A

**PROJECT REPORT**

On

**PARTICLE SWARM OPTIMIZATION BASED ADAPTIVE PID**

**CONTROLLER**

***Submitted by-***

* ***Abhishek Tripathi (1805232903****)*
* ***Gaurav Srivastava (1705232022)***
* ***Surya Prakash Tiwari (1805232911)***

Branch: - EIE, 7th Semester

****

**Department of Electronics Engineering**

**Institute of Engineering and Technology**

**Lucknow, 226021**

***Head of Department-***

*Dr. Neelam Srivastava*

*Dept. of Electronics Engg., IET Lucknow*

***Guided by-***

*Asst. Prof. Shraddhanand Verma*

*Dept. of Electronics Engg.,*

*IET Lucknow*

***Project Incharge-***

*Dr. RCS Chauhan*

*Dept. of Electronics Engg.,*

*IET Lucknow*

**Acknowledgements**

It is my great privilege to express a deep sense of gratitude and indebtedness to **Assistant Prof. Shraddhanand Verma,** Electronics Engineering (ECE) Department, Institute of Engineering & Technology, Lucknowfor providing me a very supportive and invaluable guidance in every phase of the program and for giving all possible co-operation throughout the work.

I would like to thank to **Dr. Neelam Srivastava** **(HOD, ECE)** and all other staff members of Electronics & Instrumentation Engineering for their cooperation.

Finally, I deeply express gratefulness to my parents, brother and all my friends for withstanding me and extending their continuous inspiration and support throughout this work.

**CONTENTS**

**CHAPTER-1: INTRODUCTION OF PID CONTROLLER AND SCOPE OF THE THESIS 1-25**

1.1 Introduction 1

1.2 Close-loop control with three-term controller 1

1.3 Conventional PID controller 2

1.4 PID tuning methods 5

1.4.1 Step response method 1.4.2 Ultimate cycle method 1.4.3 Method based on performance criteria

1.5 A Review on design and tuning of PID controllers 8

1.6 Scope of the thesis 16

References 19

**CHAPTER-2: THE ADAPTIVE-PID CONTROLLER 26-43**

2.1 Introduction 26

2.2 The adaptive-PID 26

2.3 Design of the adaptive-PID 27

2.4 Tuning strategy 29

2.5 Results 31

2.5.1 Second order linear process

2.6 Conclusion 42

References 43

**CHAPTER-3: DESIGN OF PARTICLE SWARM OPTIMIZATION BASED ADAPTIVE PID CONTROLLER 80-110**

3.1 Introduction 80

3.2 Particle swarm optimization 80

3.3 Objective function of particle swarm optimization 82

3.4 Initial settings of particle swarm optimization 82

3.5 Different operations used in particle swarm optimization 83

3.6 Particle swarm optimization algorithm 86

3.7 Result 88

3.7.1 Second order linear process

3.8 Conclusion 109

References 110

**CHAPTER-4: MATLAB CODES**

4.1 Ultimate gain and ultimate time period calculation for the process model

4.2 MATLAB code for CPID of TF

4.3 MATLAB code for APID of TF

4.4 PSO based PID controller, PSO-CPID of TF

**CHAPTER-5: CONCLUSION AND FUTURE SCOPE 111-113**

4.1 Conclusion 111

4.2 Future scope 113

**CHAPTER-1**

**INTRODUCTION OF PID CONTROLLER AND SCOPE OF THE THESIS**

**1.1 Introduction**

For industrial processes, Proportional, Integral, and Derivative (PID) controllers are mostly used by process engineers [1]. In spite of considerable advances in process control over the past half century, till today PID controllers / regulators are the backbone for the most industrial control systems [2]. Even if more sophisticated control techniques are developed, it is a common practice to have a hierarchical structure with PID control at the lowest level [3, 4]. According to a survey for process control systems in refinery, chemical, and paper industries, more than 95% of the control loops are found to be of PID type [8]. With its three-term functionality covering control of both transient and steady-state responses, the PID controller offers the simplest and yet most efficient solution for many real-world control problems [5, 6]. At present, the developments of PID controllers are mostly software based, so as to get the best out of PID control [5]. A number of software-based techniques have also been realized in hardware modules, while search still goes on to find the next key technology for PID tuning [7].

**1.2 Close-loop Control with Three-term Controller**

PID controllers are quite adequate for many control problems where there are modest performance requirements. Controllers used in the process industries are mainly concerned in maintaining the process variables-level, flow temperature, pressure, pH etc. at the desired operating value [8]. As these processes become large and / or more complex, the role of controller becomes more crucial [9]. In a close-loop feedback control, PID controller provides distinctive features for its three terms (P, I, and D). Depending on the instantaneous process error, necessary action is taken by the P term, it provides an overall control action proportional to the error signal similar to all pass filter. I term has the ability to eliminate steady- state–error (offset) and it behaves like a low pass filter, whereas anticipatory corrective measure is taken by D term and its behaviour is identical to a high pass filter.

**1.3 Conventional PID Controller (CPID)**

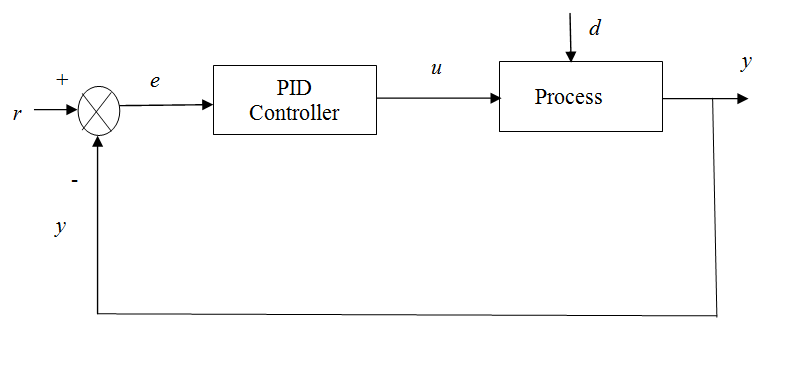
****

Fig. 1.1 Block diagram of a close loop control with PID controller

Standard nomenclature for different symbols used in Fig. 1.1

r =set point (the desired value of a controlled variable is referred to as its set point)

y=process output (controlled variables)

e=error signal

u=controller output

d=disturbance

Here, our objective is to make the controlled variable *y* equal to its set point *r* [10, 11]*.*

Block diagram of the discrete form of a conventional PID controller is shown in Fig. 1.2. The three tuneable gain parameters – proportional gain (*Kp*), integral gain (*Ki*), and derivative gain (*Kd*) play the key role in achieving the desired control performance.

Output of the controller can be expressed as:

or

Where is the proportional gain, is the integral gain, is the derivative gainis the integral time, is the derivative time, and is the sampling time period. Proper selection of the three tuning parameters – and is a critical task to attain the desired close-loop performance.

Through decades, various methods have been developed for the tuning PID parameters. Among them Ziegler-Nichols (ZN) continuous cycling method is most widely used by practicing engineers for the initial settings of PID parameters.

and = 0 then controller turns out to be a proportional controller only. In proportional mode, controller fails to bring the process output to its desired value *r*, which results in an offset.

The introduction of integral action facilitates the achievement of equality between measured value and desired value, as a constant error produces an increasing controller output until the error becomes zero.

On the other hand, the introduction of derivative action facilitates that any change in the process value can be anticipated, and thus an appropriate correction may be added prior to the actual change. So the PID controller takes the corrective measure depending on the present, past, and future status of the error signal.

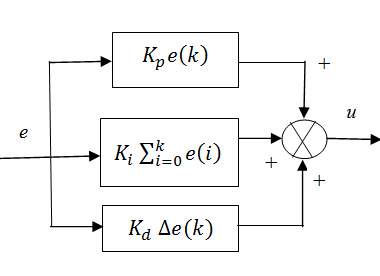
****

Fig. 1.2 (a) Parallel form of PID controller

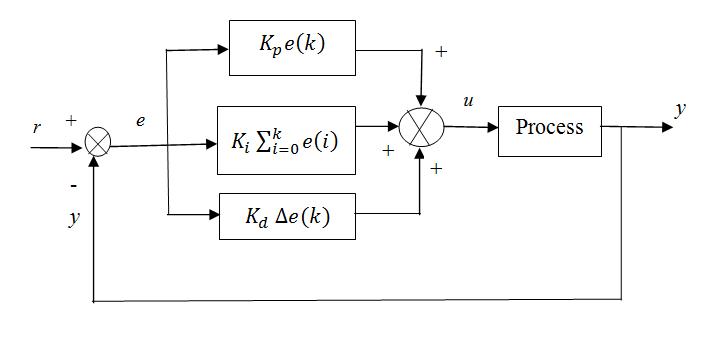


Fig. 1.2 (b) Parallel form of PID controller and process

**1.4 PID Tuning Methods**

Till today more than two hundred PID tuning rules have been proposed by the researchers [10], but none of them is suitable for all possible applications. Almost every tuning rule has some special feature for a specific class of processes, hence before selecting the tuning rule we must know the nature of the process where it is to be applied.

An extensive list of tuning rules from the mid of twentieth century to the beginning of the twenty first century is given in [10]. Depending on the nature of these tuning relations they may be broadly classified into different categories, some of them are briefly described below.

**1.4.1 Step Response / Process Reaction Curve Method**

Here, the tuning rule is based on the step response of a process, and is also known as the process reaction curve method. The process reaction curve is basically the reaction of the process to a step change in its input signal. This famous tuning was first proposed by Ziegler and Nichols in 1942 [6]. Once this method became well-known and better understood, a number of tuning rules have been proposed by different researchers [10]. In this open-loop method, the process is considered to be an integral plus time delay (IPD) model. The same technique can also be used for first-order plus time delay (FOPTD) model. For such processes Ziegler-Nichols setting would give roughly quarter-amplitude damping response. The potential problem for this tuning technique is found for processes with small dead-time or time delay to time constant ratio, since this causes the proportional gain very large and integral time low, thus making an oscillatory behaviour [9].

In 1950, Hazebroek and Van der Waerden [12] proposed a list of settings for PI controller for FOPTD processes with a range of dead-time or time delay to time constant ratio. Based on this step response method, Chien *et al*. [13] proposed regulator and servo tuning formulae for both PI and PID controllers in 1952. Their method provides two sets of tuning rules, one for 0% overshoot and other for 20% overshoot.

The main limitation for the step response based tuning techniques described above is that, only for the linear process models we can use it.

**1.4.2 Ultimate Cycle Method**

The ultimate cycle tuning method is based on the knowledge of the point where Nyquist curve intersects the negative real axis. At this point the process reaches at the verge of instability and hence a sustained oscillation is obtained in the process output. This method is called the ultimate cycle method due to the continuous cycling in process response. The first ultimate cycle based tuning method is proposed by Ziegler and Nichols in 1942 [6]. Originally it is designed to provide good responses to load disturbances. The setting for PI and PID controllers are obtained from extensive simulation study for a wide class of systems with a design criterion of quarter-amplitude damping. Later in 1967, McAvoy and Johnson [15] suggested another PID setting based on the ultimate cycle method. For restricting overshoot within 20%, a PID setting rule is suggested by Atkinson and Davey in 1968 [16].Depending on the decay rate, setting for a PI controller is developed by Hwang and Chang in 1987 [17]. The decay rate is calculated from the close-loop response with the controller having the proportional mode only.

The main drawback of ultimate cycle based tuning method [11] is that the process is to be operated in continuous cycling situation and it may cause any type of mechanical failure of the actuator parts. But the most important feature of this tuning technique is its model free approach, *i*.*e*., prior to the tuning no model is required for the process for which the controller is to be tuned. Procedure of ultimate cycle method is given below:

**Step-1**

After the process has reached steady state (at least approximately), eliminate the integral and derivative action by setting to the largest possible value.

**Step -2**

Set equal to a small value (e.g., 0.5).

**Step -3**

Introduce a small, momentary set-point change so that the controlled variable moves away from the set point. Gradually increase in small increments until continuous cycle occurs. The term continuous cycling refers to a sustained oscillation with constant amplitude. The numerical value of that produces continuous cycling (for proportional-only control) is called the ultimate gain. The period of corresponding sustained oscillation is referred to as *ultimate period,.*

**Step-4**

Calculate the PID controller settings using Ziegler-Nichols (Z-N) tuning relations in Table 1

Table 1 Controller setting based on the continuous cycling method

|  |  |  |  |
| --- | --- | --- | --- |
| Ziegler-Nichols (Z-N) |  |  |  |
| P |  | *-* | *-* |
| PI |  |  | *-* |
| PID |  |  |  |

**1.4.3 Method Based on Performance Criteria**

It is based on minimizing an appropriate performance criterion, either for optimum regulatory or for optimum servo performance. Based on the minimum *IAE* value, settings for PI and PID controllers are derived [14]. These settings are expected to provide desirable performance for time delay to time constant ratio from 0.1 to 1. Other tuning relations for PI controller for achieving minimum *IAE* value are suggested by Shinskey [8], Marlin [18], Edgar [19] etc. In 1993, Zhuang and Atherton suggested PI and PID settings based on the minimization of *ISE*, *ISTE*, and *IST*2*E* [20]. For the FOPTD process model, repeated optimization is carried out for different values of time delay to time constant ratio. Using least square fit technique, simple relations for PID parameters are obtained from graphical results. For a given range of gain margin and phase margin values, the setting for a PID controller is given by Ho [21] with *ISE* minimization.

The *ISE* criterion penalizes large errors, while the *ITAE* criterion penalizes error that persists for longer periods of time. In general, the *ITAE* criterion is the preferred criterion in practice, because it usually results in the most conservative controller settings [11]. By contrast, the *ISE* criterion provides the most aggressive settings, while the *IAE* criterion tends to produce controller settings that are between those for the *ITAE* and *ISE* criteria.

Based on *ITAE* minimization, different settings for PI and PID controllers are provided by Murrill [14], Wang [22] and many other researchers. PI and PID tuning relations based on *ITAE* performance index are also developed for the FOPTD process model by Smith *et al*. [23].

In all the above tuning rules, the optimal controller settings are different for set-point changes in comparison to those for step load disturbances. In general, the controller settings for set-point changes are more conservative. Next, we provide a brief review on PID design and tuning methods proposed by various researchers.

**1.5 A Review on Design and Tuning of PID Controllers**

In today’s competitive market scenario, process personnel are continuously striving to get more quality products with increased productivity at a reduced operating cost. All these criteria can be fulfilled from a truly effective control. The more process intelligence the designer can put into a controller, the greater is its chance of success. So a detailed understanding of the process dynamics further facilitates the formulation of appropriate tuning rules. It also indicates that successful design is not necessarily contingent on hard-core mathematics. So the performance improvement of the PID controller through more appropriate tuning relations continues to be an open challenge. This exploration has led to the incorporation of some advanced tuning algorithms into PID relations. It is found that many PID variants have been developed [2]. The inclusion of system identification and intelligent techniques in software based PID implementation helps to automate the entire design and tuning process to a useful degree. It has also led to the development of *plug and play* PID controllers, which can be easily set up and operate optimally for enhanced productivity, improved quality, and reduced maintenance requirements. Some of the software based popular robust optimal tuning methods are *PIDeasy*, *DeltaV* *Tune*, *CtrlLAB* etc [5].

Without having a detailed knowledge about the process to be controlled, it is really difficult to propose a good tuning scheme for it. So obtaining an error free model of the process is an important task. We know that for any process, no single model exists, since a model only approximates the process behaviour. In practice, it is found that as we strive for more accurate model, mathematical complexity increases. So the choice of an acceptable process model depends on its final application, as an accurate model is more complex and difficult to work with. In addition, due to natural phenomena like aging, scaling, and dirt build-up etc. process dynamics also get modified with time. To handle such varying and difficult control situations researchers are continuously trying to add different auto-tuning and gain-adaptive strategies along with the conventional tuning methods to make them more effective. Over the last few years there has been a strong resurgence in the interest towards the PID control with new capabilities to offer better solutions for modern challenges [25]. The search for more appropriate and intelligent tuning to get best out of PID is on.

The main limitation of the conventional tuning rules is that they cannot simultaneously provide good load regulation along with decent set-point tracking. It is found that, ultimate cycle-based Ziegler-Nichols relation provides a very good load regulation but its performance under set-point change is not acceptable in many cases due to excessive oscillations associated with large overshoot [6]. On the other hand, Tyreus and Luyben [26] setting shows a very low overshoot during set-point response but its load response is not equally good. Moreover, almost all the tuning rules are designed for first-order linear processes. So in case of high-order and nonlinear processes, they usually fail to provide satisfactory performances. To overcome these limitations of PI and PID controllers, various auto-tuning schemes are proposed.

A conventional PI controller fails to provide satisfactory transient performance under varying operating conditions. So in case of motor speed control applications, due to nonlinear dynamics of the electro-mechanical drive systems and as the operating point varies over a wide range, it becomes difficult for a conventional fixed gain PI controller to provide desired transient response. Theoretically it is possible to design several linear PI controllers for different operating points over the entire operating range and switch from one controller to another as the operating point changes. It is difficult to achieve manually and hence is carried out in an automated manner by using gain scheduling scheme. A gain scheduling scheme of adaptive control is proposed in [27] to continuously update the proportional and integral gains depending on the instantaneous error signal to improve both the transient and steady state performance. In [28, 29], a good review on the adaptive control strategies along with their experimental applications to process control problems are discussed. An online tuning of PID controllers for SISO systems is proposed in [24]. This technique comprises of two adaptive loops. The first loop monitors and tunes the controllers online to ensure that the system is robustly stable. When modelling errors occur, the second adaptive loop carries out recursive online identification and retunes the controller accordingly. Thus, the proposed scheme enables the system to achieve robust stability over a wide range of possible modelling errors without trading off performance against nominal design.

A real-time gain tuning method for PI controller is proposed and implemented on a Permanent Magnet Synchronous Motor (PMSM) drive [30]. The proposed method is completely model free. Here, the elements of PI controller are obtained from a performance index, which is estimated through modified binary search algorithm and the controller parameters are renewed according to the estimated intermediate gain parameter. The proposed method can assure the convergence of the gain-tuning algorithm in all cases, and it is also independent of the variations in system parameters such as load torque, reference speed, and sampling-time.

An online gain modification scheme is proposed for PI controller in [31]. This simple but effective gain updating technique continuously adjusts both the proportional and integral gains depending on the instantaneous error (*e*) and change of error (Δ*e*) of the controlled variable. The tuning strategy in this online gain adjustment scheme attempts to imitate the performance of a skilled process operator *i*.*e*., when the process variable is moving towards the desired set-point, control action is made weak to lower the overshoot; on the other hand when it moves away from the set-point control action is made aggressive for quick recovery. This technique is tested on second-order and third-order linear and nonlinear processes with a considerable improvement in process responses during both set-point change and load variation. To make this scheme more generalized, similar auto-tuning technique is also applied for PID controller [32], where the proportional, integral, and derivative gains are continuously adjusted depending on the instantaneous error (*e*) and change of error (Δ*e*) of the controlled variable. This tuning strategy is quite useful as it doesn’t require any idea regarding the process model and can be easily implemented in any conventional control loop.

Integrating processes with dead-time (IPDT) are frequently encountered in chemical plants. Many chemical processes like heating boilers, batch chemical reactors, liquid storage tanks etc. can be well approximated by such process model. A number of PI and PID tuning schemes [33-40] based on IMC technique and Direct Synthesis method are proposed for such processes. IMC technique is not straight forward and they may give poor results unless the close-loop time constant is properly selected. For similar type of IPDT process, a self-tuning PID controller is suggested in [41], which provides simultaneous improvements during set-point change and load variation. In [42], PID parameters are selected based on matching the coefficients of corresponding powers of *s* in the numerator and that in the denominator of the close-loop transfer function for a servo problem. For purely integrating system, it is found that the obtained controller has a PD form which is also verified by Visioli [43] through minimizing the *ISE* value using GA (Genetic Algorithm) based optimization.A similar method of PID tuning [44] for first-order plus dead-time (FOPTD) processes is presented with comparable performance to that of Ziegler-Nichols tuned PID controller.

To achieve improved robustness and better transient response, back-stepping adaptive PID control is proposed in [45], which leads to an adaptive PD controller for linear second-order minimal phase processes. In [46], the design of the PID controller is based on optimization of load disturbance rejection with constraints of robustness to model uncertainties. It describes a new design method for PID controllers, where the primary goal is to obtain good load disturbance responses based on the minimization of integral error. Here robustness is guaranteed by limiting the maximum sensitivity to be less than a specified value and the measurement noise is dealt with by using low-pass filter. Good set-point response is obtained by using a structure with two-degrees-of-freedom. Based on the theory of adaptive interaction, an adaptive tuning for PID controller is proposed in [47], where tuning is done through minimization of the squared error. In this method, the control system is divided into four subsystems; namely the plant, proportional control, integral control, and derivative control. An adaptive interaction algorithm is used to tune the parameters.

A model-based PI-PD controller is proposed in [48], which enables a tighter control for integrating processes. At first, relay feedback method is used to obtain the model of the integrating system with or without time delay. This method is found to exhibit better results for the set-point response maintaining satisfactory load disturbance response. In [49], a detailed comparison is provided among four different IMC design techniques for second-order plus dead-time (SOPDT) systems. The method of compensating dead-time in the IMC type of design is important especially for systems with large value of ratio between dead-time to time constant. The regions of applicability of suitable tuning rules are highlighted. It is shown that IMC designed with the Maclaurin series expansion type PID is a better choice for both set-point and load changes for systems with dead-time to time constant ratio greater than unity.

A new plant identification technique called amplitude dependent gain (ADG) is presented in [50], which gives better results than the well-known relay feedback method. In [51], this amplitude dependent gain technique is used for identification of a SOPDT model and then a PID controller is designed based on *ITAE* optimization. A new algorithm for self tuning of PID controller is proposed in [52]. This algorithm uses a combined least squares estimation and Newton-Raphson search technique to determine the ultimate gain and period of an unknown systems for tuning of PID controller based on Ziegler-Nichols (ZN) and refined Ziegler-Nichols (RZN) relations. It does not require external process perturbations. Moreover, this tuning method is very fast and can be used to automatically tune PID controllers. For ensuring good stability robustness of PI / PID controller, numerical optimization of the shortest distance from the Nyquist curve of the open-loop transfer function to the critical point is proposed in [53].

In [54], a new criterion for robust stability is developed, which makes use of both the gain and phase information of the open-loop system. Based on it, a robust PID design is presented for a SOPDT uncertain model. For a noisy environment, derivative filter is an integral part of PID design but there is no straight forward method for selecting the derivative filter constant. Almost all the tuning rules specify the values for proportional, integral, and derivative gains but not the filter constant. A systematic method for obtaining the derivative filter constant along with the tuning parameters for both PI and PID controllers is given in [55]. An IMC based PID controller design is proposed in [56] for low-order unstable plus dead-time processes to get the desired close-loop response. This approach illustrates IMC-PID parameter synthesis using Laurent series; it has an advantage over the other approximating power series as it generalizes MacLaurin series. Specifically, it is applicable for solving singularity problems, *i*.*e*., to synthesize PID parameters for IPDT type of systems. The important features of the proposed method are that the controller becomes faster and the required order of the filter is reduced to get an IMC-PID structure.

Astrom and Hagglund [57] developed the attractive relay auto-tuning technique for identifying the ultimate gain and ultimate period of oscillation for an unknown process. This technique has some distinct advantages compared to the Ziegler-Nichols continuous cycling method [6] –

(i) only a single experiment to be carried out instead of trial and error; (ii) amplitude of the process output can be restricted by adjusting the relay amplitude; (iii) the process is not forced to the stability limit; and (iv) experimental test can be easily automated using commercial products.

Furthermore, unlike other auto-tuning methods this technique eliminates the need for a careful choice of the sampling rate from the prior knowledge of the process. The PID relay auto-tuner of Astrom and Hagglund [57] is one of the simplest and most robust auto-tuning techniques for controlling processes and has been successfully applied in the industry for more than fifteen years. This tuner is based on an approximate estimation of the critical point on the process frequency response from relay oscillations. Many developments [58-64] have been reported to extend its applications. It turns out that more and accurate information on process dynamics can be obtained from the same relay test with the help of new identification techniques to tune PID controllers in a better way. Extensions are also made to tune model-based advanced controllers and multivariable controllers. A good review on relay feedback auto-tuning is given in [58]. Many research works on modifying the relay feedback auto-tuning method are reported later on. Improvements on the relay identification accuracy and efficiency are proposed in [59-63] by reducing high-order harmonic terms or using the Fourier analysis instead of the describing function method. Relay feedback auto-tuning for processes with varying time delay is proposed by Leva [64].

Of late, soft-computing tools - like fuzzy logic, neural network, and genetic algorithmare being used increasingly in designing PID controllers for performance improvement and increased robustness [65-74]. Online tuning scheme with fuzzy *If-Then* rules to update the parameters of a conventional PID controller is suggested in [65]. In [66, 67], the overall gain of the fuzzy PI and PD controllers are continuously adjusted using fuzzy *If-Then* rules defined on the current process states. The overall gain of the fuzzy logic controller (FLC) for both PI and PD implementations are adjusted by online variation in output scaling factor (SF) of the FLC. The most important feature of this proposed method is that the basic control policy of an experienced process operator is realized through a fuzzy rule and the technique is completely process model independent. Similar types of fuzzy rules are also used to parameterize a PID controller with a dead-beat control format [68]. Lee [69] proposed a PI-type fuzzy controller with resetting action to avoid large accumulation of controller output, responsible for the excessive oscillation in set-point response. The computation of the resetting factor is done by a fuzzy rule base. Here, in one method resetting action depends on the error and change of error, and for other method it depends on the error and controller output.

To obtain satisfactory performance a smart and easy structure of a self-tuning PI type fuzzy controller is proposed in [70]. Only seven tuning rules are used to tune the input-output scaling factors for the proposed fuzzy controller. Another PID type fuzzy controller with self-tuning scaling factors is given in [71]. Kulic *et al*. [72] developed a neural network-based gain scheduled PI speed controller to achieve superior performance compared to conventional controllers. A neural fuzzy inference network (NFIN) is used in [73] for temperature control of a water bath. The NFIN is a fuzzy rule-based network possessing neural network’s learning ability. The important feature of NFIN is that no pre-assignment and design of the rules are required. The rules (structure and parameter) are constructed automatically during the on-line operation. A hybrid fuzzy controller with genetic optimization of controller performance index *ITAE* is proposed in [74]. In the first stage, performance improvement is achieved by tuning the parameter of the output membership functions and later genetic algorithm-based optimization technique is applied so that *ITAE* reaches its minimum.

To minimize the excessive overshoot in the set-point response different techniques are adapted, such as - set-point weighting, gain detuning, set-point filtering etc. It is found that the set-point weighting is superior to the other techniques for lowering the overshoot [76]. In case of gain detuning, the process response is retarded during both set-point change and load disturbance. But in case of set-point weighting and set-point filtering the load disturbance response is not affected. However, on comparing the performance of set-point weighting and set-point filtering it is found that the set-point weighting is superior to set-point filtering because the speed of response is sacrificed to a large degree for the same reduction in the process overshoot. Hang *et al*. [76] used fixed set-point weighting for Ziegler-Nichols tuned PI controller where the weighting factor takes value in between zero and unity. For further improvement of transient response, the idea of variable set-point weighting is introduced in [77]. When the system output is far away from the desired value, a large value of weighting is given to speed up the process response to lower the rise time, on the other hand when the output is near to the desired value, a smaller value of the weighting factor is introduced. So, depending on the process operating conditions the set-point weighting factor switches through three different values. Here, the basic control strategy is similar to manual control. Similar control scheme is also implemented by Visioli [78] using fuzzy inference engine. In this method the weighting factor is obtained from a fuzzy rule-base depending on the values of input fuzzy variables - error and change of error.

Set-point weighting techniques for large dead-time and unstable processes are suggested in [79, 80]. The presence of a zero in the process transfer function model introduces overshoot in the close-loop response. If the zero is positive, then there will be an initial undershoot. To reduce the overshoot and initial jump, the set-point weighted PID controller is always preferred. Due to mathematical complexity, it is difficult to find out the set-point weighting factor based on overshoot or *ISE* minimization for time delayed second-order unstable system with a positive or negative zero. Leva and Colombo [75] proposed an *ISE* based optimization technique for finding out the set-point weighting parameter. This technique can be safely plugged into an existing auto-tuner. Similar optimization-based set-point weighting techniques are suggested in [81, 82]. Chidambaram [81] proposed an analytical method for calculation of optimal weight factor for both stable and unstable FOPTD systems with time delay. This method is based on minimizing the overshoot for a servo problem. By introduction of the weighting factor in the control algorithm the location of zero is moved farther to the left side, as a result there will be a reduction in overshoot. In [82], the idea of [81] has been extended where the weighting parameter is obtained by minimizing the overshoot and *ISE* valuefor the servo problem.

Authors in [83] extended the idea of separate set-point weighting for the proportional and derivative terms. The scheme is developed for unstable FOPTD systems. Rao *et* *al*. [84] proposed a set-point weighted modified Smith predictor for integrating and double integrating processes with time delay. Here, direct synthesis method is applied to satisfy both the set-point tracking and good load rejection behaviour respectively. In [85] set-point weighting parameters are calculated from empirical relations based on the ratio of time delay to dominant time constant for unstable first-order time delay systems. For the similar class of systems, set-point weighting parameters are calculated by using pole placement method [86]. For controlling a hydraulic crane boom system similar to a human operator, a PI type fuzzy controller with set-point weighting is reported in [87]. The idea of implementing the operator’s control strategy is also adopted in dynamic set-point weighting [88, 89] without the help of fuzzy rules. Here, the weighting factor is continuously updated depending on the current process operating states to achieve improved performances during both the set-point change and load variation.

**1.6 Scope of the Thesis**

Our literature survey reveals that a lot of works has been done towards improving the performance of PID controllers with increased robustness. In a broad sense, such development works on controller tuning are mostly dependent on the process model. However, for a practical process it is very difficult to find its exact model, as a result, most of the theoretical developments have limitation from practical implementation point of view. Along with mathematical complexity in finding out the appropriate process model, there is always a certain amount of uncertainty in model parameters. Model parameters are also changing with time due to natural phenomena like aging, scaling, erosion etc. So, obtaining the desired performance from a PID controller is not only goal. Additionally, it has to be robust enough to withstand the model uncertainties as well as process nonlinearities. At the same time, it is found that an optimally tuned controller is more prone to fragile [90]. So depending on the area of application, there should be a compromise between optimality and robustness of selected parameters.

Soft-computing [65, 66] tools like fuzzy logics, neural networks, genetic algorithms, and particle swarm techniques are also used by the researchers to obtain optimal settings of PID parameters. In such cases the engineers have tried to incorporate the human intelligence in the controller behaviour. Certain improvements are found in the controller performances on making them more intelligent but at the higher computational complexity. A controller designed to reduce the initial overshoot during set-point change usually fails to offer load rejection behaviour. On the other hand, a controller with better load capability cannot restrict the overshoot in the set-point response. In [31, 32], improvements in the process behaviour are observed during both set-point and load disturbance responses. These works [31, 32] motivated us for the present study, where we attempt to make the developed PID controller in [32] an optimal one using two computational intelligence tools, Genetic Algorithms and Particle Swarm Optimization, with respect to some integral performance criteria. Next, we provide a brief description about the organization of the rest part of the thesis.

**In Chapter-2,** we present the adaptive PID controller (APID) developed in [32] to have a better understanding on the various implementation issues. Classical and well adapted Ziegler-Nichols tuned PID controllers usually provide excessively large overshoots, not tolerable in most of the situations, for high-order and nonlinear processes. In order to overcome such limitations, a number of attempts have been made in [31, 32, 88, 89] for online adjustments of various gain parameters of CPID, thereby making it an adaptive PID (APID), so that an overall improved performance is achieved.

The objective behind such online gain adjustments is that, when the process is moving towards the set point, control action will be less aggressive to avoid possible large overshoots and/or undershoots, and when the process is moving away from the set point, control action will be more aggressive to make a rapid convergence of the system. Following this gain adaptive technique, in [32] a significantly improved performance of APID is found for high-order and nonlinear systems both in set-point and load disturbance responses.

However, out of the *seven* parameters of [32], *i.e.*, ,, and , the first *three* constants, *i.e.*, , and are selected based on ZN ultimate cycle rule, where as the remaining *four* constants, *i.e.*, and are chosen by trial. Therefore, there is further scope to achieve more improved performance if we can find the most appropriate settings of these parameters.

**In chapter-3**, we have presented detail description of particle swarm-based optimization of PID parameters of [32]. We have applied particle swarm based optimal PID controller on different process models*.* Simulation results of the developedParticle swarm optimization (PSO) based controller is compared with other PID controllers.

**In Chapter-4,** We have developed the MATLAB Code for CPID, APID, PSO based PID &

PSO-CPID controller . We have also calculated the ultimate gain and ultimate time period for

the process model using MATLAB codes. Detailed descriptions are given along with figures.

**In Chapter-5**, we have provided implementation issues in the optimization technique, particle swarm optimization while designing optimal PID controllers in Chapters 3 and 4. Lastly, we also try to point out future scopes for further improvement.

**References**

[1] K. J. Astrom and T. Hagglund, Automatic tuning of PID controllers, Instrument Society of America, Research Triangle Park, 1988.

[2] K. J. Astrom and T. Hagglund, Revisiting the Ziegler-Nichols step response method for PID control, Journal of Process Control, Vol. 14, No. 6, pp. 635-650, 2004.

[3] L. Desbourough and R. Miller, Increasing customer value of industrial control performance monitoring - Honeywell’s experience, Proc. Sixth International Conference on Chemical Process Control, AIChE Symposium Series Number 326, Vol. 98, 2002.

[4] K. J. Astrom and T. Hagglund, The future of PID control, Control Engineering Practice, Vol. 9, No. 11, pp. 1163-1175, 2001.

[5] K. H. Ang, G. Chong, and Y. Li, PID control system analysis, design, and technology, IEEE Transaction on Control System Technology, Vol. 13, No. 4, pp. 559-576, 2005.

[6] J. G. Ziegler and N. B. Nichols, Optimum setting for automatic controllers, ASME Transaction, Vol. 64, No. 11, pp. 759-768, 1942.

[7] P. Marsh, Turn on, tune in, New Electronics, Vol. 31, No. 4, pp. 31-32, 1998.

[8] F. G. Shinsky, Process Control Systems - Application, Design and Tuning, McGraw-Hill, New York, 1998.

[9] B. W. Bequette, Process Control Modeling, Design, and Simulation, Pearson Education, New Jersey, 2003.

[10] A. O’Dwyer, Hand book of PI and PID Controller Tuning Rules, Imperial College Press, London, 2003.

[11] D. E. Seborg, T. F. Edgar, and D. A. Mellichamp, Process Dynamics and Control, John Wiley & Sons, Singapore, 2004.

[12] P. Hazebroek and B. L. Van der Waerden, The optimal adjustment of regulators, ASME Transaction, Vol. 72, No. 3, pp. 317-322, 1950.

[13] K. L. Chien, J. A. Hrones, and J. B. Reswick, On the automatic control of generalized passive systems, ASME Transaction, Vol. 74, No. 2, pp. 175-185, 1952.

[14] P. W. Murrill, Automatic control of processes, International Textbook Company, Pennsylvania, USA, 1967.

[15] T. J. McAvoy and E. F. Johnson, Quality of Control Problems for Dead-Time Plants, Industrial and Engineering Chemistry Process Design and Development, Vol. 6, No. 4, pp. 440-446, 1967.

[16] P. Atkinson and R. L. Davey, A theoretical approach to the tuning of pneumatic three term controllers, Control, pp. 238-242, 1968.

[17] S. H. Hwang and H. C. Chang, A theoretical examination of close-loop properties and tuning methods of single loop PI controllers, Chemical Engineering Science, Vol. 42, No. 10, pp. 2395-2415, 1987.

[18] T. E. Marlin, Process Control, McGraw-Hill, New York, 1995.

[19] T. F. Edgar *et al*., Perry's Chemical Engineers’ Handbook, McGraw-Hill, New York, 1997.

[20] M. Zhuang and D. P. Atherton, Automatic tuning of optimum PID controllers, IEE Proc. - Control Theory Applications, Vol.140, No. 3, pp. 216-224, 1993.

[21] W. K. Ho, K. W. Lim, and W. Xu, Optimal gain and phase margin tuning for PID controllers, Automatica, Vol. 34, No. 8, pp. 1009-1014, 1998.

[22] F. S. Wang, W. S. Juang, and C. T. Chan, Optimal tuning of PID controllers for single and cascade control loops, Chemical Engineering Communications, Vol. 132, No. 1, pp. 15-34, 1995.

[23] C. A. Smith and A. B. Corripio, Principles and Practice of Automatic Control, John Wiley, New York, 1997.

[24] H. P. Huang, M. L. Roan, and J. C. Jeng, On-line adaptive tuning for PID controllers, IEE Proc. - Control Theory Applications, Vol. 149, No. 1, pp. 60-67, 2002.

[25] K. J. Astrom, P. Albertos, and J. Quevedo, Preface: PID Control, Control Engineering Practice, Vol. 9, No. 11, pp. 1159-1161, 2001.

[26] B. D. Tyreus and W. L. Luyben, Tuning PI controllers for integrator / dead-time process, Industrial & Engineering Chemistry Research, Vol. 31, pp. 2625-2628, 1992.

[27] S. K. Panda, J. M. S. Lim, P. K. Dash, and K. S. Lock, Gain scheduled PI speed controller for PMSM drive, Proc. IEEE industrial electronic society international conference - IECON-1997, Vol. 2, pp. 925-930, 1997.

[28] D. E. Seborg and T. F. Edgar, Adaptive control strategies for process control: A survey, AIChE Journal, Vol. 32, No. 6, pp. 881-913, 1986.

[29] K. J. Astrom and B. Wittenmark, A Survey of Adaptive Control Applications, Proc. IEEE 34th International Conference on Decision & Control, pp. 649-654, 1995.

[30] M. Tursini, F. Parasiliti, and D. Zhang, Real time gain tuning of PI controllers for high performance PMSM drives, IEEE Transaction on Industry Applications, Vol. 38, No. 4, pp. 1018-1026, 2002.

[31] R. K. Mudi, C. Dey, and T. T. Lee, An improved auto-tuning scheme for PI controllers, ISA Transaction, Vol. 47, No. 1, pp. 45-52, 2008.

[32] C. Dey and R. K. Mudi, An improved auto-tuning scheme for PID controllers, ISA Transaction, Vol. 48, No. 4, pp. 396-409, 2009.

[33] I. K. Kookos, A. I. Lygros, and K. G. Arvanitis, On-line PI controller tuning for integrator / dead-time processes, European Journal of Control, Vol. 5, pp.19-31, 1999.

[34] E. Poulin, and A. Pomerleau, PI settings for integrating processes based on ultimate cycle information, IEEE Transaction on Control System Technology, Vol. 7, No. 4, pp. 509-511, 1999.

[35] W. Zhang, Xu. Y, and Y. Sun, Quantitative performance design for integrating processes with time delay, Automatica, Vol. 35, No. 4, pp. 719-723, 1999.

[36] Y. G. Wang, and W. J. Cai, Advanced proportional-integral-derivative tuning for integrating and unstable process with gain and phase margin specifications, Industrial & Engineering Chemistry Research, Vol. 41, No. 6, pp. 2910-2914, 2002.

[37] D. Chen, and D. E. Seborg, PI / PID controller design based on direct synthesis and disturbance rejection, Industrial & Engineering Chemistry Research, Vol. 41, No. 19, pp. 4807-4822, 2002.

[38] S. Skogestad, Simple analytic rules for model reduction and PID controller tuning, Journal of Process Control, Vol. 13, No. 4, pp. 291-309, 2003.

[39] I. Kaya, Two-degree-of-freedom IMC structure and controller design for integrating processes based on gain and phase-margin specifications, IEE Proc. - Control Theory Applications, Vol. 151, No. 4, pp. 401-407, 2004.

[40] T. Liu, X. He, D. Y. Gu, and W. D., Zhang, Analytical decoupling control design for dynamic plants with time delay and double integrators, IEE Proc. - Control Theory Applications, Vol. 152, No. 2, pp. 238-246, 2005.

[41] C. Dey and R. K. Mudi, A simple auto-tuning PID controller for integrating plus dead-time processes, Control and Intelligent Systems.

[42] M. Chidambaram and R. Padma Sree, A simple method of tuning PID controllers for integrator / dead-time processes, Computers and Chemical Engineering, Vol. 27, No. 2, pp. 211-215, 2003.

[43] A. Visioli, Optimal tuning of PID controllers for integral and unstable processes, IEE Proc. - Control Theory Applications, Vol. 148, No. 2, pp. 180-184, 2001.

[44] R. Padma Sree, M. N. Srinivas, and M. Chidamabaram, A simple method of tuning PID controllers for stable and unstable FOPTD systems, Computers and Chemical Engineering, Vol. 28, No. 11, pp. 2201-2218, 2004.

[45] A. R. Benaskeur and A. Desbiens, Backstepping-based adaptive PID control, IEE Proc. - Control Theory Applications, Vol. 149, No. 1, pp. 54-59, 2002.

[46] H. Panagopoulos, K. J. Astrom, and T. Hagglund, Design of PID controllers based on constrained optimization, IEE Proc. - Control Theory Applications, Vol. 149, No. 1, pp. 32-40, 2002.

[47] F. Lin, R. D. Brandt, and G. Saikalis, Self-tuning of PID controllers by adaptive interaction, Proc. American Control Conference, pp. 3676-3681, 2000.

[48] I. Kaya and D. P. Atherton, A PI-PD controller design for integrating processes, Proc. American Control Conference, pp. 258-262, 1999.

[49] R. C. Panda, C. C. Yu, and H. P. Huang, PID tuning rules for SOPDT systems: Review and some new results, ISA Transaction, Vol. 43, No. 2, pp. 283-295, 2004.

[50] R. R. Pecharroman and F. L. Pagola, Improved identification for PID controllers auto-tuning, Proc. European Control Conference -ECC-1999, in CD, Germany, 1999.

[51] F. L. Pagola and R. R. Pecharroman, PID auto-tuning based on a second point of frequency response, Proc. IFAC world congress 2002, in CD, Spain, 2002.

[52] A. B. Rad, W. L. Lo, and K. M. Tang, Self-tuning PID Controller Using Newton-Raphson Search Method, IEEE Transaction on Industrial Electronics, Vol. 44, No. 5, pp. 717-725, 1997.

[53] R. Toscano, A simple robust PI / PID controller design via numerical optimization approach, Journal of Process Control, Vol. 15, No. 1, pp. 81-88, 2005.

[54] Q. G. Wang, C. C. Hang, Y. Yang, and J. B. He, Quantitative robust stability analysis and PID controller design, IEE Proc. - Control Theory Applications, Vol. 149, No. 1, pp. 3-7, 2002.

[55] A. J. Isaksson and S. F. Graebe, Derivative filter is an integral part of PID design, IEE Proc. - Control Theory Applications, Vol. 149, No.1, pp. 41-45, 2002.

[56] R. C. Panda, Synthesis of PID controller for unstable and integrating processes, Chemical Engineering Science, Vol. 64, No. 12, pp. 2807-2816, 2009.

[57] K. J. Astrom and T. Hagglund, Automatic tuning of simple regulators with specifications on phase and amplitude margins, Automatica, Vol. 20, No. 5, pp. 645-651, 1984.

[58] C. C. Hang, K. J. Astrom, and Q. G. Wang, Relay feedback auto-tuning process controllers - a tutorial review, Journal of Process Control, Vol. 12, No. 1, pp. 143-162, 2002.

[59] T. H. Lee, Q. G. Wang, and K. K. Tan, A modified relay-based technique for improved critical point estimation in process control, IEEE Transaction on Control Systems Technology, Vol. 3, No. 3, pp. 330-337, 1995.

[60] S. W. Sung, J. H. Park, and I. Lee, Modified relay feedback method, Industrial & Engineering Chemistry Research, Vol. 34, No. 11, pp. 4133-4135, 1995.

[61] S. Shen, H. Yu, and C. Yu, Use of saturation-relay feedback for autotune identification, Chemical Engineering Science, Vol. 51, No. 8, pp. 1187-1198, 1996.

[62] Q. G. Wang, C. C. Hang, and B. Zou, Low-order modelling from relay feedback, Industrial and Engineering Chemistry Research, Vol. 36, No. 2, pp. 375-381, 1997.

[63] Q. G. Wang, C. C. Hang, and B. Zou, A frequency response approach to auto-tuning of multivariable controllers, Chemical Engineering Research and Design, Vol. 75, No. 8, pp. 797-806, 1997.

[64] A. Leva, PID auto-tuning algorithm based on relay feedback, IEE Proc.- D, Vol. 140, No. 5, pp. 328-338, 1993.

[65] A. Visioli, Tuning of PID controllers with fuzzy logic, IEE Proc.- Control Theory Applications, Vol. 148, No. 1, pp. 1-8, 2001.

[66] R. K. Mudi and N. R. Pal, A robust self-tuning scheme for PI and PD type fuzzy controllers, IEEE Transaction on Fuzzy System, Vol. 7, No. 1, pp. 2-16, 1999.

[67] R. K. Mudi and N. R. Pal, A self-tuning fuzzy PI controller, Fuzzy Sets and System, Vol. 115, No. 2, pp. 327-338, 2000.

[68] R. Bandyopadhyay and D. Patranabis, A new autotuning algorithm for PID controllers using dead-beat format, ISA Transaction, Vol. 40, No. 3, pp. 255-266, 2001.

[69] J. Lee, On methods for improving performance of PI-type fuzzy logic controllers, IEEE Transaction on Fuzzy System, Vol. 1, No. 4, pp. 298-301, 1993.

[70] H. Y. Chung, B. C. Chen, and J. J. Lin, A PI-type fuzzy controller with self-tuning scaling factors, Fuzzy Sets and System, Vol. 93, No. 1, pp. 23-28, 1998.

[71] Z. W. Woo, H. Y. Chung, and J. J. Lin, A PID type fuzzy controller with self-tuning scaling factors, Fuzzy Sets and System, Vol. 115, No. 2, pp. 321-326, 2000.

[72] F. Kulic, D. Kukolj, and E. Levi, Artificial Neural Network as a Gain Scheduler for PI Speed Controller in DC Motor Drives, Proc. Neural Network Applications in Electrical Engineering NEUREL-2000, pp. 199-203, 2000.

[73] C. T. Lin, C. F. Juang, and C. P. Li, Water bath temperature control with a neural fuzzy inference network, Fuzzy Sets and System, Vol. 111, No. 2, pp. 285-306, 2000.

[74] S. K. Oh, D. K. Lee, W. Pedrycz, and D. H. Kim, The genetic design of hybrid fuzzy controllers, Cybernetics and Systems, Vol. 35, No. 4, pp. 333-361,2004

[75] A. Leva and A. M. Colombo, Method for optimizing set-point weights in ISA-PID auto tuners, IEE Proc. - Control Theory Applications, Vol. 146, No. 2, pp.137-146, 1999.

[76] C. C. Hang, K. J. Astrom, and W. K. Ho, Refinements of Ziegler-Nichols tuning formula, IEE Proc. - D, Vol. 138, No. 2, pp.111-118, 1991.

[77 ] C. C. Hang and L. Cao, Improvement of Transient Response by Means of Variable Set-point Weighting, IEEE Transaction on Industrial Electronics, Vol. 43, No. 4, pp. 477-484, 1996.

[78] A. Visioli, Fuzzy logic-based set-point weighting for PID controllers, IEEE Transaction on System Man Cybernetics - A, Vol. 29, No. 6, pp. 587-592, 1999.

[79] B. Z. Khan and B. Lehman, Set-point PI controllers for systems with large normalized dead-time, IEEE Transaction on Control System Technology, Vol. 4, No. 4, 1996.

[80] R. Padma Sree and M. Chidambaram, Simple method of calculating set-point weighting parameter for unstable systems with a zero, Computers and Chemical Engineering, Vol. 28, No. 11, pp. 2433-2437, 2004.

[81] M. Chidambaram, Set-point weighted PI / PID controllers, Chemical Engineering Communications, Vol. 179, No. 1, pp. 1-13, 2000.

[82] R. Padma Sree and M. Chidambaram, Set-point weighted PID controllers for unstable systems, Chemical Engineering Communications, Vol. 192, No. 1, pp. 1-13, 2005.

[83] G. Prashanti and M. Chidambaram, Set-point weighted PID controllers for unstable systems, Journal of the Franklin Institute, Vol. 337, No. 2-3, pp. 201-215, 2000.

[84] A. S. Rao, V. S. R. Rao, and M. Chidambaram, Set-point weighted modified Smith predictor for integrating and double integrating processes with time delay, ISA Transaction, Vol. 46, No. 1, pp. 59-71, 2007.

[85] A. M. DePaor and M. O’Malley, Controllers of Ziegler-Nichols type for unstable process with time delay, International Journal of Control, Vol. 49, No. 4, pp. 1273-1284, 1989.

[86] V. Clement and M. Chidambaram, PID control of unstable time delay systems, Chemical Engineering Communications, Vol. 162, No. 1, pp. 63-74, 1997.

[87] Y. Yang, A. Luo, and K. E. Rydberg, PI control based on fuzzy set-point weighting tracking for hydraulic crane boom system, Journal of Control Theory and Applications, Vol. 4, No. 4, pp. 327-330, 2006.

[88] C. Dey and R. K. Mudi, Dynamic set-point weighting based PI controllers, IEEE Transaction on Control System Technology (submitted).

[89] C. Dey, R. K. Mudi, and T. T. Lee, Dynamic set-point weighted PID controller, Control and Intelligent Systems, Vol. 48, No. 1, pp. 1-8, 2010.

[90] L. H. Keel and S. P. Bhattacharyya, Robust, fragile, or optimal?, IEEE Transaction on Automatic Control, Vol. 42, No. 3, pp. 1098-1105, 1997.

[91] D.E. Goldberg, Genetic algorithm in search optimization and machine learning. Addison-Wesley Publishing Co., Inc.(1989).

[92] Randy L. Haupt, and Sue Ellen Haupt , Practical Genetic Algorithms, Wiley publishing Co. Second edition(2004) .

[93] James McCaffrey, <http://msdn.microsoft.com/en-us/magazine/hh335067.aspx>

**CHAPTER-2**

**THE ADAPTIVE PID CONTROLLER (APID) [5]**

**2.1 Introduction**

Conventional PID (CPID) are found to perform quite satisfactory for first-order process, but they usually fail to provide acceptable performance for high-order and non-linear process due to large over shoots and poor load regulation [1-6]. To overcome these drawbacks Dey and Mudi [5] developed an adaptive PID known as APID.

While running a plant in manual mode, an operator generally adjusts the controller gains according to the current process trend to attain the desired response. The basic idea behind such gain manipulation strategy is that when process variable is moving away from the set-point, controller takes aggressive action to bring it back to the desired value as soon as possible. On the other hand, when the process is moving fast towards the set-point, control action is reduced to restrict the potential overshoot and undershoot in subsequent operating phases. In the proposed APID [5] authors try to realize the above gain modification strategy with the help of some simple heuristic rules incorporating an online gain updating factor α, defined on the normalized e and ∆e. Here, proportional integral and derivative gains of APID are adjusted towards improving the process response during set-point change as well as load disturbance. Since, our proposed work is based on this APID [5] so we will study this in detail in the next section.

**2.2 The Adaptive-PID (APID)**

The simplified block diagram of the proposed APID [5] is shown in Fig. 2.1.

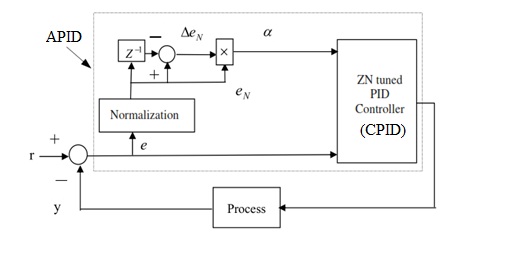


Fig. 2.1.Block diagram of the APID [5]

It shows that the gain updating factor α, a function of process error (e) and change of error (∆e) continuously adjusts the parameter of a CPID. Fig. 2.1 indicates that the starting point of the APID for a given process is its corresponding CPID, which means initial settings of the proposed PID auto-tuner are based on ZN rules.

Each of such ZN tuned parameters of APID (i.e., proportional, integral and derivative gains) is updated online by the single modifying factor α through some simple relations.

**2.3 Design of the Adaptive-PID**

Let the discrete form of conventional PID is described as

(2.1)

In equation (2.1), is the control action at kth sampling instant, is the proportional gain, is the integral gain, is the derivative gainis the integral time, is the derivative time, and is the sampling time interval. are calculated according to ZN ultimate cycle tuning rules (i.e.,=, where are the ultimate gain and ultimate period respectively).

Here e(k) and ∆e(k) are expressed as

e(k)=r−y(k) (2.2)

∆e(k)=e(k ) − e(k-1) , (2.3)

Where r is the set point, and y(k) is the process output. The proposed gain updating factor α is defined by

, (2.4)

, (2.5)

. (2.6)

Variables in eq.(2.5) and eq.(2.6) are the normalised value of e(k) and ∆e(k) respectively. From eq. (2.4), without loss if generality it may be assumed that the possible variation of α will lie in the range [−1 ,1] for all close-loop stable processes.

In APID and will be continuously modified by the gain updating factor α with the following simple heuristic relations:

(2.7)

(2.8)

(2.9)

Thus, from eqs. (2.1) and (2.7)-(2.9) , APID can be expressed as

, (2.10)

Where (*k*), and are the modified proportional, integral and derivative gains respectively at *k*th instant, and is the corresponding control action. In eqs.(2.7)-(2.9), are three positive constants, which will make the required variations in (*k*), , and around their respective initial values.

The objective of proposed auto-tuning scheme is that, subsequent to any set-point change or load disturbance, the three parameters of APID (i.e.,) will be continuously adjusted by the nonlinear updating factor α in order to have a quick recovery of the process during both the set-point change and load variation without a large number of oscillations.

Such real time nonlinear gain variations are introduced towards achieving an enhanced control performance. Eqs. (2.7)-(2.9) indicates that compared to CPID, in APID both proportional and derivatives gains i.e., are increased throughout the entire operating cycle, though may not be in same proportion (due to difference in the value of), while the integral gain () is either increased or decreased from its initial setting depending on operating phases.

**2.4 Tuning Strategy** [5]

While designing APID, the following major points are taken into consideration to provide the appropriate control action in different operating phases. For a better understanding, typical close-loop response of an under-damped second-order process and its corresponding variation of α are illustrated in Fig.2.2.

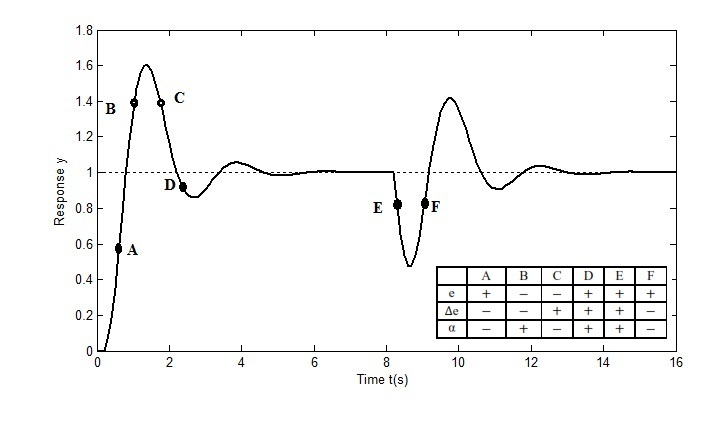


Fig. 2.2 close loop response of second order process

1. While the process is far from the set-point and moving fast towards it (e.g., points A, C, or F in Fig. 2.2), proportional gain should be reasonably large to reach the set-point quickly but the integral gain should be small enough to prevent the large accumulation of control action, which may result in a large overshoot or undershoot in future. At the same time, to reduce oscillations derivative gain should be increased for higher damping. Observe that, in such transient phase e and ∆e are of opposite signs. Therefore, α becomes negative according to eq.(2.4) which will make both the proportional and derivatives gain higher, and integral gain lower than their corresponding initial values (i.e., ) as indicated by eqs.(2.7)-(2.9).Thus, the gain adaptive rules (eqs.(2.7)-(2.9)) try to adjust the parameters of APID towards reducing the overshoot and/or undershoot, and oscillation in the process response.
2. When the process is moving further away from the set-point (e.g., points B, D, or E in Fig.2.2), increased proportional, and derivative as well as integral gains are expected to bring back the process variable to its desired value quickly. Under such situations both e and ∆e will have the same sign, thereby making α positive (eq.(2.4)),which in turn makes all gain parameters of APID (i.e.,) larger than their respective initial values according to eqs.(2.7)-(2.9). As a result the control action becomes more aggressive (i.e., ) which will try to restrict further deterioration of such situations. Therefore, APID satisfies the need for a relatively strong control action to improve process recovery.

From the above discussion it is evident that the proposed auto-tuning scheme always attempt to modify APID parameters (proportional, integral, and, integral gains) in the right directions to generate required control action in different transient phases for providing improved performance under both set-point change and load disturbance. Of course, depending on the type of response desired to achieve, suitable value of are to be selected by the designer either from the knowledge about the process to be controlled or through trial and error. Next, we provide extensive simulation study with APID.

**2.5 Results**

To have a better understanding of the performance of APID, here we study the following systems with dead-time (*L*):

, L=0.2s, and 0.3s; (2.11)

For detailed comparison, in addition to the close loop response characteristics, several performance indices, such as percentage overshoot (%*OS*), rise time (*tr*), settling time (*ts*), integral absolute error (*IAE*) and integral time absolute error (*ITAE*) is calculated for APID and compared with CPID. The value of tuning parameters are chosen by trial and it is for all the processes discussed in this chapter.

**2.5.1 Second order linear process**

Transfer function of the process is given by

**** (2.11)

Response of second order liner process in (2.11) with L=0.2s, and L=0.3s under CPID, and APID is shown in Fig. 2.3. Performance indices of the process in (2.11) for CPID and APID controller are given in the Table 2.1. Though the controller are tuned for L= 0.2s, a higher value i.e., L= 0.3s is also tested without changing controllers settings (for both controller CPID and APID). In case of APID overshoot (%*OS*) is reduced from 60.30% to 5.37% and also from 93.93% to 15.58% for L=0.3s. Performance analysis reveals that unlike CPID, APID is capable of providing acceptable and remarkably improved performance during both set point change and load disturbance.

**Table 2.1 Performance analysis of second order process **

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **L** | **Controller** | **%OS** | **tr(s)** | **ts(s)** | **IAE** | **ITAE** |
| 0.2s | **CPID** | 60.30 | 0.9 | 4.4 | 2.08 | 9.29 |
| **APID** | 5.37 | 1.4 | 4.2 | 1.36 | 6.26 |
| 0.3s | **CPID** | 93.93 | 0.9 | 12.2 | 4.40 | 35.28 |
| **APID** | 15.58 | 1.0 | 5.7 | 1.85 | 14.10 |



**Fig. 2.3(a) Response of second order linear process in (2.11) for L=0.2s**



**Fig 2.3(b) Response of second order linear process in (2.11) for L=0.3s**

**2.6 Conclusion**

We have made a detailed study on the performance of APID [5] for a wide range of processes under both set-point and load disturbance. The effectiveness of APID has been tested under a significant perturbation in the process parameter (dead-time). APID has shown improved performance compared to CPID for both set point change and load disturbance.

Observe that, here the unknown parameters are and other is 0.3 for gain adjusting (let it is *k4*). Thus, there is a further scope to improve the close-loop performance by selecting more appropriate values of. So in the next two Chapters 3 and 4, we will use genetic algorithms and particle swarm optimization for achieving the optimal value of.

**References**

[1] K. J. Astrom, C.C. Hang, P.Person, and W.K.Ho, Towards intelligent PID control,Automatica, Vol. 28 ,No. 1, pp.1-9,1922.

[2] C. Dey, R.K. Mudi, and D. Simhachalam, An Auto-tuning PID Controller for Integrating Plus Dead-time Processes. Advanced Materials Research. 403-408, 4934-4943(2012).

[3] C. Dey, R.K. Mudi, and D.Simhachalam, A Simple Nonlinear PD Controller for Integrating Processes. ISA Trans. 53(1), 162-172(2014).

[4]D. E. Seborg , and T.F. Edgar, Adaptive control strategies for process control: A survey. AICHE J. 32(6), 881–913(1986).

[5] C. Dey , and R.K. Mudi, An improved auto-tuning scheme for PID controllers. ISA Trans. 48(4), 396–409(2009).

[6] R. K. Mudi, C. Dey, and T.T. Lee, An improved auto-tuning scheme for PI controllers. ISA Transactions. 47, 45-52(2008).

**CHAPTER-3**

**DESIGN OF A PARTICLE SWARM OPTIMIZATION BASED ADAPTIVE PID CONTROLLER**

**3.1 Introduction**

Inthis chapter we will vary all the seven parameters, i.e.,,, and of APID [1] within a defined range. Particle swarm optimization (PSO) is used to find out best set of solution which will optimize the given objective function. In 1995, Edward and Kennedy [2-4] first introduced the PSO method motivated by social behaviour of organisms such as fish schooling and bird flocking. It is also a population-based search technique. PSO can be easily implemented and usually results in faster convergence rates than other techniques. Unlike the GA, PSO has no evolution operators such as crossover and mutation.

**3.2 Particle Swarm Optimization (PSO)**

Particle swarm optimization [2-4] is an artificial intelligence (AI) technique which can be is used to find approximate solutions to extremely difficult or impossible numeric maximization and minimization problems. The idea behind the algorithm was inspired by social behavior of animals such as bird flocking and fish schooling. This theory can be understood by the concept of techniques used by birds or fishes for searching the food in wide area. Suppose the following scenario, a group of birds or fishes are randomly searching for food in a wide area. There is only one piece of food in the area being searched. All the birds/fishes do not know the exact location where the food is. In that condition, they travel in search space according to the own experience as well as neighbour’s experience. That mean, in each iteration, they compare the distance between its own location and the target with respect to its previous experience as well as the best position of neighbor which is closest to the target. After that they modify its own speed for the best strategy to find the food. This is the basic principle of Particle Swarm Optimization (PSO).

In technical term, each bird or fish is called “Particle” and its flock is called “Particle Population”. All the particles have own fitness or objective value which is calculated by the objective function. For the optimization of objective function, particles positions are updated by velocity vector which depends on its personal influence as well as social influence.

Technically in other words PSO starts with random set of solutions that is called the particles. Each particle has positions (value of the variables) and velocities. The particles update their velocities and positions based on the local best solution (best solution associated with current population) and global best solution (best solution associated with population found so far).

To avoid confusion, I am giving some nomenclature which is used in PSO.

|  |  |
| --- | --- |
| **Term** | **Explanation** |
| Particle | One set of solution, i.e., one set of value of variables. |
| Position | Value of individual variable. |
| velocity | Corresponding to each variable there is a velocity. It is a vector quantity. |
| Population | It is the no of particle in a swarm, i.e., set of solutions |
| fitness | f (particle) |
| Variables | Define all the independent variables |
| Range of variables | Define the range of all independent variables |
| Velocity range | [ -|(span of the variable)| , |(span of the variable)| ]  Where span is the difference between largest and smallest value of that variable. |
| Local \_best \_particle | It is the best solution associated with minimum fitness of the current population. |
| Global \_best \_particle | It is the best solution associated with minimum fitness of the population found so far. |

3.3 Objective Function of Particle Swarm Optimization

The function to be optimized is known as objective function. Here, minimization of the *integral-absolute-error* (*IAE*) or *integral-time-absolute-error* (*ITAE*) or combination of both i.e., (*IAE+ITAE*) is defined as the objective function (performance index or fitness function). The *IAE* and *ITAE* are calculated as:

**3.4 Initial Settings of Particle Swarm Optimization -** It is given in the Table 4.1

**Table 3.1**

|  |  |
| --- | --- |
| Population | 10 |
| Variables | ,, and |
| Range of variables | , are ±20% of their respective CPID,  nd. |
| Velocity range | [ -|(span of the variable)| , |(span of the variable)|]  Where span is the difference between largest and smallest value of that variable. |

3.5 Different Operations Used in Particle Swarm Optimization

Population

Here, the population size is 10. We have 7 variables . We generate random set of solution by MATLAB-command random (population size, variables), i.e., random (10, 7). All the values are between 0 to 1 that mean (0, 1) and matrix size is 10\*7.

Bring the particle’s positions (variables) in range

Since value of the variables is not in the defined range therefore we bring it in a defined range to get the real value of the optimization variable. The following linear mapping is used for this purpose.

Let i denotes the particles so i=1, 2, 3...10

And let j denotes the variables so j=1, 2, 3…7

(4.3)

Where, particle(i,j)=real value of variable

maximum value of jth variable of any set of solutions

minimum value of jth variable of any set of solutions

Span of decimal no = 0.9999 - 0.0001

Velocity and velocity range

Each variable has velocity which is a vector quantity. To generate velocity the same MATLAB command is used as we have used to generate population, i.e., random (10, 7). Now we bring the decimal value of velocity in a defined range to get the real value of velocity. The following linear mapping is used for this purpose.

(4.4)

Where, velocity (i,j) is the real value of velocity.

Velocity range is defined as:

=| is the maximum velocity of j th variable of any set of solution

| is the minimum velocity of j th variable of any set of solution

Other unknowns are same as previous case.

Particle’s velocity update, particle’s position update and inertia weight (w)

We update particle’s velocity first then we update particle’s position by the following method.

Update particle’s velocity by

New\_velocity(i,j) = w\*velocity(i,j) + c1\*r1\*(Local\_best\_particle(i,j)-particle(i,j) ) + c2\*r2\*(Global\_best\_particle(i,j) - particle(i,j) )

(4.5)

Update particle’s position by

particle(i,j) = particle(i,j) + new\_velocity(i,j) (4.6)

Where i=1, 2, 3…10 is the particles and j=1, 2, 3…7 is the variables. i and j is used to indicate the coordinate of the particle. Here w is the inertia weight. It is used to control the search. At the starting phase of search inertia weight is almost equal to 0.99 and search is in exploration mode. At the end phase of search inertia weight is very low (almost equal to 0.01) and search is in exploitation mode. Thus, at the starting phase of search change in particle’s velocity and position are very high compared to end phase of search where change in particle’s velocity and position are very small.

Note- if the exploration mode is very high (w >1) then you jump from one solution to another solution with much gap (span) because of the high velocity of the particle. You may over jump best solution. And if exploitation mode is very high (w 0.01) then program will take too much time to converge. Because w affects our results drastically therefore you cannot give importance to anyone of the modes. I have made w as a time varying quantity (dynamic nature) to control the both modes, i.e., w gradually decreases from 0.99 to 0.01 as the no of iteration increases.

Local weight and global weight

c1 and c2 are constant. c1 is called the cognitive or personal or local weight. c2 is called the social or global weight. c1=2 and c2=2. r1 and r2 are the random number and both are in range of (0,1).

Surety that you are at global optima

From the second term in the velocity update equation (4.5), we can say that particles always try to move towards Local\_best\_particle, i.e., local minimum and from the third term we can say particles always try to move towards Global\_best\_particle, i.e., global minimum.

If the Global\_best\_particle is too far from the Local\_best\_particle, then Global\_best\_particle has huge impact on the velocities and positions of the particles which are nearer to Local\_best\_particle. It means that for those particles high change in velocities and positions occur. Therefore, particles finally move towards Global\_best\_particle. Also, the random variables r1 and r2 add a random component to the particles movement and help to prevent particles from getting stuck at a non-optimal local minimum solution.

### 3.6 Particle Swarm Optimization Algorithm

(1) Generate population with uniform random number and bring it within the define range.

(2) Generate velocity with uniform random number and bring it within the define range.

(3) For each particle, i.e., i=1, 2, 3…10

Evaluate the fitness, i.e., objective function, i.e., f(particle( i))

(4) Find out the particle associated with minimum fitness. Let for ith particle we are getting minimum fitness then

Local\_best\_particle = particle(ith)

Global\_best\_particle = particle(ith)

Start of PSO loop: repeat until slope of objective is almost zero or until maximum no of iteration is reached.

(1) For each particle, i.e., i=1, 2, 3...10

For each variable, i.e., j=1, 2, 3…7

{ Generate the random number r1, r2.

Update the velocity by

New\_velocity(i,j)=w\*velocity(i,j)+c1\*r1\*(Local\_best\_particle(i,j)-particle(i,j)) +c2\*r2\*(Global\_best\_particle(i,j) - particle(i,j))

Check the velocity limit and if it is out of range, bring it in range.

If New\_velocity(i,j) > Vmax (j) then New\_velocity(i,j) = Vmax(j)

Else if New\_velocity(i,j) < Vmin(j) then New\_velocity(i,j) = Vmin(j)

}

(2) For each particle, i.e., i=1, 2, 3...10

For each variable, i.e., j=1, 2, 3…7

{

Update the position by

particle(i,j)= particle(i,j)+new\_velocity(i,j)

Check the particle’s position limit and if it is out of range bring it in range.

If particle (i,j) > Xmax (j) then particle (i,j)=Xmax(j)

Else if particle (i,j) < Xmin (j) then particle (i,j)=Xmin(j)

}

(3) For each particle, i.e., i=1, 2, 3…10

Evaluate the fitness, i.e., objective function i.e. f (particle(i))

(4) Find out the particle associated with minimum fitness in the current population. Let for ith particle we are getting minimum fitness.

Local\_best\_particle=particle(ith)

If f (Local\_best\_particle) < f (Global\_best\_particle )

Global\_best\_particle= Local\_best\_particle

End

Hence Global\_best\_particle is our solution for which fitness is minimum.

**3.7 Results**

For simulation study, we consider the following systems with dead-time (*L*):

, L=0.2s, and 0.3s; (4.7)

For each process model, like previous chapter, we have used four different types of controller.

(a) **CPID.**

(b) **PSO-CPID** -, are ±20% of their respective CPID and these are calculated by PSO.

(c) **APID.**

(d) **PSO-APID** - In these all seven parameters are varying within the defined range and these are calculated by PSO.

We have calculated the close loop response characteristics for above process model by using different controllers. For detailed comparison, in addition to the response characteristics, several performance indices, such as percentage overshoot (%*OS*), rise time (*tr*), settling time (*ts*), integral absolute error (*IAE*) and integral time absolute error (*ITAE*) are calculated for each controller. Performance of our PSO-APID is compared with CPID, PSO-CPID, and APID. Fourth-order Range-Kutta method is used for numeric integration. The detailed performance analysis for various types of process is discussed below.

**3.7.1 Second Order Linear Process**

Transfer function of the process is given by

**** (4.7)

Response of second order liner process in (4.7) with L=0.2s, and L=0.3s under CPID, PSO-CPID, APID, and PSO-APID is shown in Fig. 4.1. Performance indices of the process in (4.7) for different controllers are given in Table 4.2(a) and Table 4.2(b). Though the controller are tuned for L=0.2s, a higher value i.e., L=0.3s is also tested without changing controllers settings (for CPID and APID). Performance analysis reveals that unlike CPID, PSO-CPID, and APID, our PSO-APID is capable of providing acceptable and remarkably improved performance during both set point change and load disturbance.

**Table 3.2(a) -Performance analysis of second order linear process  with L=0.2s**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **L=**  **0.2s** | **Controller** | **Objective function** | **%OS** | **tr(s)** | **ts(s)** | **IAE** | **ITAE** |
|  | **CPID** | **IAE** | 60.30 | 0.9 | 4.4 | 2.08 | 9.29 |
| **PSO-CPID** | 44.49 | 0.8 | 3.8 | 1.60 | 6.65 |
| **APID** | 5.37 | 1.4 | 4.2 | 1.36 | 6.26 |
| **PSO-APID** | 2.21 | 0.9 | 1.9 | 1.12 | 4.92 |
|  | **CPID** | **ITAE** | 60.30 | 0.9 | 4.4 | 2.08 | 9.29 |
| **PSO-CPID** | 44.49 | 0.8 | 3.8 | 1.60 | 6.65 |
| **APID** | 5.37 | 1.4 | 4.2 | 1.36 | 6.26 |
| **PSO-APID** | 2.21 | 0.9 | 1.9 | 1.12 | 4.92 |
|  | **CPID** | **IAE**  **+**  **ITAE** | 60.30 | 0.9 | 4.4 | 2.08 | 9.29 |
| **PSO-CPID** | 47.31 | 0.8 | 3.7 | 1.61 | 6.73 |
| **APID** | 5.37 | 1.4 | 4.2 | 1.36 | 6.26 |
| **PSO-APID** | 0.0 | 2.3 | 1.9 | 1.37 | 4.57 |

**Table 3.2(b) -Performance analysis of second order linear process  with L=0.3s**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **L=**  **0.3s** | **Controller** | **Objective function** | **%OS** | **tr(s)** | **ts(s)** | **IAE** | **ITAE** |
|  | **CPID** | **IAE** | 93.93 | 0.9 | 12.2 | 4.41 | 35.28 |
| **PSO-CPID** | 61.81 | 1.0 | 3.5 | 2.27 | 13.98 |
| **APID** | 15.58 | 1.0 | 5.7 | 1.85 | 14.10 |
| **PSO-APID** | 2.25 | 1.3 | 3.2 | 1.61 | 11.80 |
|  | **CPID** | **ITAE** | 93.93 | 0.9 | 12.2 | 4.41 | 35.28 |
| **PSO-CPID** | 61.81 | 1.0 | 3.5 | 2.27 | 13.98 |
| **APID** | 15.58 | 1.0 | 5.7 | 1.85 | 14.10 |
| **PSO-APID** | 2.56 | 1.3 | 3.1 | 1.62 | 11.33 |
|  | **CPID** | **IAE**  **+**  **ITAE** | 93.93 | 0.9 | 12.2 | 4.41 | 35.28 |
| **PSO-CPID** | 61.81 | 1.0 | 3.5 | 2.27 | 13.98 |
| **APID** | 15.58 | 1.0 | 5.7 | 1.85 | 14.10 |
| **PSO-APID** | 1.85 | 2.7 | 2.4 | 1.63 | 11.40 |

****

**Fig 3.1(a) Response of second order linear process in (4.7) for L=0.2s, minimization of IAE**

****

**Fig 3.1(b) Response of second order linear process in (4.7) for L=0.3s, minimization of IAE**

****

**Fig 3.1(c) Response of second order linear process in (4.7) for L=0.2s, minimization of ITAE**

****

**Fig 3.1(d) Response of second order linear process in (4.7) for L=0.3s, minimization of ITAE**

****

**Fig 3.1(e) Response of second order linear process in (4.7) for L=0.2s, minimization of IAE +ITAE**



**Fig 3.1(f) Response of second order linear process in (4.7) for L=0.3s, minimization of IAE +ITAE**

**4.8 Conclusion**

PSO has also been found to be a powerful optimization technique like genetic algorithms. The developed PSO based controller, i.e., PSO-APID has provided significantly improved performance compared to others. Remarkable improvement has been observed in case of second-order nonlinear and pH-neutralization processes.

**References**

[1] C. Dey, and R.K. Mudi, An improved auto-tuning scheme for PID controllers. ISA Trans. 48(4), 396–409(2009).

[2] James McCaffrey, <http://msdn.microsoft.com/en-us/magazine/hh335067.aspx>

[3] Microsoft, Msdn magazine (august-2011), 86-91.

[4] Randy L. Haupt, and Sue Ellen Haupt, Practical Genetic Algorithms, Wiley publishing Co. Second edition(2004).

**Chapter -4**

**MATLAB CODES**

We have written the code for following process model.

* , L=0.2 and 0.3;
* , L=0.2 and 0.3;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SN. | PROCESS MODEL | DELAY | ULTIMATE GAIN (calculated at first delay ) | ULTIMATE PERIOD |
| 1- |  | L=0.2 and 0.3 | 10.5 | 2.0333 |
| 2 |  | L=0.2 and 0.3 | 5.09 | 2.9 |

**Matlab code for ultimate gain and ultimate time period calculation for the process model **

%%%% calculation of ultimate gain and time period

%%%%tf=exp(-0.2s)/(s2+2s+1)

clc;

clear all;

h=0.1;

t = 0:h:30;

y = zeros(1,length(t));

u = zeros(1,length(t));

e = zeros(1,length(t));

x = zeros(1,length(t));

i = zeros(1,length(t));

input = 1;

y(1) = 0;

x(1)=0 ;

e(1)=input - y(1);

kp=10.5;

u(1)=kp\*e(1);

F\_xy = @(x) -2\*x;

for i = 1:2

%y and u taken as time input and x as output

k\_1 = F\_xy(x(i));

k\_2 = F\_xy(x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

u(i+1)=kp\*e(i+1);

end

F\_xy = @(u,y,x) u-y-2\*x;

for i = 3:length(t)

k\_1 = F\_xy(u(i-2),y(i),x(i));

k\_2 = F\_xy(u(i-2)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-2)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-2)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

u(i+1)=kp\*e(i+1);

end

z=y(1:301)

plot(t,z)

xlabel('value of time t')

ylabel('value of y')

title(' calculation of ultimate gain,k =10.5 ultimate period=2.033sec, second order system tf=exp(-0.2s)/(s2+2s+1) ')

grid on

**Matlab code for CPID of TF1**

clc;

clear all;

h=0.1;

t = 0:h:16;

tf=16/h;

y = zeros(1,length(t));

u = zeros(1,length(t));

e = zeros(1,length(t));

x = zeros(1,length(t));

y(1) = 0;

x(1)=0 ;

r=1

e(1)=r - y(1);

ku=10.5;

tu=2.033;

kc=0.6\*ku

ti=tu/2

td=tu/8

u(1)=kc\*(e(1) +(0.1/ti)\*sum(e));

F\_xy = @(x) -2\*x;

for i = 1:2

%y and u taken as time input and x as output

k\_1 = F\_xy(x(i));

k\_2 = F\_xy(x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=r-y(i+1);

er=sum(e);

ed=e(i+1)-e(i);

u(i+1)=kc\*(e(i+1) +(0.1/ti)\*sum(e)+(td/0.1)\*ed);

end

F\_xy = @(u,y,x) u-y-2\*x;

for i = 3:(tf/2)

k\_1 = F\_xy(u(i-2),y(i),x(i));

k\_2 = F\_xy(u(i-2)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-2)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-2)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=r-y(i+1);

er=sum(e);

ed=e(i+1)-e(i);

u(i+1)=kc\*(e(i+1) +(0.1/ti)\*sum(e)+(td/0.1)\*ed);

end

% effect of load on the process(25%)

u((tf/2)+1)=-14

F\_xy = @(u,y,x) u-y-2\*x;

for i = (tf/2)+1:length(t)

k\_1 = F\_xy(u(i-2),y(i),x(i));

k\_2 = F\_xy(u(i-2)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-2)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-2)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=r-y(i+1);

er=sum(e);

ed=e(i+1)-e(i);

u(i+1)=kc\*(e(i+1) +(0.1/ti)\*sum(e)+(td/0.1)\*ed);

end

z=y(1:tf+1);

plot(t,z,'--')

xlabel('Time t ')

ylabel('Response y')

title(' CPID(- - -) Response of second order system TF=exp(-0.2s)/s2+2s+1)')

grid on

**Matlab code for APID of TF1**

%%%%APID adaptive PID CONTROLLER Response of second order system TF=exp(-0.2s)/s2+2s+1

clc;

clear all;

h=0.1;

t = 0:h:16;

tf=16/h;

y = zeros(1,length(t));

u = zeros(1,length(t));

e = zeros(1,length(t));

x = zeros(1,length(t));

kpp = zeros(1,length(t));

kii = zeros(1,length(t));

kdd = zeros(1,length(t));

v = zeros(1,length(t));

x = zeros(1,length(t));

r= 1;

setpoint=r;

y(1) = 0;

x(1)=0 ;

e(1)=r - y(1);

ku=10.5;

tu=2.033;

kp=0.6\*ku

ti=tu/2

td=tu/8

ki=kp\*(0.1/ti)

kd=kp\*(td/0.1)

k1=1

k2=1

k3=12

u(1)=kp\*e(1)+ki\*sum(e);

kpp(1)=0;

kii(1)=0;

kdd(1)=0;

v(1)=0;

F\_xy = @(x) -2\*x;

for i = 1:2

k\_1 = F\_xy(x(i));

k\_2 = F\_xy(x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=r-y(i+1);

er=sum(e);

ed=e(i+1)-e(i);

v(i+1)=(e(i+1)/r)\*(ed/r);

kpp(i+1)=kp\*(1+k1\*abs(v(i+1)));

kii(i+1)=ki\*(0.3+k2\*v(i+1));

kdd(i+1)=kd\*(1+k3\*abs(v(i+1)));

u(i+1)=kpp(i+1)\*e(i+1)+ kii(i+1)\*sum(e)+ kdd(i+1)\*ed;

end

F\_xy = @(u,y,x) u-y-2\*x;

for i = 3:(tf/2)

k\_1 = F\_xy(u(i-2),y(i),x(i));

k\_2 = F\_xy(u(i-2)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-2)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-2)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=r-y(i+1);

ed=e(i+1)-e(i);

er=sum(e);

v(i+1)=(e(i+1)/r)\*(ed/r);

kpp(i+1)=kp\*(1+k1\*abs(v(i+1)));

kii(i+1)=ki\*(0.3+k2\*v(i+1));

kdd(i+1)=kd\*(1+k3\*abs(v(i+1)));

u(i+1)=kpp(i+1)\*e(i+1)+ kii(i+1)\*sum(e)+ kdd(i+1)\*ed;

end

u((tf/2)+1)= -14;

F\_xy = @(u,y,x) u-y-2\*x;

for i = (tf/2)+1:length(t)

k\_1 = F\_xy(u(i-2),y(i),x(i));

k\_2 = F\_xy(u(i-2)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-2)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-2)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=r-y(i+1);

ed=e(i+1)-e(i);

er=sum(e);

v(i+1)=(e(i+1)/r)\*(ed/r);

kpp(i+1)=kp\*(1+k1\*abs(v(i+1)));

kii(i+1)=ki\*(0.3+k2\*v(i+1));

kdd(i+1)=kd\*(1+k3\*abs(v(i+1)));

u(i+1)=kpp(i+1)\*e(i+1)+ kii(i+1)\*sum(e)+ kdd(i+1)\*ed;

end

z=y(1:(tf+1));

plot(t,z)

hold on

xlabel('Time t ')

ylabel('Response y')

title('APID Response of second order system TF=exp(-0.2s)/s2+2s+1) ')

grid on

**Matlab code for PSO based PID controller, PSO-CPID of TF1**

%%%%%%%%%%%%%%%genetic algorithm based pid controller

%%%% exp(-0.2s) /(s+1)^2

%%%first we will clculate kp ki kd from zeigler nichols method.

%By varying plus minus 20% of all the tuned values(kp ki kd)

%We will calculate best tuned valuve of kp ki kd with the help of particle swarm optimization algorithms

%%%%Transfer func= TF = exp(-0.2s)/(s+1)^2

%%%%%%%znpid with pso-algorithm

clc ;

clear all;

popsize=10;

npar=3;

c1=2;

c2=2;

no\_of\_variable=3;

par=rand(popsize,npar);

vel=rand(popsize,npar);

%%%%%%%%%%%%%%%%%%initialization %%%%%%

ku=10.5;

tu=2.0333;

kp=0.6\*ku;

ti=tu/2;

td=tu/8;

ki=kp\*(0.1/ti);

kd=kp\*(td/0.1);

%%%%%%%%%%%%%%%%%5

xh1=kp+0.20\*kp;

xl1=kp-0.20\*kp;

xh2=ki+0.20\*ki;

xl2=ki-0.20\*ki;

xh3=kd+0.20\*kd;

xl3=kd-0.20\*kd;

h=0.1;

t = 0:h:16;

tf=16/h

delay=0.2;

de=2;

w1=1;

w2=1;

%%%% bring position vector in range

for i=1:10

x1(i,1)=xl1+((xh1-xl1)/(0.9999-0.0001))\*par(i,1);

x2(i,1)=xl2+((xh2-xl2)/(0.9999-0.0001))\*par(i,2);

x3(i,1)=xl3+((xh3-xl3)/(0.9999-0.0001))\*par(i,3);

end

p=[x1 x2 x3 ] ;

%%%define the range of velocity

vh1=(xh1-xl1);

vl1= -(xh1-xl1);

vh2=(xh2-xl2);

vl2= -(xh2-xl2);

vh3=(xh3-xl3);

vl3= -(xh3-xl3);

%%% Bring velocity vector in range

for i=1:10

v1(i,1)=vl1+((vh1-vl1)/(0.9999-0.0001))\*vel(i,1);

v2(i,1)=vl2+((vh2-vl2)/(0.9999-0.0001))\*vel(i,2);

v3(i,1)=vl3+((vh3-vl3)/(0.9999-0.0001))\*vel(i,3);

end

v=[v1 v2 v3 ] ;

%%%%%%%objective fun..

for j=1:10

kp=p(j,1);

ki=p(j,2);

kd=p(j,3);

y = zeros(1,length(t)+100);

x = zeros(1,length(t)+100);

u = zeros(1,length(t)+100);

e = zeros(1,length(t)+100);

er = zeros(1,length(t)+100);

ed = zeros(1,length(t)+100);

g = zeros(1,length(t)+100);

input = 1;

y(1) = 0;

x(1)=0 ;

e(1)=input - y(1);

u(1)=kp\*e(1)+ki\*sum(e);

F\_xy = @(x) -2\*x;

for i = 1:de

k\_1 = F\_xy(x(i));

k\_2 = F\_xy(x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

ed=e(i+1)-e(i);

u(i+1)=kp\*e(i+1)+ki\*sum(e)+kd\*ed;

end

F\_xy = @(u,y,x) u-y-2\*x;

for i = (de+1):(tf/2)

k\_1 = F\_xy(u(i-de),y(i),x(i));

k\_2 = F\_xy(u(i-de)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-de)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-de)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

ed=e(i+1)-e(i);

u(i+1)=kp\*e(i+1)+ki\*sum(e)+kd\*ed;

end

u((tf/2)+1)=-20;

F\_xy = @(u,y,x) u-y-2\*x;

for i = (tf/2)+1:length(t)

k\_1 = F\_xy(u(i-de),y(i),x(i));

k\_2 = F\_xy(u(i-de)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-de)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-de)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

ed=e(i+1)-e(i);

u(i+1)=kp\*e(i+1)+ki\*sum(e)+kd\*ed;

end

z=y(1:161);

figure(j+1)

plot(t,z)

xlabel('Time t ')

ylabel('Response y')

title(' exp(-0.2s) /(s+1)^2 ')

grid on

iaeiae(j,1)=0.1\*sum(abs(e));

h=0.1;

for i= 1:length(t)

g(i)=0.01\*i\*e(i);

end

itaeitae(j,1)=sum(abs(g));

end

iae=w1\*iaeiae+w2\*itaeitae;

[iae ind]=sort(iae);

p=p(ind,:);

v=v(ind,:);

localpar=p(1,:);

localminima(1,1)=iae(1,1);

globalpar=p(1,:);

globalminima(1,1)=iae(1,1);

display('start of while loop')

iter=0;

m=1;

n=10;

maxit=30;

while iter < maxit

iter=iter+1;

r1=rand(popsize,npar);

r2=rand(popsize,npar);

wt=(maxit-iter)/maxit;

for i=1:10

v(i,:)= wt\*v(i,:) + c1\*r1(i,:).\*(localpar-p(i,:)) + c2\*r2(i,:).\*(globalpar-p(i,:));

end

v;

v1=v(:,1);

v2=v(:,2);

v3=v(:,3);

%%In main equation of PSO there are three term of addition so sometimes it

%%may cross the boundary limit so we will do with the following method

for i=1:10

if v1(i,1) > vh1

v1(i,1)= vh1;

elseif v1(i,1) < vl1

v1(i,1)= vl1;

end

end

for i=1:10

if v2(i,1) > vh2

v2(i,1)= vh2;

elseif v2(i,1) < vl2

v2(i,1)= vl2;

end

end

for i=1:10

if v3(i,1) > vh3

v3(i,1)= vh3;

elseif v3(i,1) < vl3

v3(i,1)= vl3;

end

end

v1;

v2;

v3;

v=[v1 v2 v3];

%%%%%new position of particle will be

p=p+v;

%%% bring p in the range

x1=p(:,1);

x2=p(:,2);

x3=p(:,3);

%%%bring kp ki kd in range....it means x1 x2 x3

for i=1:10

if x1(i,1) > xh1

x1(i,1)= xh1;

elseif x1(i,1) < xl1

x1(i,1)= xl1;

end

end

for i=1:10

if x2(i,1) > xh2

x2(i,1)= xh2;

elseif x2(i,1) < xl2

x2(i,1)= xl2;

end

end

for i=1:10

if x3(i,1) > xh3

x3(i,1)= xh3;

elseif x3(i,1) < xl3

x3(i,1)= xl3;

end

end

x1;

x2;

x3;

v1;

v2;

v3;

v=[v1 v2 v3];

p=[x1 x2 x3];

p;

%%%objective function

for j=1:10

kp=p(j,1);

ki=p(j,2);

kd=p(j,3);

y = zeros(1,length(t)+100);

x = zeros(1,length(t)+100);

u = zeros(1,length(t)+100);

e = zeros(1,length(t)+100);

er = zeros(1,length(t)+100);

ed = zeros(1,length(t)+100);

g = zeros(1,length(t)+100);

input = 1;

y(1) = 0;

x(1)=0 ;

e(1)=input - y(1);

u(1)=kp\*e(1)+ki\*sum(e);

F\_xy = @(x) -2\*x;

for i = 1:de

k\_1 = F\_xy(x(i));

k\_2 = F\_xy(x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

ed=e(i+1)-e(i);

u(i+1)=kp\*e(i+1)+ki\*sum(e)+kd\*ed;

end

F\_xy = @(u,y,x) u-y-2\*x;

for i = (de+1):(tf/2)

k\_1 = F\_xy(u(i-de),y(i),x(i));

k\_2 = F\_xy(u(i-de)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-de)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-de)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

ed=e(i+1)-e(i);

u(i+1)=kp\*e(i+1)+ki\*sum(e)+kd\*ed;

end

u((tf/2)+1)=-20;

F\_xy = @(u,y,x) u-y-2\*x;

for i = (tf/2)+1:length(t)

k\_1 = F\_xy(u(i-de),y(i),x(i));

k\_2 = F\_xy(u(i-de)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-de)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-de)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

ed=e(i+1)-e(i);

u(i+1)=kp\*e(i+1)+ki\*sum(e)+kd\*ed;

end

z=y(1:161);

figure(j+1)

plot(t,z)

xlabel('Time t ')

ylabel('Response y')

title(' exp(-0.2s) /(s+1)^2 ')

grid on

iaeiae(j,1)=0.1\*sum(abs(e));

h=0.1;

for i= 1:length(t)

g(i)=0.01\*i\*e(i);

end

itaeitae(j,1)=sum(abs(g));

end

iae=w1\*iaeiae+w2\*itaeitae;

[iae ind]=sort(iae);

p=p(ind,:);

v=v(ind,:);

localpar=p(1,:);

localminima(iter+1,1)=iae(1,1);

ss=length(localminima);

tt=length(globalminima);

if (localminima(ss,1) < globalminima(tt,1))

globalpar= p(1,:);

globalminima(tt+1,1)= iae(1,1);

else

globalminima(tt+1,1)= globalminima(tt,1);

end

vv(m:n,:)=v(1:10,:);

pp(m:n,:)=p(1:10,:);

m=n+1;

n=n+10;

end

display('please note the kp ki kd -this is the result of pso algorithm')

globalpar

%%%%%%%%

for j=1

kp=globalpar(j,1);

ki=globalpar(j,2);

kd=globalpar(j,3);

y = zeros(1,length(t)+100);

x = zeros(1,length(t)+100);

u = zeros(1,length(t)+100);

e = zeros(1,length(t)+100);

er = zeros(1,length(t)+100);

ed = zeros(1,length(t)+100);

g = zeros(1,length(t)+100);

input = 1;

y(1) = 0;

x(1)=0 ;

e(1)=input - y(1);

u(1)=kp\*e(1)+ki\*sum(e);

F\_xy = @(x) -2\*x;

for i = 1:de

k\_1 = F\_xy(x(i));

k\_2 = F\_xy(x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

ed=e(i+1)-e(i);

u(i+1)=kp\*e(i+1)+ki\*sum(e)+kd\*ed;

end

F\_xy = @(u,y,x) u-y-2\*x;

for i=(de+1):(tf/2)

k\_1 = F\_xy(u(i-de),y(i),x(i));

k\_2 = F\_xy(u(i-de)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-de)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-de)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

ed=e(i+1)-e(i);

u(i+1)=kp\*e(i+1)+ki\*sum(e)+kd\*ed;

end

u((tf/2)+1)=-20;

F\_xy = @(u,y,x) u-y-2\*x;

for i = (tf/2)+1:length(t)

k\_1 = F\_xy(u(i-de),y(i),x(i));

k\_2 = F\_xy(u(i-de)+0.5\*h,y(i)+0.5\*h,x(i)+0.5\*h\*k\_1);

k\_3 = F\_xy((u(i-de)+0.5\*h),(y(i)+0.5\*h),(x(i)+0.5\*h\*k\_2));

k\_4 = F\_xy((u(i-de)+h ) ,(y(i)+h) ,(x(i)+k\_3\*h));

x(i+1) = x(i) + (1/6)\*(k\_1+2\*k\_2+2\*k\_3+k\_4)\*h;

y(i+1)=y(i)+h\*x(i+1);

e(i+1)=1-y(i+1);

ed=e(i+1)-e(i);

u(i+1)=kp\*e(i+1)+ki\*sum(e)+kd\*ed;

end

z=y(1:tf+1);

figure(1)

plot(t,z)

xlabel('Time t ')

ylabel('Response y')

title(' exp(-0.2s) /(s+1)^2 ')

grid on

end

%%%time specification

y;

yy = y(1:(tf/2));

[yymax tp]=max(yy);

peak\_time=(tp-1)\*0.1;

overshoot=(yymax-1)\*100

rr=1;

while y(rr)<1.0001

rr=rr+1;

end

rise\_time=(rr-1)\*0.1

s=(tf/2);

while y(s)>0.98 & y(s)<1.02;

s=s-1;

end

settling\_time=(s-1)\*0.1

iae=0.1\*sum(abs(e))

h=0.1;

for i= 1:length(t)

g(i)=0.01\*i\*e(i);

end

itae=sum(abs(g));

ita

**CHAPTER-5**

**CONCLUSION AND FUTURE SCOPE**

**5.1 Conclusion**

In chapter-2, we have first calculated Ziegler-Nichols based PID controller setting then we have online updated the proportional, integral and derivative gain based on the error. We have seen that APID based controller has better response compared to CPID. But major draw backs are that *k1, k2* , and *k3* are chosen by trial. So there is a further scope to achieve more improved performance if we can find the most appropriate settings of these parameters.

In chapter-3, we have studied computational intelligence-based optimization technique. In the introduction section we mentioned different search methods for optimization. We have studied how the GA is different than traditional methods. We have developed genetic algorithms based adaptive PID controller (GA-APID). We have tested the GA-APID with different process model and we have found that GA-APID gives much improved performance compared to CPID, GA-CPID, and APID for both set-point change and load disturbance.

In chapter-4, we have designed PSO based optimal PID controller (PSO-APID). We have tested a number of linear and nonlinear processes. Like GA-APID, here also, we have observed that PSO-APID gives improved performance compared to CPID, PSO-CPID, and APID for both set-point change and load disturbance.

In this study, GA and PSO based adaptive PID controllers are developed with certain objective function (minimization of *IAE, ITAE* or both) and tested on a number of process models without changing range of the search space (i.e., range of the variables) and other conditions. The results of GA and PSO based controller are almost similar.

Some common pints of GA and PSO are:

* Both do not require derivative information.
* Both are based on probabilistic transition rules, not deterministic rules.
* Both are the successful optimization techniques in the present aspect.
* Both Deals with a large number of variables.\
* Both optimizes variables with extremely complex cost surfaces (they can jump out of a local minima)
* Both provide a list of optimum variables not just a single solution.

In spite of these above similarities, there are some differences which are given below:

* GA is the natural evolutionary based search technique, whereas PSO is the artificial intelligence-based search technique.
* In the GA so many operations is involved due to the programming complexity.
* In case of PSO programming is easy.

GA starts with binary number (in case of binary coded GA), Therefore binary to decimal and vice versa conversion is required many times in the program. PSO starts with decimal number of each variable. Here we only have to bring all the initial solutions in to the defined range.

Convergence is slow in case of GA, whereas convergence is fast in case of PSO.

In case of GA mutation is used to avoid local optima, whereas in PSO random weight is used to avoid local optima (it is confirmed that both will reach at global optima).

**5.2 Future scope**

We have used Ziegler-Nichols based continuous cycling method for tuning relations .The main reason for selecting this tuning relation it its wider acceptance, simple relational expression. Other tuning relations may also be used, to start with, towards achieving more improved controllers.

In Chapters 3 and 4, while developing GA-APID and PSO-APID, respectively, we have considered range of variables for, are ±20% of their respective CPID, and for the four constants are and. Therefore, by increasing the range of the variables,, and , we may further improve the performance.

We have optimized *IAE, ITAE* and *IAE+ITAE.* In future we can optimize the multi-objective function with different weights, and other performance indices may be tried for achieving given performance specifications. Moreover, we may try with other optimization techniques.