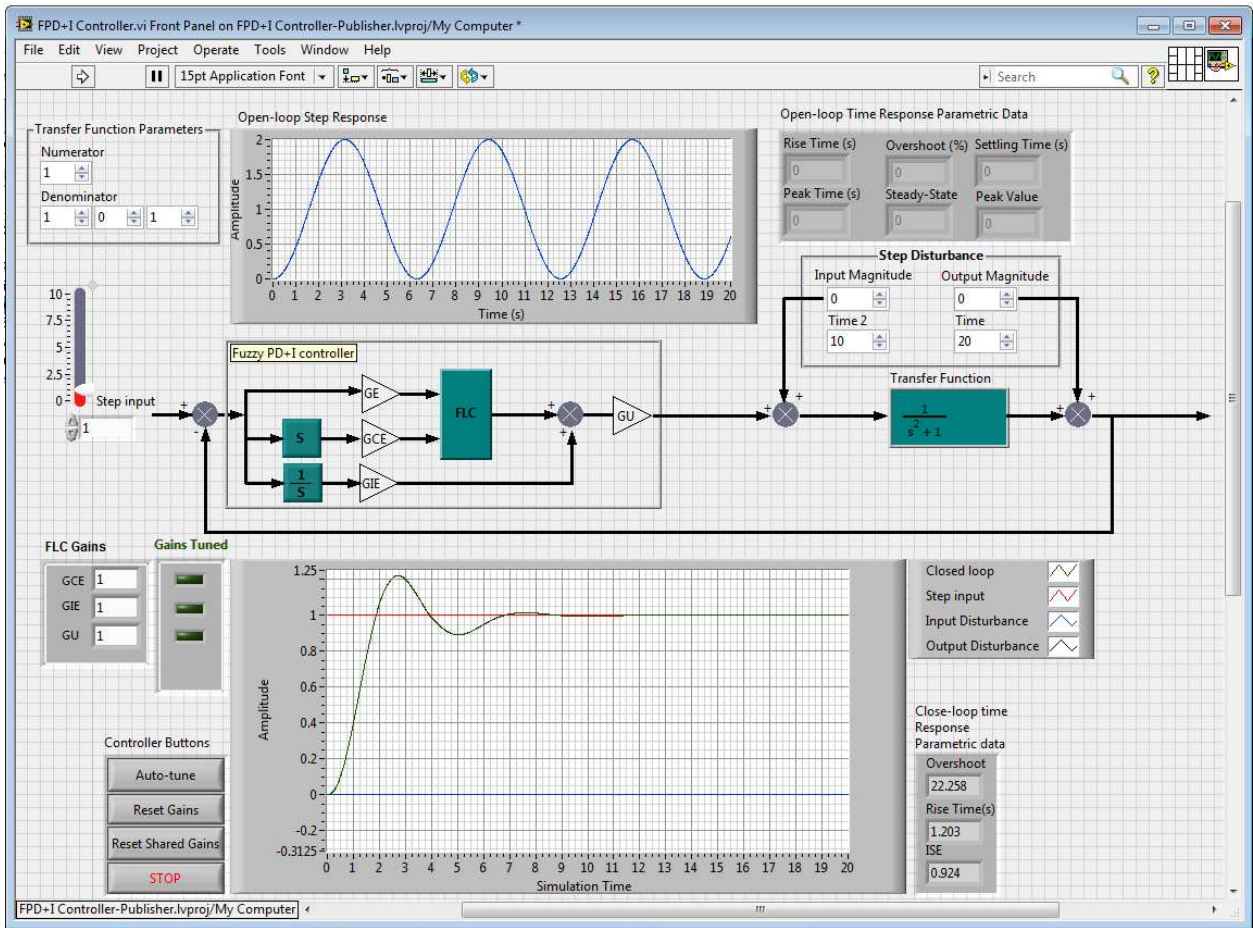
Figure D-3: The basic FPD+I controller settings and step response of Case 4.
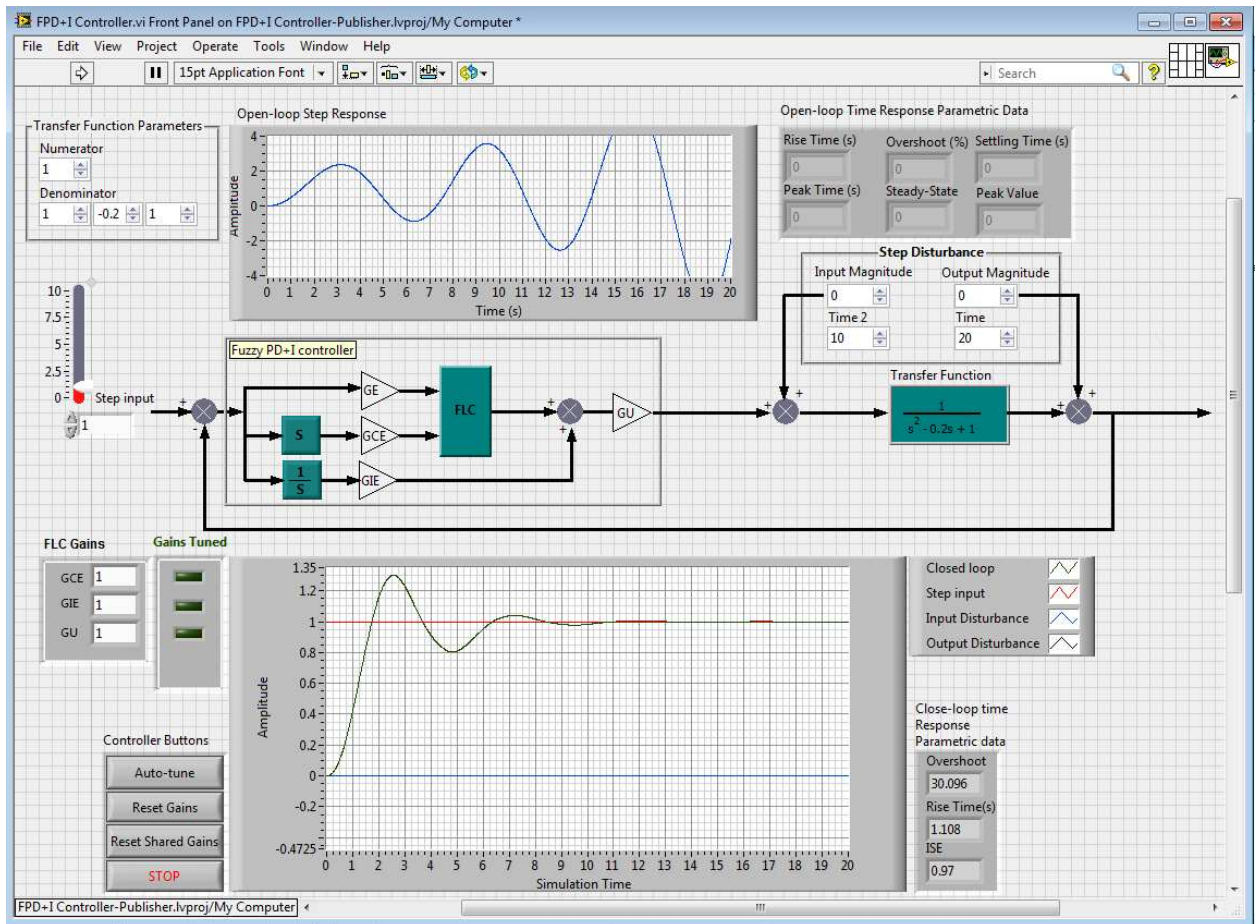
Figure D-4: The basic FPD+I controller settings and step response of Case 5.
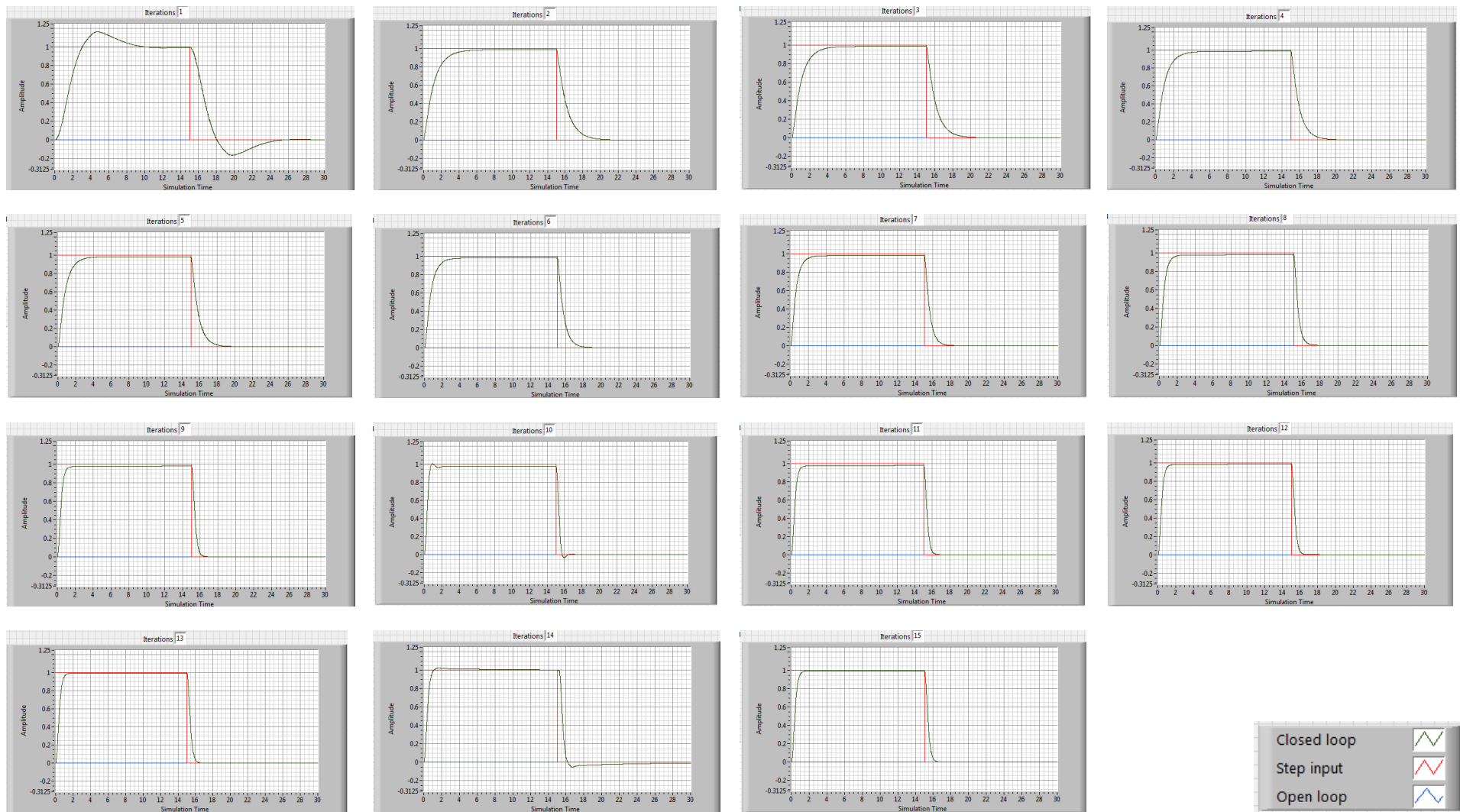
Figure D-5: Closed-loop step response for Case 1 using the auto-tune algorithm.
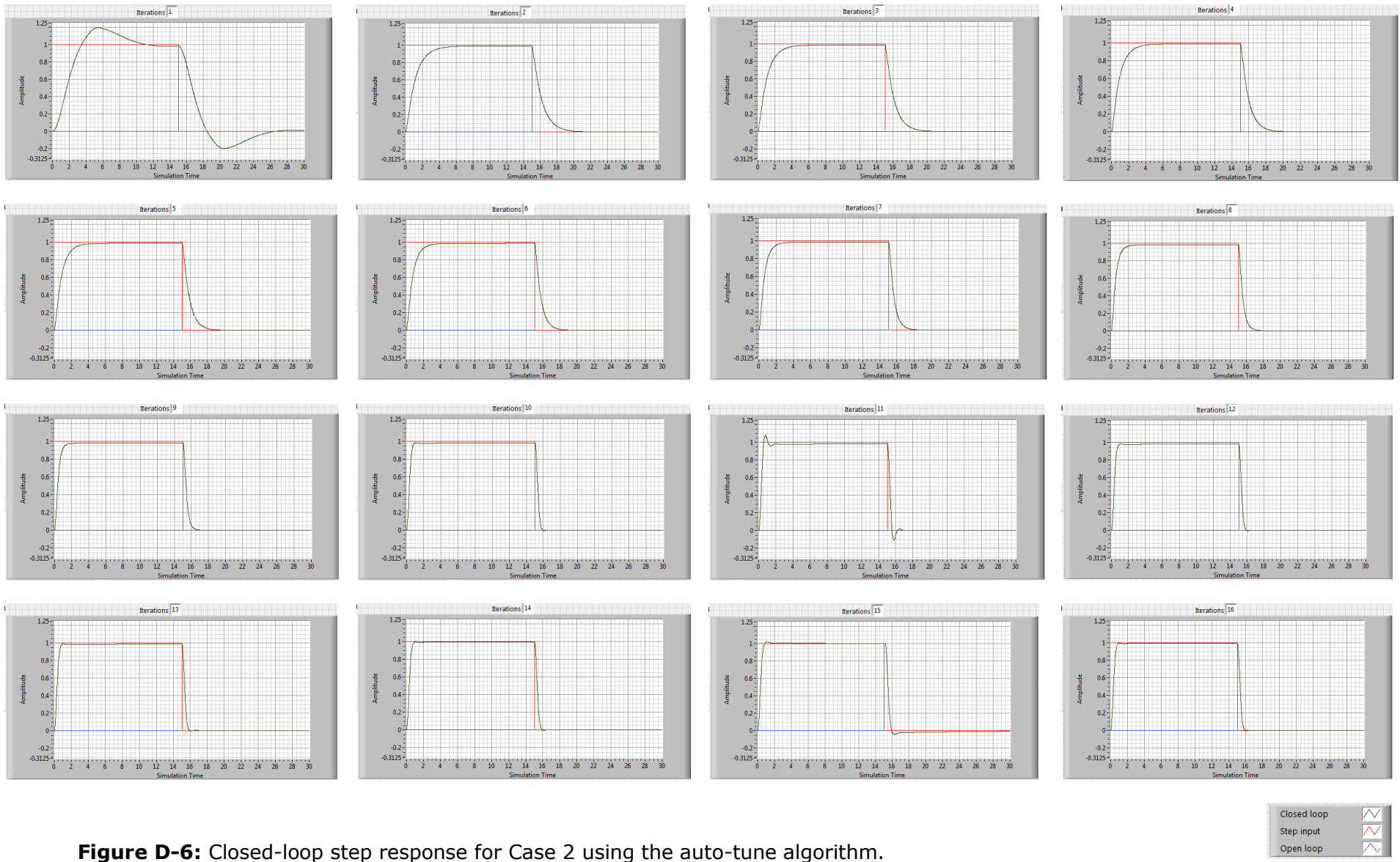
Figure D-6: Closed-loop step response for Case 2 using the auto-tune algorithm.
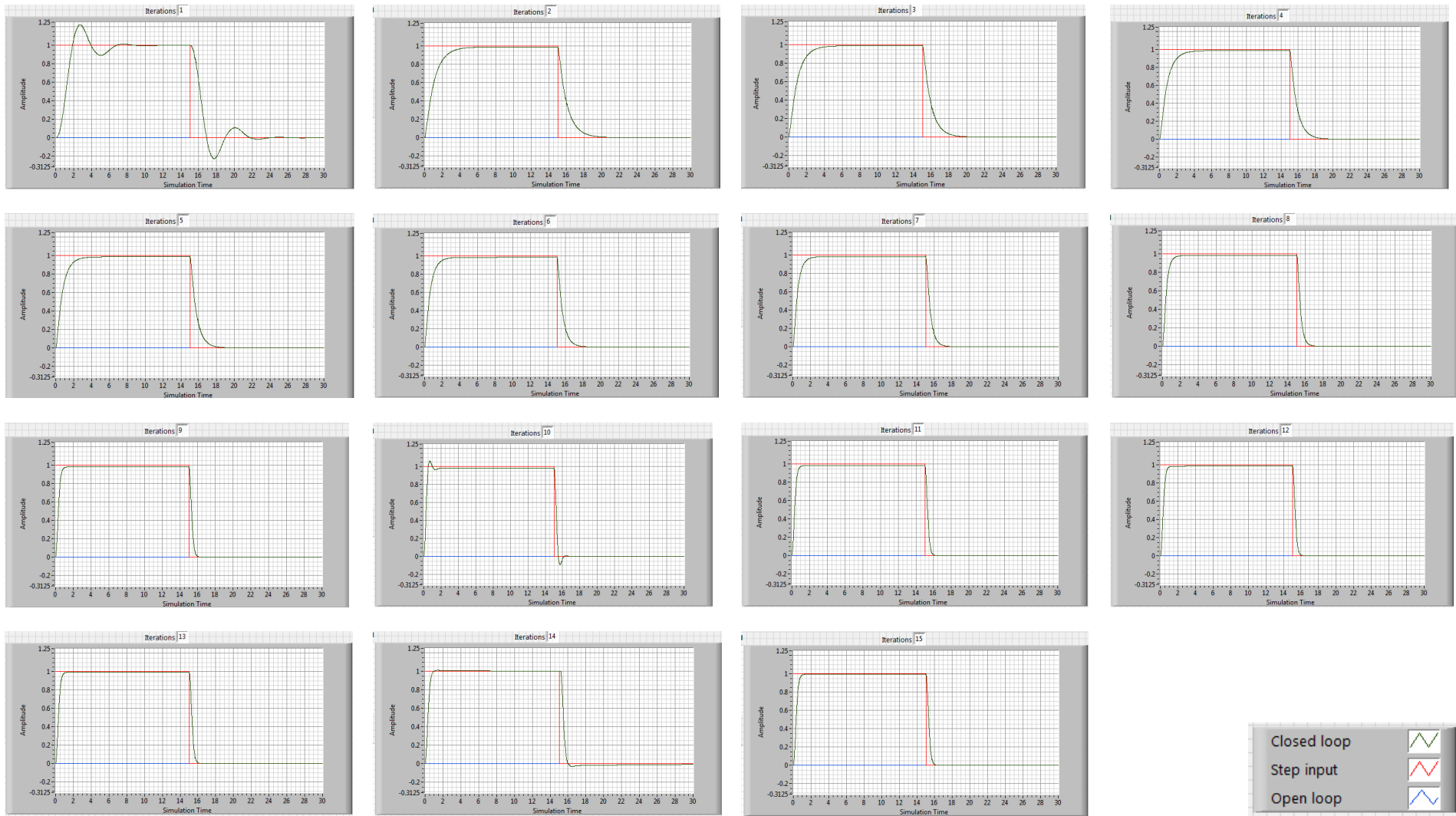
Figure D-7: Closed-loop step response for Case 4 using the auto-tune algorithm.
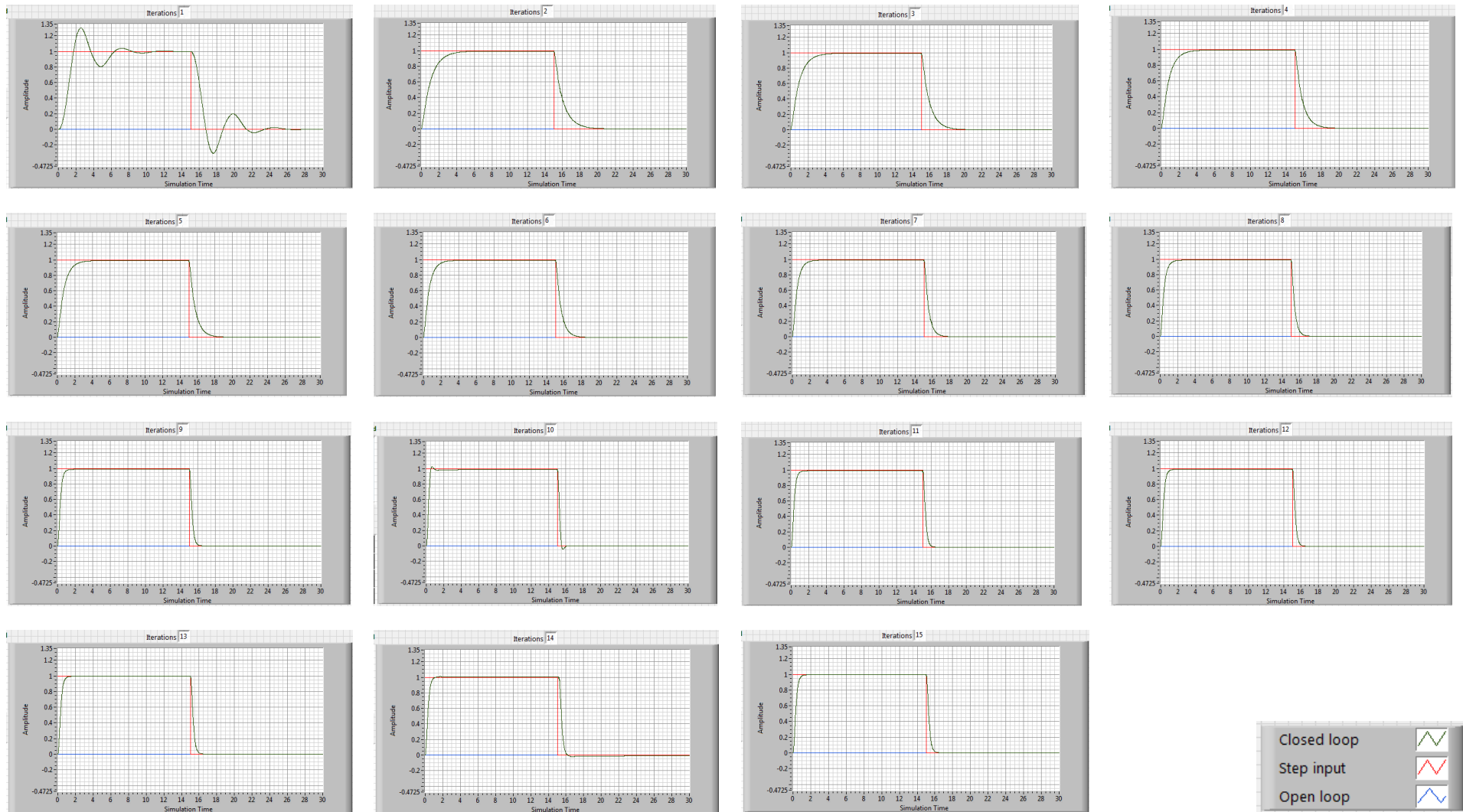
Figure D-8: Closed-loop step response for Case 5 using the auto-tune algorithm

Table D-1: Controller gains' values and closed-loop characteristics of Case 1 using the auto-tuning algorithm.

Iteration	Controller Gain				ISE	Closed-loop Characteristics	
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p
1	1	1	1	1	1.186	2.028	16.384
2	1	1	0.031	16.384	0.647	2.467	0
3	1	0.9	0.031	16.384	0.594	2.238	0
4	1	0.8	0.031	16.384	0.541	2.005	0
5	1	0.7	0.031	16.384	0.488	1.765	0
6	1	0.6	0.031	16.384	0.437	1.518	0
7	1	0.5	0.031	16.384	0.388	1.216	0
8	1	0.4	0.031	16.384	0.344	0.988	0
9	1	0.3	0.031	16.384	0.3	0.699	0
10	1	0.2	0.031	16.384	0.272	0.47	0.697
11	1	0.3	0.031	16.384	0.3	0.699	0
12	1	0.3	0.061	16.384	0.296	0.683	0
13	1	0.3	0.122	16.384	0.29	0.655	0
14	1	0.3	0.244	16.384	0.287	0.612	2.606
15	1	0.3	0.122	16.384	0.29	0.655	0

Table D-2: Controller gains' values and closed-loop characteristics of Case 2 using the auto-tuning algorithm.

Iteration	Controller Gain				ISE	Closed-loop Characteristics	
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p
1	1	1	1	1	1.374	2.352	19.595
2	1	1	0.026	19.595	0.651	2.502	0
3	1	0.9	0.026	19.595	0.597	2.275	0
4	1	0.8	0.026	19.595	0.543	2.044	0
5	1	0.7	0.026	19.595	0.489	1.808	0
6	1	0.6	0.026	19.595	0.437	1.566	0
7	1	0.5	0.026	19.595	0.386	1.316	0
8	1	0.4	0.026	19.595	0.338	1.056	0
9	1	0.3	0.026	19.595	0.295	0.788	0
10	1	0.2	0.026	19.595	0.261	0.51	0
11	1	0.1	0.026	19.595	0.251	0.385	8.031
12	1	0.2	0.026	19.595	0.261	0.51	0
13	1	0.2	0.051	19.595	0.258	0.504	0
14	1	0.2	0.102	19.595	0.254	0.493	0
15	1	0.2	0.204	19.595	0.252	0.475	2.064
16	1	0.2	0.102	19.595	0.254	0.493	0

Table D-3: Controller gains' values and closed-loop characteristics of Case 4 using the auto-tuning algorithm.

Iteration	Controller Gain				ISE	Closed-loop Characteristics	
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p
1	1	1	1	1	0.924	1.203	22.258
2	1	1	0.022	22.258	0.6	2.284	0
3	1	0.9	0.022	22.258	0.545	2.054	0
4	1	0.8	0.022	22.258	0.491	1.82	0
5	1	0.7	0.022	22.258	0.438	1.581	0
6	1	0.6	0.022	22.258	0.386	1.336	0
7	1	0.5	0.022	22.258	0.335	1.081	0
8	1	0.4	0.022	22.258	0.287	0.813	0
9	1	0.3	0.022	22.258	0.245	0.537	0
10	1	0.2	0.022	22.258	0.217	0.339	6.282
11	1	0.3	0.022	22.258	0.245	0.537	0
12	1	0.3	0.044	22.258	0.243	0.53	0
13	1	0.3	0.088	22.258	0.24	0.519	0
14	1	0.3	0.176	22.258	0.238	0.498	1.224
15	1	0.3	0.088	22.258	0.24	0.519	0

Table D-4: Controller gains' values and closed-loop characteristics of Case 5 using the auto-tuning algorithm.

Iteration	Controller Gain				ISE	Closed-loop Characteristics	
	G_E	G_{CE}	G_{IE}	G_U		t_r	M_p
1	1	1	1	1	0.97	1.108	30.096
2	1	1	0.017	30.096	0.582	2.253	0
3	1	0.9	0.017	30.096	0.53	2.029	0
4	1	0.8	0.017	30.096	0.477	1.8	0
5	1	0.7	0.017	30.096	0.424	1.567	0
6	1	0.6	0.017	30.096	0.371	1.329	0
7	1	0.5	0.017	30.096	0.32	1.085	0
8	1	0.4	0.017	30.096	0.27	0.83	0
9	1	0.3	0.017	30.096	0.225	0.56	0
10	1	0.2	0.017	30.096	0.19	0.325	2.487
11	1	0.3	0.017	30.096	0.225	0.56	0
12	1	0.3	0.034	30.096	0.223	0.554	0
13	1	0.3	0.068	30.096	0.221	0.544	0
14	1	0.3	0.136	30.096	0.219	0.525	0.776
15	1	0.3	0.068	30.096	0.221	0.544	0

E. LabVIEW VIs

The sections below illustrate the main LabVIEW structures and VIs used in the design of the intelligent wireless fuzzy controller network which are extracted from the Help file of LabVIEW 2011.

LabVIEW Control Design and Simulation Module

The Control Design and Simulation Module provides tools to create models from first principles using transfer function, state-space, or zero-pole-gain representation. You also can analyse and synthesize open-loop and closed-loop model behaviour, design controllers, simulate online and offline systems, and conduct physical implementations.

Fuzzy System Designer

Requires: PID and Fuzzy Logic Toolkit

Use this dialog box to design and test fuzzy systems. You also can use the Fuzzy Logic VIs to design, control, and modify fuzzy systems programmatically.

While Loop

Owning Palette: Structures

Requires: Base Package

Repeats the subdiagram inside it until the conditional terminal, an input terminal, receives a particular Boolean value. The Boolean value depends on the continuation behaviour of the While Loop. Right-click the conditional terminal and select **Stop if True** or **Continue if True** from the shortcut menu. You also can wire an error cluster to the conditional terminal, right-click the terminal, and select **Stop on Error** or **Continue while Error** from the shortcut menu. The While Loop always executes at least once.

The iteration (**i**) terminal provides the current loop iteration count, which is zero for the first iteration. If iteration count exceeds 2,147,483,647, or 2^{31}-1, the iteration terminal remains at 2,147,483,647 for all further iterations. If you need to keep count of more than 2,147,483,647 iterations, you can use shift registers with a greater integer range.

If you select a While Loop on the Execution Control Express VIs and Structures palette and place it on the block diagram, a stop button also appears on the block diagram and is wired to the conditional terminal. If you select a While Loop on the Structures palette and place it on the block diagram, a stop button does not appear.

After you create a While Loop, you can use shift registers to pass values from one iteration to the next. If you wire an array to a While Loop, you can read and process every element in that array by enabling auto-indexing.

To convert a While Loop to a For Loop, right-click the While Loop and select **Replace with For Loop** from the shortcut menu.**(ETS, VxWorks, Windows)** To convert a While Loop to a Timed Loop, right-click the While Loop and select **Replace with Timed Loop** from the shortcut menu.

Control & Simulation Loop

Owning Palette: Simulation VIs and Functions

Requires: Control Design and Simulation Module

Executes the simulation diagram until the Control & Simulation Loop reaches the simulation final time or until the Halt Simulation function stops the execution programmatically. You must place all Simulation functions within a Control & Simulation Loop or in a simulation subsystem. You also can place simulation subsystems within a Control & Simulation Loop or another simulation subsystem, or you can place simulation subsystems on a block diagram outside a Control & Simulation Loop or run the simulation subsystems as stand-alone VIs.
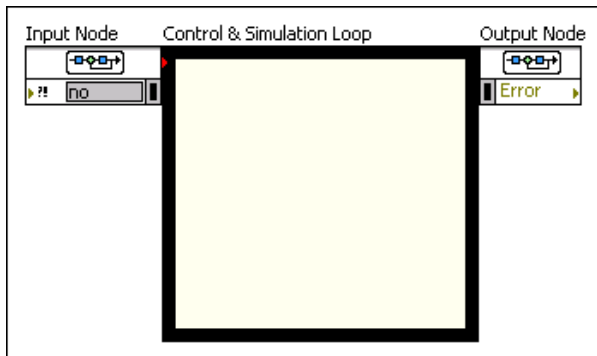
The Control & Simulation Loop has an Input Node and an Output Node. Use the Input Node to configure simulation parameters programmatically. You also can configure these parameters interactively using the Configure Simulation Parameters dialog box. Access this dialog box by double-clicking the Input Node or by right-clicking the border and selecting **Configure Simulation Parameters** from the shortcut menu.

FL Fuzzy Controller VI

Owning Palette: Fuzzy Logic VIs

Requires: PID and Fuzzy Logic Toolkit

Implements a fuzzy logic controller for the fuzzy system you specify.

By default, this VI implements a fuzzy logic controller for a single-input single-output (SISO) fuzzy system. You must manually select the polymorphic instance you want to use.

Implements a fuzzy logic controller for a multiple-input single-output (MISO) fuzzy system.
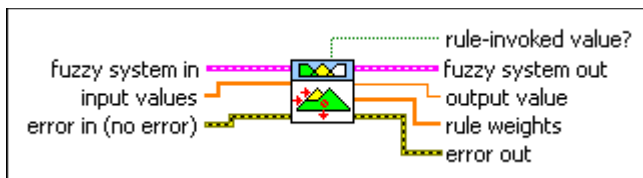
fuzzy system in specifies the complete information for a fuzzy system. Wire the **fuzzy system out** output from another VI to the **fuzzy system in** input of this VI.

input values specifies the values of the input variables in the fuzzy system. The fuzzy logic controller evaluates the **output value(s)** according to the **input values** and the rules of the fuzzy

system.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

rule-invoked value? indicates whether the fuzzy logic controller invoked a rule to evaluate the corresponding **output value**. **output value** is zero either if the fuzzy controller evaluates the output variable to zero based on the **input value(s)** and the rules of the fuzzy system or if the fuzzy logic controller does not invoke any rule to evaluate the output variable. **rule-invoked value?** indicates, when FALSE, that the rule base is incomplete.

fuzzy system out returns the complete information for a fuzzy system. Wire the **fuzzy system out** output of this VI to the **fuzzy system in** input of another VI.

output value returns the value of the output variable in the fuzzy system. The fuzzy logic controller evaluates the **output value** according to the **input value(s)** and the rules of the fuzzy system. If **output value** is zero, use the **rule-invoked value?** indicator to determine whether the fuzzy controller evaluated the corresponding output variable to zero or if the fuzzy logic controller did not invoke any rule to evaluate the output variable.

rule weights returns the rule weights that the fuzzy logic controller uses to scale the membership functions of the output linguistic variables. The implication method specifies how the fuzzy logic controller performs this scaling. For each rule, the rule weight is the truth value of the aggregated antecedent multiplied by the degree of support you specify for the rule.

error out contains error information. This output provides standard error out functionality.

FL Load Fuzzy System VI

Owning Palette: Fuzzy Logic VIs

Requires: PID and Fuzzy Logic Toolkit

Loads a fuzzy system from a `.fs` file. Use the FL Save Fuzzy System VI to save the `.fs` file after you make any modifications. You also can load and save `.fs` files in the Fuzzy System Designer.

file path specifies the path to the `.fs` file. You must specify an absolute path. If you specify an empty or relative path, this VI returns an error.

error in describes error conditions that occur before this node runs. This input provides standard error infunctionality.

fuzzy system out returns the complete information for a fuzzy system. Wire the **fuzzy system out** output of this VI to the **fuzzy system in** input of another VI.

error out contains error information. This output provides standard error out functionality.

CD Construct Transfer Function Model VI

Owning Palette: Model Construction VIs

Requires: Control Design and Simulation Module

Creates a transfer function representation of a system using the **Sampling Time (s),Numerator, Denominator,** and **Delay**. This VI also produces a transfer function model which specifies the data in symbolic form. You must manually select the polymorphic instance to use.

Details
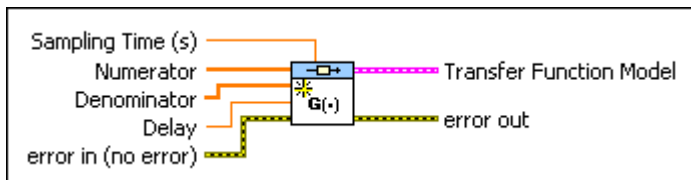
Sampling Time (s) defines whether the model represents a continuous-time system or a discrete-time system. If the model represents a continuous-time system, **Sampling Time (s)** must equal 0. If the model represents a discrete-time system, **Sampling Time (s)** must be greater than 0 and equal to the sampling rate, in seconds, of the discrete system. A value of -1 specifies that **Sampling Time (s)** is irrelevant. The default is 0.

> **Note** If you use the inputs to create a continuous-time system, setting the **Sampling Time (s)** to a value greater than 0 does not yield the discrete-time equivalent of the system. You must use the CD Convert Continuous to Discrete VI to convert the continuous-time system to the discrete-time equivalent of the system.

Numerator contains the constant coefficients, in ascending order, of a polynomial that represents the numerator of a SISO transfer function. The coefficients take the following form: $b_0 + b_1s + ... + b_ms^m$.

Denominator contains the constant coefficients, in ascending order, of a polynomial that represents the denominator of a SISO transfer function. The coefficients take the following form: $a_0 + a_1s + ... + a_ns^n$.

Delay is the transport time delay that might exist in the system. Refer to the LabVIEW Control Design User Manual for more information about delays.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

Transfer Function Model is the system model this VI creates. When the sampling time is zero (for

continuous-time), the **Numerator** and **Denominator** collectively represent the [mathematical model](#) (in Laplace transformation) of a dynamic system *H(s)* that provides the relationship between the input *U(s)* and output *Y(s)* of the system. To access and modify the data in the model, use the [Model Information](#) VIs.

[📁] **error out** contains error information. This output provides [standard error out](#) functionality.

CD Draw Transfer Function Equation VI

Owning Palette: Model Construction VIs

Requires: Control Design and Simulation Module

Displays the transfer function equation of the model. Wire data to the **State-Space Model** input to determine the polymorphic instance to use or manually select the instance.

Note Real-time targets do not support this VI.

Details
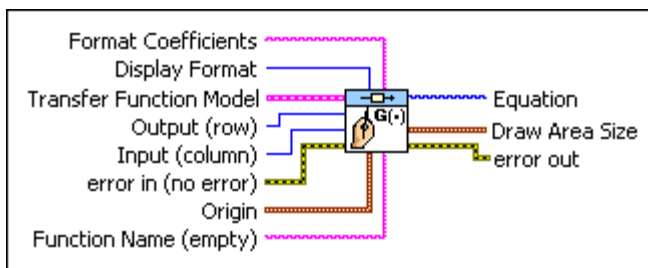
[abc] **Format Coefficients** specifies how to format the resulting equation. You can enter any of the [LabVIEW format specifier syntax elements](#). The default value is `%#_g`, which removes trailing zeros and specifies that LabVIEW uses fractional or scientific notation depending on the exponent of the number.

[U16] **display format** specifies the format in which this VI displays the equation.

0	**Descending**
1	**Ascending**
2	**Negative Power**

[📁] **Transfer Function Model** contains a [mathematical representation](#) of and [information](#) about the system for which this VI draws an equation.

[I32] **Output (row)** specifies the index number of the output row from which to draw the transfer function matrix. The index is zero-based. The default is −1, which draws all outputs.

[I32] **Input (column)** specifies the index number of the input column from which to draw the transfer

188

function matrix. The index is zero-based. The default is –1, which draws all inputs.

[icon] **error in** describes error conditions that occur before this node runs. This input provides standard error in functionality.

[icon] **Origin** specifies the upper-left position of the equation this VI draws.

    [I16] **x** specifies the *x*-coordinate of the origin. The default value is 10.

    [I16] **y** specifies the *y*-coordinate of the origin. The default value is 10.

[abc] **Function Name** specifies the function this VI draws. For example, if you enter H, this VI displays H(s)= for continuous equations and H(z)= for discrete equations.

[icon] **Equation** draws a picture of the model equation this VI defines.

[icon] **Draw Area Size** indicates the size of the area in the picture control this VI uses to draw the model equation.

    [U32] **Width** indicates the width of the area in the picture control this VI uses to draw the model equation.

    [U32] **Height** indicates the height of the area in the picture control this VI uses to draw the model equation.

[icon] **error out** contains error information. This output provides standard error out functionality.

CD Step Response VI

Calculates the output of the system when a step input excites it. This VI assumes the initial states of the system to be zero. Wire data to the **State-Space Model** input to determine the polymorphic instance to use or manually select the instance.

Details
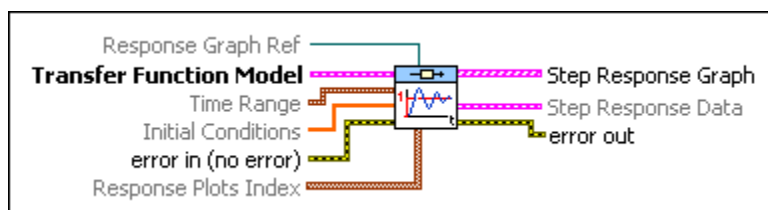
[icon] **Response Graph Ref** is a reference to the **Step Response** graph. **Response Graph Ref** configures the x-scale, y-scale, and legend properties. If you want to use the default settings or customize the settings for these properties, do not wire a value to this input.

Transfer Function Model contains a [mathematical representation](#) of and [information](#) about the system of which this VI calculates step response.

Time Range contains information about the initial time, final time, and time step.

> **t0** is the initial time in seconds in the **Step Response** graph. The default is −1.

> **dt** is the constant interval between successive values in the time vector. The default is −1.

> **tf** is the final time in seconds up to which this VI calculates the step response. The default is −1.

Initial Conditions specifies the initial values of the states or outputs. The default is 0.

error in describes error conditions that occur before this node runs. This input provides [standard error in](#) functionality.

Response Plots Index specifies the index number of the inputs and outputs of the system.

> **Input #** is the index number of the specific input to the system. This VI displays the response to this input on the **Step Response** graph. The index is zero-based.

> **Output #** is the index number of the specific output of the system that this VI displays on the **Step Response** graph. The index is zero-based.

Step Response Graph displays a graph that shows the forced response of the system when the forcing function is a step. For MIMO systems, this VI determines the step response by applying a step on one input at a time and letting other inputs to the system be zero.

Step Response Data returns information about the step response. To access the **Step Response Data**, use the [CD Get Time Response Data](#) VI.

> **Time** is the uniformly-spaced time vector against which this VI plots the step response and the state trajectories.

> **Outputs Data** returns data about the time response of the outputs to the inputs. Refer to the [Details](#) section for more information about the **Outputs Data**.

> **States Data** returns data about the time response of the states to the inputs. For transfer function and zero-pole-gain models, this array is empty. Refer to the [Details](#) section for more information about the **States Data**.

error out contains error information. This output provides [standard error out](#) functionality.
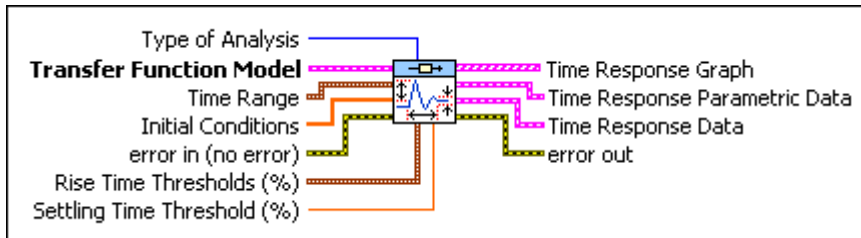
CD Parametric Time Response VI

Owning Palette: Time Response VIs

Requires: Control Design and Simulation Module

Calculates parametric information, such as rise time, peak time, settling time, steady-state gain, overshoot, and peak value of an input model based on time response data. If you use the Internal instances of this VI, this VI internally calculates the time response data for which it calculates the parametric information. If you use the External instances of this VI, you must specify the time response data for which this VI calculates the parametric information. Wire data to the **State-Space Model** or **Time Range** inputs to determine the polymorphic instance to use or manually select the instance.

Details

Type of Analysis specifies the type of time response analysis this VI performs on the model.

0	**Step Response** (default)—Specifies this VI uses a step response to obtain the parametric information.
1	**Impulse Response**—Specifies this VI uses an impulse response to obtain the parametric information.
2	**Initial Response**—Specifies this VI uses an initial response to obtain the parametric information.

Transfer Function Model contains a mathematical representation of and information about the system of which this VI calculates parametric information.

Time Range contains information about the initial time, final time, and time step.

$\quad$ **t0** is the initial time in seconds in the **Step Response** graph. The default is −1.

$\quad$ **dt** is the constant interval between successive values in the time vector. The default is −1.

$\quad$ **tf** is the final time in seconds up to which this VI calculates the step response. The default is −1.

Initial Conditions are the initial values the parametric response uses.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

Rise Time Thresholds (%) specifies the lower and upper thresholds that define the rise time this VI returns. By default, the rise time is the time required for the system response to rise from 10% to 90% of the settling point.

This VI calculates rise time by performing a step response and measuring the time required for the system response to rise from the **Lower** percentage of the final steady-state value to the **Upper** percentage of the final steady-state value. If a system has a step response where the initial overshoot is in a direction opposite to that of the final steady-state value, that portion of the step response does not affect the calculation of the rise time.

> **Lower** specifies the lower limit of the rise time threshold. The default value is 10%.

> **Upper** specifies the upper limit of the rise time threshold. The default value is 90%.

Settling Time Threshold (%) defines the percentage in which the signal must fall to be within range of the steady-state value. The default is 1%. Therefore, the settling time is the time required for the signal to fall within a 1% range of the steady-state value.

Time Response displays an XY graph containing the time response of the model.

Time Response Parametric Data returns the parametric time response data this VI measures from the **Time Response Data**.

> **Rise Time (s)** is the time required for the dynamic system response to rise from 10% of its final value to 90% of its final value.

> **Peak Time (s)** is the time required for the dynamic system response to reach the peak value of its first overshoot.

> **Settling Time (s)** is the time required for the response to reach 1% of its final value.

> **Overshoot** is the dynamic system response value that most exceeds unity, expressed as a percent.

> **Steady-State Gain** is the final value of the signal after transient responses decay.

> **Peak value** returns the value at which the maximum absolute value of the time response occurs.

Time Response Data returns the data before this VI parameterizes it. To access the **Time Response Data**, use the CD Get Time Response Data VI.

> **Time** returns the uniformly-spaced time vector against which this VI plots the impulse,

initial, or step response and the state trajectories.

[DBL] **Outputs Data** returns data about the time response of the outputs to the inputs. Refer to the Details section for more information about the **Outputs Data**.

[DBL] **States Data** returns data about the time response of the states to the inputs. For transfer function and zero-pole-gain models, this array is empty. Refer to the Details section for more information about the **States Data**.

error out contains error information. This output provides standard error out functionality.

Array Max & Min Function

Owning Palette: Array Functions

Requires: Base Package

Returns the maximum and minimum values found in **array**, along with the indexes for each value.

The connector pane displays the default data types for this polymorphic function.

Details
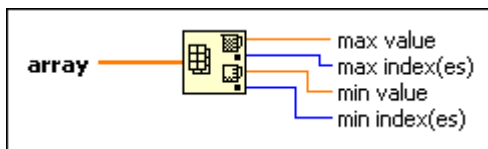
[DBL] **array** can be an *n*-dimensional array of any type.

[DBL] **max value** is of the same data type and structure as the elements in **array**.

[I32] **max index(es)** is the index for the first max value. If **array** is multidimensional, **max index(es)** is an array whose elements are the indexes for the first maximum value in **array**.

[DBL] **min value** is of the same data type and structure as the elements in **array**.

[I32] **min index(es)** is the index for the first min value. If **array** is multidimensional, **min index(es)** is an array whose elements are the indexes for the first minimum value in **array**.

Threshold 1D Array Function

Owning Palette: Array Functions

Requires: Base Package

Interpolates points in a 1D array that represents a 2D non-descending graph. This function compares **threshold y** to the values in **array of numbers or points** starting at **start index** until it finds a pair of consecutive elements such that **threshold y** is greater than the value of the first element and less than or equal to the value of the second element.

The connector pane displays the default data types for this polymorphic function.

Details
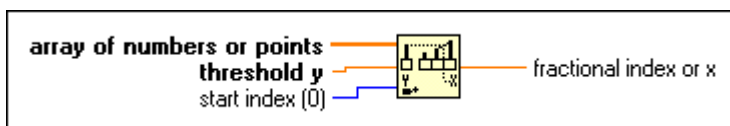
[DBL] **array of numbers or points** can be an array of numbers or an array of points where each point is a cluster of x- and y-coordinates. If this input is an array of points, this function uses the second elements in the clusters, or the y-coordinates, to obtain a fractional index that it then uses to interpolate the corresponding x value.

[EXT] **threshold y** is the threshold value for the function. If **threshold y** is less than or equal to the array value at **start index**, the function returns **start index** for **fractional index or x**. If **threshold y** is greater than every value in the array, the function returns the index of the last value. If the array is empty, the function returns `NaN`.

[I32] **start index** must be a number. The default is 0, which means the function returns the result calculated from the entire array, rather than a specified section of the array.

[DBL] **fractional index or x** is the interpolated result LabVIEW calculates for the **array of numbers or points** 1D input array. For example, suppose **array of numbers or points** is an array of four numbers [4, 5, 5, 6], **start index** is 0, and **threshold y** is 5. The **fractional index or x** is 1, corresponding to the index of the first value of 5 the function finds. Suppose the array elements are 2.3, 5.2, 7.8, 7.9, 10.0, the **start index** is 0, and the **threshold y** is 6.5. The output is 1.5 because 6.5 is halfway between 5.2 (index 1) and 7.8 (index 2). If **threshold y** is 7 for the same set of numbers, the output is 1.69. If **threshold y** is 14.2, **start index** is 5, and the values in the array starting at index 5 are 9.1, 10.3, 12.9, and 15.5, **threshold y** falls between elements 7 and 8 because 14.2 is midway between 12.9 and 15.5. The value for **fractional index or x** is 7.5, that is, halfway between 7 and 8. If the array input consists of an array of points where each point is a cluster of x- and y-coordinates, the output is the interpolated x value corresponding to the interpolated position of **threshold y** among the y-coordinates, rather than the fractional index of the array. If the interpolated position of **threshold y** is midway between the y values at indexes 4 and 5 of the array with x values of −2.5 and 0 respectively, the output is not an index value of 4.5 as it would be for a numeric array, but rather an x value of −1.25. In other words, this function returns the interpolated x value associated with the given y value if you graphed the points. This function works the same for arrays of numbers as it does for arrays of points. If you have an array of numbers, this function assumes the x-coordinates are the same as the indexes of the array itself. In other words, this function assumes the points are uniformly spaced.

DAQ Assistant Express VI

Creates, edits, and runs tasks using NI-DAQmx.
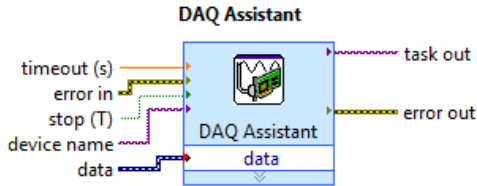
Creates, edits, and runs tasks using NI-DAQmx. Refer to the NI-DAQmx Readme for a complete listing of devices NI-DAQmx supports.

When you place this Express VI on the block diagram, the DAQ Assistant launches to create a new task. After you create a task, you can double-click the DAQ Assistant Express VI to edit that task. For continuous measurement or generation, place a while loop around the DAQ Assistant Express VI.

For continuous single-point input or output, the DAQ Assistant Express VI might not provide optimal performance. Refer to the Cont Acq&Graph Voltage-Single Point Optimization VI in examples\DAQmx\Analog In\Measure Voltage.llb for an example of techniques to create higher-performance, single-point I/O applications.

References

Ahny, H.-S., Chenz, Y., & Moorex, K. L. (2011, 28-30 Sept. 2011). *Multi-agent coordination by iterative learning control: centralized and decentralized strategies.* Paper presented at the 2011 IEEE International Symposium on Intelligent Control (ISIC), Denver, CO, USA.

Altas, I. H., & Sharaf, A. M. (2007). A generalized direct approach for designing fuzzy logic controllers in Matlab/Simulink GUI environment. *International Journal of Information Technology and Intelligent Computing*.

Amtel. (2013). MCU Wireless Microcontrollers Retrieved from http://www.atmel.com/products/wireless/zigbee/

Antsaklis, P. J. (1994). Defining Intelligent Control *Report of the IEEE Control Systems Society Task Force on Intelligent Control* (Vol. 14, pp. 4-5 & 58-66).

Antsaklis, P. J. (1999). Intelligent Control. In I. Control (Ed.), *Encyclopedia of Electrical and Electronics Engineering*: John Wiley & Sons, Inc.

AOS. (2013). JACK. Retrieved October 15, 2013, from http://aosgrp.com/products/jack/

Åström, K. (1989). Toward intelligent control. *Control Systems Magazine, IEEE, 9*(3), 60-64. doi: 10.1109/37.24813

Åström, K., & Murray, R. (2009). *Feedback systems: an introduction for scientists and engineers*: Princeton University Press.

Åström, K., & Wittenmark, B. r. (1995). *Adaptive control*. Reading, Mass: Addison-Wesley.

Babuska, R., & Mamdani, E. H. (2008). Fuzzy Control. http://www.scholarpedia.org/article/Fuzzy_control

Berezyuk, T. (2013). Icons-land. Retrieved May 1, 2013, from www.icons-land.com

Bitter, R., Mohiuddin, T., & Nawrocki, M. (2007). *LabView: advanced programming techniques* (Second Edition ed.): CRC Press. Taylor & Francis Group.

Breemen, A. v., & Vries, T. d. (2000). *An Agent-Based Framework for Designing Multi-Controller Systems.* Paper presented at the Proceedings of the Fifth International Conference on The Practical Applications of Intelligent Agents and Multi-Agent Technology, Manchester, U.K.

Choi, J., Oh, S., & Horowitz, R. (2009). Distributed learning and cooperative control for multi-agent systems. *Automatica, 45*(12), 2802-2814. doi: 10.1016/j.automatica.2009.09.025

Chopra, S., Mitra, R., & Kumar, V. (2008). Auto Tuning of Fuzzy PI Type Controller Using Fuzzy Logic. *International Journal of Computational Intelligent, 6*(1), 12-18.

Chung, H. Y., Chen, B. C., & Lin, J. J. (1998). A PI-type fuzzy controller with self-tuning scaling factors. *Fuzzy Sets and Systems, 93*(1), 23-28.

Dalci, K. B., Uzunoglu, M., & Kucukdemiral, I. B. (2004). Genetic algorithm based optimal self-tuning fuzzy logic controller for power system static VAR stabiliser. *INTERNATIONAL JOURNAL OF ELECTRICAL ENGINEERING EDUCATION, 41*(1), 71-19.

de Silva, C. W. (1995). *Intelligent control: fuzzy logic applications*. Boca Raton: CRC Press.

Deliyannis, T., Sun, Y., & Fidler, J. K. (1998). *Continuous-time active filter design*: CRC Press.Taylor & Francis Group.

Dorf, R., & Bishop, R. (2001). *Modern control systems*. Upper Saddle River: Prentice Hall.

Driankov, D., Hellendoorn, H., & Reinfrank, M. (1996). *An introduction to fuzzy control*. New York: Springer.

Duan, X.-G., Deng, H., & Li, H.-X. (2013). A Saturation-Based Tuning Method for Fuzzy PID Controller. *IEEE Transactions on Industrial Electronics, 60*(11), 5177-5185.

Eberhart, R., & Shi, Y. (2007). *Computational intelligence: concepts to implementations*. US: Morgan Kaufmann.

Edgar, T. F., Seborg, D. E., & Mellichamp, D. A. (2004). *Process dynamics and control*. Hoboken, N.J: Wiley.

Elahi, A., & Gschwender, A. (2009). *ZigBee wireless sensor and control Network*: Prentice Hall.

Engelbbrecht, A. (2007). *Computational intelligence: an introduction* (Second Edition ed.): John Wiley & Sons, Ltd.

Fabio, B., Giovanni, C., & Dominic, G. (2007). *Developing multi-agent systems with JADE*: Wiley-Blackwell.

Fu, K. (1971). Learning control systems and intelligent control systems: An intersection of artifical intelligence and automatic control. *Automatic Control, IEEE Transactions on, 16*(1), 70-72. doi: 10.1109/tac.1971.1099633

Gang, F. (2006). A Survey on Analysis and Design of Model-Based Fuzzy Control Systems. *Fuzzy Systems, IEEE Transactions on, 14*(5), 676-697. doi: 10.1109/TFUZZ.2006.883415

Gao, Z., Trautzsch, T. A., & Dawson, J. G. (2002). A stable self-tuning fuzzy logic control system for industrial temperature regulation. *Industry Applications, IEEE Transactions on, 38*(2), 414-424.

Gislason, D. (2008). *Zigbee wireless networking*. Oxford: Newnes/Elsevier.

Golnaraghi, M. F., & Kuo, B. C. (2010). *Automatic control systems*. Hoboken, N.J: Wiley.

Grigorie, L. (Ed.). (2011). *Fuzzy controllers, theory and applications*: InTech.

Haugen, F. (2009). Second Order Systems: TechTeach.

Hoffmann, F. (2001). Evolutionary algorithms for fuzzy control system design. *Proceedings of the IEEE, 89*(9), 1318-1333. doi: 10.1109/5.949487

Iqbal, S., Boumella, N., & Garcia, J. C. F. (Eds.). (2012). *Fuzzy controllers- recent advances in theory and applications*: InTech.

JADE. (2013). Java Agent DEvelopment Framework. Retrieved October 10, 2013, from http://jade.tilab.com/

Jantzen, J. (2007). *Foundations of fuzzy control*. Chichester: Wiley.

Jennings, N. R., & Bussmann, S. (2003). Agent-based control systems: Why are they suited to engineering complex systems? *Control Systems, IEEE, 23*(3), 61-73. doi: 10.1109/mcs.2003.1200249

Jie, X., Liyun, L., Derong, Yanbo, C., & Shiyu, W. (2008, 16-18 July 2008). *Fuzzy gain based adaptive fuzzy logic controller for BLDCM drive.* Paper presented at the Control Conference, 2008. CCC 2008. 27th Chinese.

Johnson, J., & Picton, P. (2001). *Designing intelligent machines*. Oxford: Butterworth-Heinemann in association with the Open University.

Jorge Ierache, R. G.-M., Armando De Giusti. (2010). Learning by collaboration in intelligent autonomous systems. *International Federation for Information Processing (IFIP) Advances in Information and Communication Technology (AICT), 331*, 143-152.

Juang, C.-F., & Chang, P.-H. (2010). Designing fuzzy-rule-based systems using continuous ant-colony optimization. *IEEE Transactions on Fuzzy Systems, 18*(1), 138-149.

Kanagaraj, N., Sivashanmugam, P., & Paramasivam, S. (2008). Fuzzy Coordinated PI Controller: Application to the Real-Time Pressure Control Process. *Advances in Fuzzy Systems, 2008*.

Kanagaraj, N., Sivashanmugam, P., & Paramasivam, S. (2009). A fuzzy logic based supervisory hierarchical control scheme for real time pressure control. *International Journal of Automation and Computing, 6*(1), 88-96.

Karr, C. (1999). *Practical applications of computational intelligence for adaptive control*. Boca Raton [Fla.]: CRC Press.

Kelly, I. D., & Keating, D. A. (1998). Faster learning of control parameters through sharing experiences of autonomous mobile robots. *International Journal of Systems Science 29*(7), 783-793.

Khan, S., Salami Femi, A., Lawal Wahab, A., Alam, A. H. M. Z., Momoh Jimoh, E. S., Shihab Ahmed, H., . . . Mohd Rafiqul, I. (2008). Design and Implementation of an Optimal Fuzzy Logic Controller Using Genetic Algorithm. *Journal of Computer Science, 4*(10), 799.

Kia, P. J., Far, A. T., Omid, M., Alimardani, R., & Naderloo, L. (2009). Intelligent Control Based Fuzzy Logic for Automation of Greenhouse Irrigation System and Evaluation in Relation to Conventional Systems. *World Applied Sciences Journal, 6*(1), 16-23.

Kumar, M., & Garg, D. P. (2004). *Intelligent learning of fuzzy logic controllers via neural network and genetic algorithm.* Paper presented at the Japan/USA Conference on Flexible Automation.

Kumar, V., Nakra, B., & Mittal, A. (2011). A Review on Classical and Fuzzy PID Controllers. *International Journal of Intelligent Control and Systems, 6*(3), 170-181.

Lee, C. C. (1990). Fuzzy logic in control systems: fuzzy logic controller. I. *Systems, Man and Cybernetics, IEEE Transactions on, 20*(2), 404-418. doi: 10.1109/21.52551

Li, H.-X. (1997). A comparative design and tuning for conventional fuzzy control. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 27*(5), 884-889.

Lilly, J. H. (2011). *Fuzzy control and identification* Wiley-Blackwell.

Lima, P. U., & Saridis, G. N. (1996). *Design of intelligent control systems based on hierarchical stochastic automata*: World Scientific Publishing Co., Inc.

Liu, D., & Wang, F.-Y. (2006). *Advances in computational intelligence*. New Jersey: World Scientific.

Lu, Q., & Mahfouf, M. (2010). A model-free self-organizing fuzzy logic control system using a dynamic performance index table. *Transactions of the Institute of Measurement and Control, 32* (1), pp. 51–72.

Lu, W., Zhang, Y., & Xie, Y. (2011, June 29 2011-July 1 2011). *A multi-agent adaptive traffic signal control system using swarm intelligence and neuro-fuzzy reinforcement learning.* Paper presented at the 2011 IEEE Forum on Integrated and Sustainable Transportation System (FISTS), Vienna, Austria.

Makaya, C., & Pierre, S. (2012). *Emerging wireless networks: concepts, techniques, and applications*. Boca Raton, FL: CRC Press. Taylor & Francis Group.

Mamdani, E. H. (1974). Application of fuzzy algorithms for control of simple dynamic plant. *Electrical Engineers, Proceedings of the Institution of, 121*(12), 1585-1588. doi: 10.1049/piee.1974.0328

Mamdani, E. H. (1999). An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller. *International Journal of Human-Computer Studies, 51*(2), 135.

Mamdani, E. H., & Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies, 7*(1), 10-13.

Mann, G. K. I., Hu, B.-G., & Gosine, R. G. (2001). Two-level tuning of fuzzy PID controllers. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 31*(2), 263-269.

Mary, P. M., Marimuthu, N. S., & Singh, N. A. (2007). Design of Intelligent Self-Tuning Temperature Controller for Water Bath Process. *INTERNATIONAL JOURNAL OF IMAGING SCIENCE AND ENGINEERING (IJISE), VOL.1*(NO.4).

MathWorks. (2013). MathWorks. Retrieved April, 1, 2013, from http://www.mathworks.co.uk/

McArthur, S. D. J., Davidson, E. M., Catterson, V. M., Dimeas, A. L., Hatziargyriou, N. D., Ponci, F., & Funabashi, T. (2007). Multi-Agent Systems for Power Engineering Applications-Part I: Concepts, Approaches, and Technical Challenges. *22*, 1743-1752. doi: 10.1109/tpwrs.2007.908471

Millan, Y. A., Vargas, F., Molano, F., & Mojica, E. (2011, 1-4 Oct. 2011). *A Wireless Networked Control Systems review.* Paper presented at the Robotics Symposium, 2011 IEEE IX Latin American and IEEE Colombian Conference on Automatic Control and Industry Applications (LARC).

Mizumoto, M. (1992, 8-12 Mar 1992). *Realization of PID controls by fuzzy control methods.* Paper presented at the Fuzzy Systems, 1992., IEEE International Conference on.

Murad, M., Cheok, K. C., & Das, M. (2009). *Methodology to simplify the tuning process of self-organizing fuzzy logic controllers.* Paper presented at the Intelligent Engineering Systems, 2009. INES 2009. International Conference on.

Narendra, K. (1991). Intelligent control. *Control Systems, IEEE, 11*(1), 39-40. doi: 10.1109/37.103351

Narendra, K. (2005). From feedback control to complexity management: a personal perspective. In R. Murray-Smith & R. Shorten (Eds.), *Switching and learning in feedback systems* (Vol. 3355, pp. 1-30): Springer Berlin Heidelberg.

National Instruments. (2013a). LabVIEW Application Design Patterns. Retrieved January 1, 2013, from http://www.ni.com/white-paper/5237/en/

National Instruments. (2013b). LabVIEW System Design Software. Retrieved January 1, 2013, from http://www.ni.com/labview/

National Instruments. (2013c). NI PCI-6025E. Retrieved January 1, 2013, from http://sine.ni.com/nips/cds/view/p/lang/en/nid/10971

National Instruments. (2013d). Using the LabVIEW Shared Variable. Retrieved January 1, 2013, from http://www.ni.com/white-paper/4679/en

Nguyen, D.-H., & Huynh, T.-H. (2008, 17-20 Dec. 2008). *A SFLA-based fuzzy controller for balancing a ball and beam system.* Paper presented at the 10th International Conference on Control, Automation, Robotics and Vision (ICARCV).

NXP. (2013). 32-bit MCU and IEEE802.15.4 transceiver for low-power wireless networks. Retrieved October 15, 2013, from http://www.nxp.com/products/microcontrollers/wireless_microcontrollers/

OKAWA-Electric-Design. (2008). Sallen-Key Low-pass Filter Design Tool Retrieved April 1, 2013, from http://sim.okawa-denshi.jp/en/OPseikiLowkeisan.htm

Passino, K. (2005). *Biomimicry for optimization, control, and automation*. New York: Springer.

Passino, K., & Yurkovich, S. (1998). *Fuzzy control*. Menlo Park, Calif: Addison-Wesley.

Pham, D., Darwish, A. H., Eldukhr, E., & Otri, S. (2007). *Using the bees algorithm to tune a fuzzy logic controller for a robot gimnasta.* Paper presented at the Innovative Production Machines and Systems Virtual Conference, Cardiff, UK.

Pham, D. T., & Kalyoncu, M. (2009). *Optimisation of a fuzzy logic controller for a flexible single-link robot arm using the Bees Algorithm.* Paper presented at the Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on.

Pham, D. T., Soroka, A. J., Ghanbarzadeh, A., Koc, E., Otri, S., & Packianather, M. (2006). *Optimising Neural Networks for Identification of Wood Defects Using the Bees Algorithm.* Paper presented at the Industrial Informatics, 2006 IEEE International Conference on.

Pivonka, P. (2002, 2002). *Comparative analysis of fuzzy PI/PD/PID controller based on classical PID controller approach.* Paper presented at the Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on.

Ponce-Cruz, P., & Ramírez-Figueroa, F. (2010). *Intelligent control systems with LabVIEW*. London: Springer.

Procyk, T. J., & Mamdani, E. H. (1979). A linguistic self-organizing process controller. *Automatica, 15*(1), 15-30.

Ross, T. J. (2004). *Fuzzy logic with engineering applications*: John Wiley.

Rowell, D. (2004). Review of First- and Second-Order System Response. Massachusetts Institute of Technology

Ruano, A. (2005). *Intelligent control systems using computational intelligence techniques*. London: Institution of Electrical Engineers.

Rutkowski, L. (2008). *Computational intelligence: methods and techniques*. Berlin: Springer.

Saeed, B., & Mehrdadi, B. (2011). *Zero overshoot and fast transient response using a fuzzy logic controller.* Paper presented at the 17th International Conference on Automation and Computing (ICAC), Huddersfield, UK.

Saeed, B., & Mehrdadi, B. (2012a). Design of an iterative auto-tuning algorithm for a fuzzy PID controller Paper presented at the 25th International Congress on Condition Monitoring and Diagnostic Engineering, Huddersfield, UK.

Saeed, B., & Mehrdadi, B. (2012b). Designing a Wireless Network of Intelligent Fuzzy Logic Controllers With NI LabVIEW.

Saifizul, A. A., Zainon, Z., Osman, N. A. A., Azlan, C. A., & Ibrahim, U. (2006). Intelligent Control for Self-erecting Inverted Pendulum Via Adaptive Neuro-fuzzy Inference System. *American Journal of Applied Sciences*.

Sala, A., Guerra, T. M., & Babuška, R. (2005). Perspectives of fuzzy systems and control. *Fuzzy Sets and Systems, 156*(3), 432-444.

Schöllig, A., Alonso-Mora, J., & D'Andrea, R. (2010, 15-17 Dec. 2010). *Independent vs. joint estimation in multi-agent iterative learning control.* Paper presented at the 49th IEEE Conference on Decision and Control (CDC).

Shen, J.-C., & Chiang, H.-K. (2004, 20-23 July 2004). *PID tuning rules for second order systems.* Paper presented at the 5th Asian Control Conference, Taiwan.

Shen, Q. (2008). Special issue on UK Fuzzy Systems Research. *International Journal of Computational Intelligence Research, 4*(4), 297–299.

Shin, Y., & Xu, C. (2009). *Intelligent systems: modeling, optimization, and control*: CRC Press. Taylor & Francis Group.

Skogestad, S., & Postlethwaite, I. (2005). *Multivariable feedback control: analysis and design*. Chichester: John Wiley.

Soliman, M. S., Guan-zheng, T., & Abdullah, M. Y. (2008, 12-14 Dec. 2008). *Novel Adaptive Hybrid Optimization (AHO) Technique Using Biologically-Inspired Algorithms with FLC.* Paper presented at the Computer Science and Software Engineering, 2008 International Conference on.

Stewart, J., Stewart, P., Gladwin, D., & Parr, M. (2009). Multi-objective evolutionary-fuzzy augmented flight control for an F16 aircraft. *Proceedings of the Institution of Mechanical Engineers, 224*(G3), 293-309.

Takagi, T., & Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *Systems, Man and Cybernetics, IEEE Transactions on, SMC-15*(1), 116-132. doi: 10.1109/tsmc.1985.6313399

Tan, V., Yoo, D.-S., & Yi, M.-J. (2008a). *A Multiagent-System Framework for Hierarchical Control and Monitoring of Complex Process Control Systems*. Paper presented at the Proceedings of the 11th Pacific Rim International Conference on Multi-Agents: Intelligent Agents and Multi-Agent Systems, Hanoi, Vietnam.

Tan, V., Yoo, D.-S., & Yi, M.-J. (2008b). A Multiagent-System Framework for Hierarchical Control and Monitoring of Complex Process Control Systems. In T. Bui, T. Ho & Q. Ha (Eds.), *Intelligent Agents and Multi-Agent Systems* (Vol. 5357, pp. 381-388): Springer Berlin Heidelberg.

Tang, W. J., & Wu, Q. H. (2009). Biologically inspired optimization: a review. *Transactions of the Institute of Measurement & Control, 31*(6), 495-515.

Texas Instruments. (2013). A Powerful System-On-Chip for 2.4-GHz IEEE 802.15.4-2006 and ZigBee Applications Retrieved October 1, 2013, from http://www.ti.com/product/cc2538

Tipsuwan, Y., & Chow, M.-Y. (2003). Control methodologies in networked control systems. *Control Engineering Practice, 11*(10), 1099-1111. doi: 10.1016/s0967-0661(03)00036-4

Tryllian. (2013). Agent Dev Kit. Retrieved October 15, 2013, from http://www.tryllian.com/adk.html

Tyan, C. Y., & Wang, P. P. (1995). A Complete Tuning Procedure and Root Locus Stability Evaluation for Fuzzy Logic Control Systems. *Fuzzy Logic Research Laboratory, Department of Electrical Engineering, Duke University*.

Vaishnav, S., & Khan, Z. (2007). *Design and performance of PID and fuzzy logic controller with smaller rule set for higher order system.* Paper presented at the Proceedings of the World Congress on Engineering and Computer Science, San Francisco, USA.

Victor, J., & Dourado, A. (1997). *Adaptive scaling factors algorithm for the fuzzy logic controller.* Paper presented at the Fuzzy Systems, 1997., Proceedings of the Sixth IEEE International Conference on.

Visioli, A. (2006). *Practical PID control*. London: Springer.

Wang, L.-X. (1994). *Adaptive fuzzy systems and control*. Englewood Cliffs, N.J: PTR Prentice Hall.

Wang, T. M., Liao, I.-J., Liao, J.-C., Suen, T.-W., & Lee, W.-T. (2009). An Intelligent Fuzzy Controller for Air-Condition with Zigbee Sensors. *INTERNATIONAL JOURNAL ON SMART SENSING AND INTELLIGENT SYSTEMS, 2*(NO. 4), 636 – 652.

Woo, Z.-W., Chung, H.-Y., & Lin, J.-J. (2000). A PID type fuzzy controller with self-tuning scaling factors. *Fuzzy Sets and Systems, 115*(2), 321-326.

Yu, Y., Huang, Y., & Zeng, T. (2005). *The intelligent fuzzy controller with dynamic integral term.* Paper presented at the Industrial Technology, 2005. ICIT 2005. IEEE International Conference on.

Yunfeng, X., Yan, Z., & Rui, M. (2011, 28-30 Oct. 2011). *Ethernet-based network measure and control cystems cmbedded MCU.* Paper presented at the 2011 Fourth International Symposium on Computational Intelligence and Design (ISCID).

Yung-Yaw, C., & Chen-Cheng, Y. (1992, 11th-13th November 1992). *PD-type vs. PID-type fuzzy controllers.* Paper presented at the IEEE Region 10 Conference.

Yung-Yaw, C., & Chiy-Ferng, P. (1994, 26-29 Jun 1994). *Input scaling factors in fuzzy control systems.* Paper presented at the Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the Third IEEE Conference on.

Yuqing, L., & Huajing, F. (2005, 26-29 June 2005). *Control methodologies of large delays in networked control systems.* Paper presented at the International Conference on Control and Automation.

Zacharie, M. (2010). Adaptive fuzzy knowledge based controller for autonomous robot motion control. *Journal of Computer Science, 6*(10), 1019-1026.

Zadeh, L. A. (1965). Fuzzy sets. *Information and control, 8*(3), 338-353.

Zadeh, L. A. (1996). A rationale for fuzzy control. In J. K. George & Y. Bo (Eds.), *Fuzzy sets, fuzzy logic, and fuzzy systems* (pp. 123-126): World Scientific Publishing Co., Inc.

Zilouchian, A., & Jamshidi, M. (2001). *Intelligent control systems using soft computing methodologies*: CRC Press, Inc.