Implementation of Real Value Genetic Algorithm To Determine Three PID Parameter

M Yusuf A A
School Of Electrical Engineering
Telkom University Bandung,
Indonesia
mochamadyusufabdulaziz@gmail.com

Agung N J
School Of Electrical Engineering
Telkom University Bandung,
Indonesia
agungnj@telkomuniversity.ac.id

Unang S
School Of Applied Science
Telkom University
Bandung, Indonesia
unangsunarya@telkomuniversity.ac.id

***Abstract*—This work is about generating optimum combination for PID parameter using genetic algorithm method. While genetic algorithm use evolutionary theorem like crossover and mutation, this work proposed modified fitness function by using Integral Time Weighted Absolute Error (ITAE) criterion, multiplying it with α as ITAE gain, summarize it with Mean Squared Error (MSE), multiplying with β as MSE gain, last its summarize with maximum overshoot error and multiplying with γ for overshoot gain. The result of this work will give you the best combination of *Kp*, *Ki*, and *Kd* as PID parameter gain with best performance in error sampling plant in the system.**

***Index Terms*—PID, ITAE Criterion, MSE, Maximum Overshoot, modified fitness.**

I. INTRODUCTION

Usual method used for tuning PID could be done by Ziegler-Nichols method, but demands field experience in the control system to get the best output. Manual tuning using trial and error will also be needed in the control system areas, PID tuning experience will play a good part too for getting better output in the plant system. Actually, there are many method that could be used such as in [1] which contain Ziegler-Nicholes, Ziegler Nicholes Modified, Cohen Coon Method, Tyrus Liben Method, Error Criterion and show comparison between those method.

This work is done by using Genetic Algorithm application to search the best combination of proportional gain, integral gain, and derivative gain in PID. Some modification in genetic algorithm were applied in this work, like the implementation of weighted multi objective Fitness calculation to get importance priority of the objective. Result of this work gave a self-tuning PID with optimum performance in error sampling for your plant system. Self-tuning PID concept have existed decade ago like

Autonomous Underwater Vehicle based on Taguchi Method [2], Self-tuning of Proportional-Integral speed controller gains using fuzzy logic controller [3], Self-tuning Proportional-Integral-Derivative control structures [4], Determine Proportional-Integral-Derivative using Genetic Algorithm [5].

This work contains design system of a genetic algorithm implementation used in PID tuning and modified fitness function used in genetic algorithm described in second section. Third section describe system implementation of the design as described in second section. Fourth section are the result of this genetic algorithm implemented in PID. Last section in this paper show conclusion and references used by this work.

II. SYSTEM DESIGN METHODOLOGY

First and foremost, Genetic Algorithm design should have been adapted and implemented in PID. [6] explained in the book that "the chromosomes in a Genetic Algorithm population typically take the form of bit strings. Each locus in the chromosome has two possible alleles: 0 and 1. Each chromosome can be thought of as a point in the search space of candidate solutions. The Genetic Algorithm processes populations of chromosomes, successively replacing one such population with another. The Genetic Algorithm most often requires a fitness function that assigns a score (fitness) to each chromosome in the current population. The fitness of a chromosome depends on how well that chromosome solves the problem at hand ".

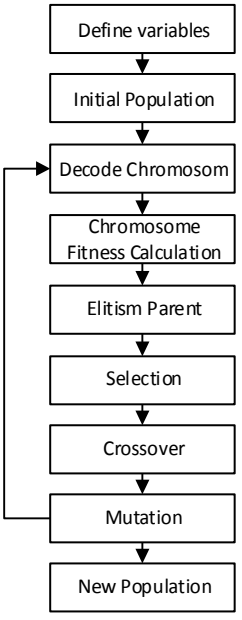

Fig. 1 General Binary String Genetic Algorithm

Typical bit strings form usually used in most genetic algorithm implementation, but [7] shows that "if the number of variables is large, the size of the chromosome is also large. Of course, 1s and 0s are not the only way to represent a variable. One could, in principle, use any representation conceivable for encoding the variables. When the variables are naturally quantized, the binary Genetic Algorithm fits nicely. However, when the variables are continuous, it is more logical to represent them by floating-point numbers. In addition, since the binary Genetic Algorithm has its precision limited by the binary representation of variables, using floating point numbers instead easily allows representation to the machine precision".

The continuous Genetic Algorithm known as Real Value Genetic Algorithm are implemented in this work. So, typical crossover used in bit string form will not work. Arithmetical Crossover as shown in [8] use weighted arithmetic mean of two parents. Children are feasible with respect to linear constraints and bounds. δ is random value between 0 and 1 with. If A as first parent and B as second parent, then A has better fitness after calculation, the function from A returned to the child.

$$= * + (1 - * \quad (1)$$

Mutation used in typical bit string form Genetic Algorithm will also not work in this case, so other mutation method will be used. Most people implement Real Valued Genetic Algorithm using add, a normally distributed random number to variable selected mutation.

$$= + \sigma * (0,1) \quad (2)$$

Where σ symbolize standard deviation of the normal distribution, and (0,1) is the standard normal distribution (mean=0, and variance = 1). But [7] didn't use that technique because good value for σ must be chosen, the addition of the random number cause the variable exceed its bounds, and it might have longer computing time.

Another method using Gaussian mutation operator like [9]. Let $x \in [$ be a real variable.

$$= \min(\max(, , , \quad (3)$$

$$:= b - a \quad (4)$$

$$\frac{\sigma}{} = \frac{1}{10} * \quad (5)$$

Then the Gaussian mutation operator M_G changes x defined in equation (3) where σ may depend on length defined in equation (4). Then the interval and typically used as equation (4) might be applied. The value of σ may also depend on time, i.e., the number of current generation, and σ usually decreases with time. The reason for a decreasing σ is that stronger mutation supports the sampling of the search space and smaller displacements towards the end aid in fine tuning extreme values during the beginning part of an optimization.

Offspring created by genetic algorithm operation after crossover and mutation will be included in next population and this process will loop until it reach maximum generation.

In genetic algorithm there some important guidelines introduced by [6] for setting crossover probability (p_c) and mutation probability (p_m) based population size use in genetic algorithm. These guidelines are shown by Tab. 1.

Tabel 1 Probability Guidelines

population size	crossover probability (p_c)	mutation probability (p_m)
30 (small population)	0.9	0.01
100 (large population)	0.6	0.001

As genetic algorithm will need fitness function for assesing how good a chromosome is, therefore this work proposed weighted multi objective fitness function that use Integral Time Weighted Absolute Error (ITAE), Mean Squared Error (MSE) and Maximum Overshoot. The proposed function shown in equation 6.

$$= \max(\frac{1}{((*) + (*) + (*) * 100}) \quad (6)$$

Where and is real number [0,1] and + + = 1.

III. IMPLEMENTATION

Based on the system design methodology, this work used real value genetic algorithm or some known as continuous genetic algorithm. Because of the search variable in genetic algorithm use float number for proportional controller gain (*Kp*), integral controller gain (*Ki*), and derivative controller gain (*Kd*). Every parameter has minimum and maximum value, and it's not limited to be the similar for each parameter. Every parameter will be evaluated with equation (6) and returned as a fitness function for its parameter combination.

As real value genetic algorithm, we cannot use typical crossover and mutation, so this work used an Arithmetic crossover as shown in equation (1) for pairing up after tournament selection process twice for getting two parent. Arithmetical crossover operation or pairing up parent are done in order to get new offspring and can only be done if generated number are below the defined crossover probability. Likewise arithmetical crossover operation and mutation use Gaussian mutation operator as used in equation (6). Note that it can only be done if generated number are below the defined mutation probability.

As a typical genetic algorithm, this work use elitism method to copy best chromosome or solution to a new population without genetic algorithm operation.

Tabel 2 Used Variables Defined

Variable Name	Value
Minimum Kp, Ki, and Kd	0
Maximum Kp	10
Maximum Ki	5
Maximum Kd	3
Population Size	100
Elitism Number	1
Crossover probability (p_c)	0.6
Mutation probability (p_m)	0.001
Maximum Generation	1000
□(ITAE Weight)	0.4
□(MSE Weight)	0.4
□(Max. OS Weight)	0.2

Genetic algorithm used in this work are shown step-by step and by following Fig. 3.

First, an initial population will create chromosomes by randomly generating a float number between minimum-maximum defined value for Kp, Ki, and Kd. Generated float number will fill three PID parameter and repeated until total chromosome reach its population size.

Each chromosome or solution will be evaluated by setting plant with the three PID parameter and record the error. After that fitness function will asses recorded error and return the fitness value of chromosome and repeated until all chromosome in the population have fitness value. Simple real plant used in this work is control the height of a swinging arm by varying the torque on a motor [10].

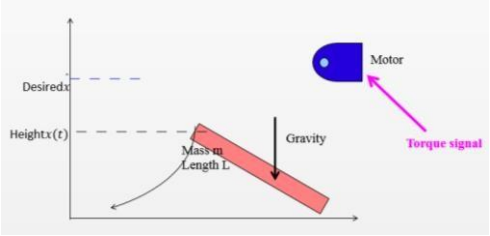

Fig. 2 Swinging arm plant

□ = *current position*

□ = *desired position*

□ = □ − □ *error signal*

$$\square = -\square \cdot \square - \square \int \square\,\square - \square \cdot \frac{\partial \square}{\partial \square} \quad (7)$$

Current population should be sorted descending by fitness value. And the first chromosome that the best chromosome in the population copied to next population as much as defined elitism number.

Tournament selection used in this work repeated twice to get two parent. If first parent chromosome same as second parent, then tournament selection run again until second parent got a different chromosome.

The offspring or child comes from Arithmetic Crossover Operation will go through Gaussian Mutation Operation to get an mutation offspring. The new offspring will be placed in new population. The genetic algorithm operation will be repeated until new population reached defined population size.

All step will be repeated from the second step by evaluating chromosome until creates new population. The generation number will be added by one after new population created and the genetic algorithm will stop after the generation reached defined maximum generation as shown in the table 2.

The best chromosome in the last population after reach maximum generation is the best three parameter PID that genetic algorithm can get as an optimum combination.

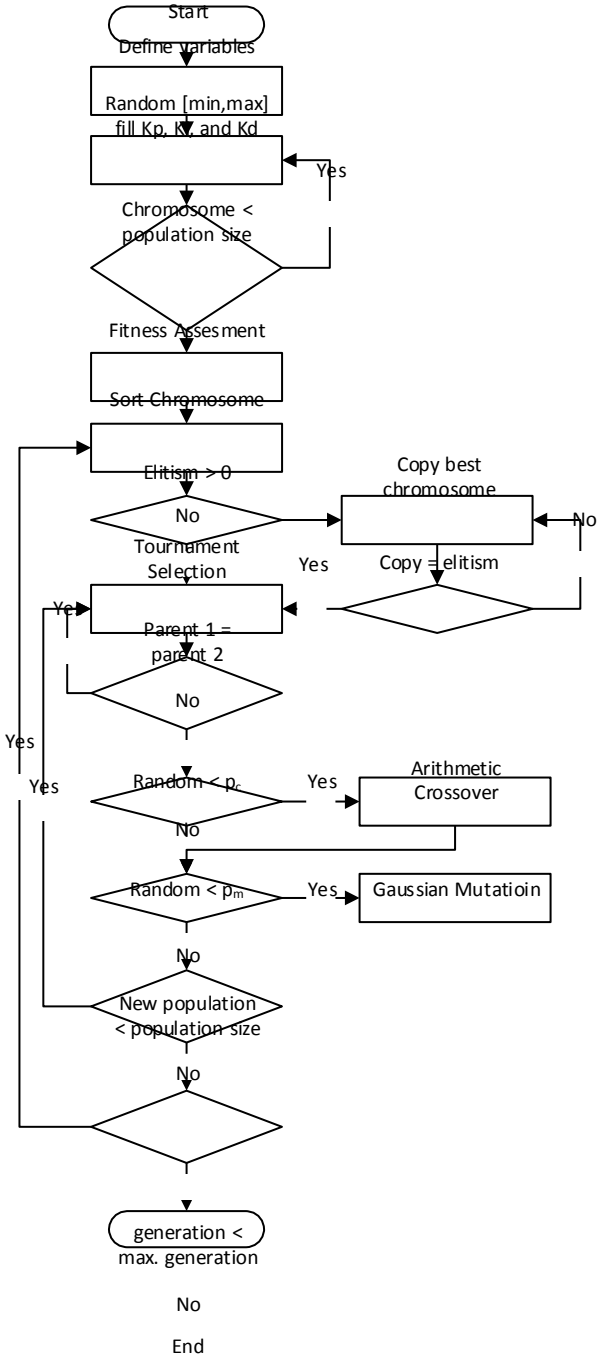

Fig. 3 Implementation Genetic Algorithm in PID

IV. RESULT

As result of real value genetic algorithm for proportional-integral-derivative controller gain,

A. Fitness value to generation

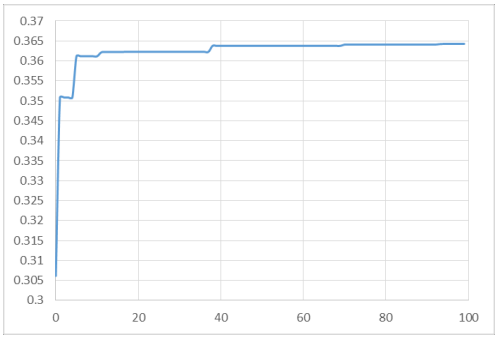

Fig. 4 Fitness to Generation first experiment

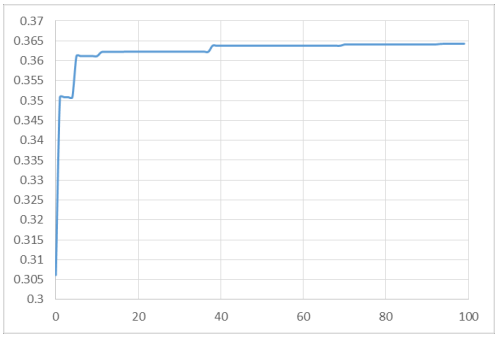

Fig. 5 Fitness to Generation second experiment

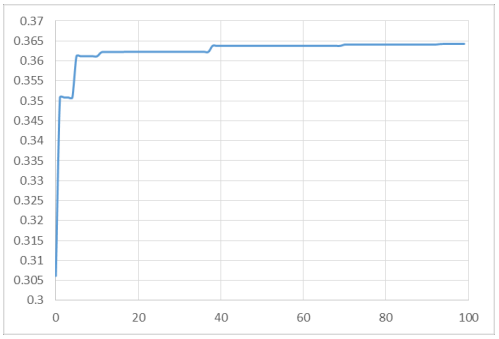

Fig. 6 Fitness to Generation third experiment

From the figure 4, figure 5, and figure 6 both look as same figure, it is caused from population size, elitism number, crossover probability and mutation probability use the same variable in table 2. All figure have improved by generation, in fifth generation fitness value improved to 0.35 from 0.3 and has better fitness in the last generation 0.364.

B. Maximum Overshoot to Generation

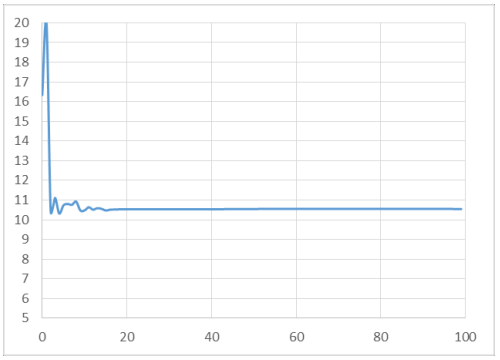

Fig. 7 Maximum Overshoot to Generation first experiment

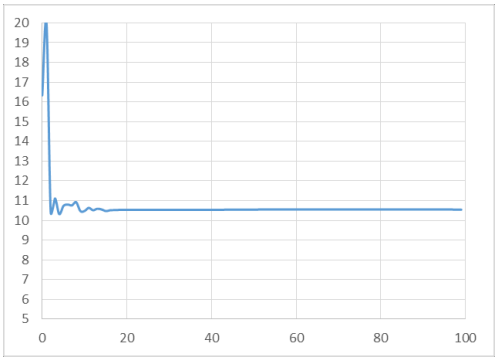

Fig. 8 Max. Overshoot to Generation second experiment

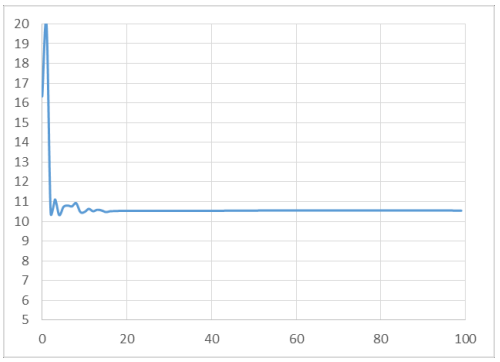

Fig. 9 Max. Overshoot to Generation third experiment

From the figure 7, figure 8 and figure 9 shown improved maximum overshoot from best proportional-integral-derivative gain controller. Second generation both all figure show highest overshoot all the time that has 19.9, its almost 100% error from the target position. But, from generation 15th until last generation, maximum overshoot shown steady in 10,5.

C. Rise Time to Generation

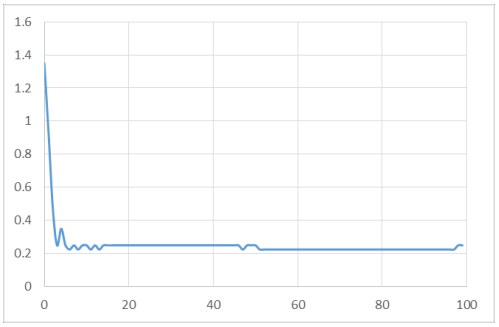

Fig. 10 Rise Rime to Generation first experiment

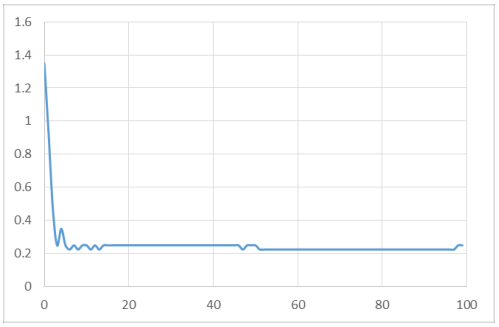

Fig. 11 Rise Time to Generation third experiment

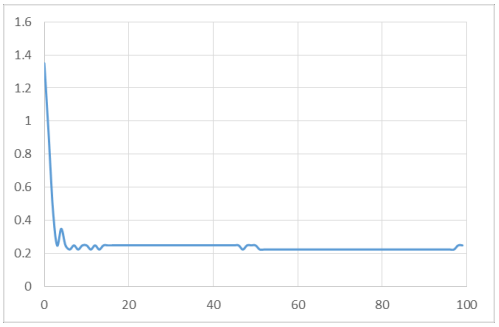

Fig. 12 Rise Time to Generation second experiment

Figure 10, figure 11, and figure 12 has a worst rise time with 1.35 ms, but the rise time both all figure decreased to 0.25 ms in 5th generation and have best rise time 0.225 even though last generation has increase rise time to 0.25 ms.

D. PID Parameter Gain Output

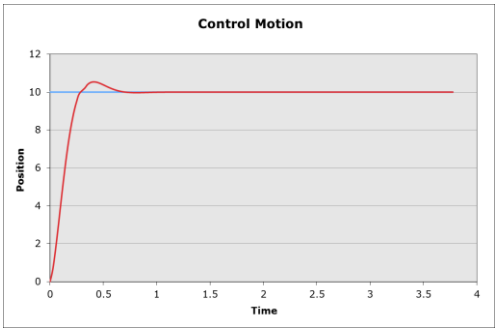

Fig. 13 PID Response with Last Best Generation in first experiment

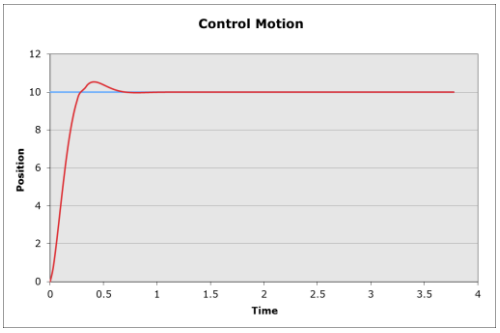

Fig. 14 PID Response with Last Best Generation in second experiment

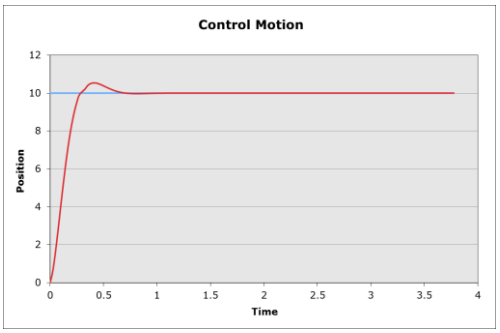

Fig. 15 PID Response with Last Best Generation in third experiment

After three times experiment, with 100 generation and 30 chromosome or solution each population, both figure 13, figure 14 and figure 15 has the same proportional-integral-derivative gain controller in the last generation with 9.839 proportional gain, 1.977 integral gain, and 1.242 derivative gain.

V. CONCLUSION

Implementation of real value genetic algorithm for determining PID parameter runs very well in this case. The result especially both figure 4 and 8 described how well the fitness value improved between generation until it reach maximum generation. As good as fitness value to the generation, rise time and maximum overshoot shown are also improving faster and the result generate minimum overshoot as the operation goes by.

This operation run three times with user defined variable, the result showed similar PID Parameter. This work run perfectly and gave some good result. System have faster rise time with maximum overshoot which are minimum. Weighted fitness function for Integral Weighted Time Absolute Error, Mean Squared Error and Maximum overshoot gave some difference value of the three PID parameter.

ACKNOWLEDGMENT

This work supported by Electronic and Intelligence Robotic Research Group, School of Electrical Engineering Telkom University. This work also supported by Ministry of Research, Technology and Higher Education.

REFERENCES

[1] M. Shahrokhi and A. Zomorrodi, "Comparison of PID Controller Tuning Methods," *8th National Iranian Chemical Engineering Congress*, 2003.

[2] M. Santhakumar and T. Asokan, "A Self-Tuning Proportional-Integral-Derivative Controller for an Autonomous Underwater Vehicle, Based On Taguchi Method," 2010.

[3] M. Nour, O. Bouketir and C. E. Yong, "Self-Tuning of PI Speed Controller Gains Using Fuzzy Logic Controller," 2008.

[4] P. J. Gawthrop, "Self-tuning PID control structures," 1996.

[5] D. Joko, A. Warsito and A. Triwiyatno, "Penalaan parameter pengendali PID dengan Algoritma Genetik," 2011.

[6] J. J. Grefenstette, "Optimization of controll parameter for Genetic Algorithm," *IEEE Trans Systems, Man, and Cybernetics*, vol. 16, pp. 122-128, 1986.

[7] R. L. Haupt and S. E. Haupt, Practical Genetic Algorithm, Second Section, New Jersey: John Wiley & Sons, Inc., 2004.

[8] Y. KAYA, M. UYAR and R. TEKĐN, "A Novel Crossover Operator for Genetic Algorithms: Ring Crossover," Cornell University, New York, 2011.

[9] C. Heitzinger, "Simulation and Inverse Modeling of Semiconductor Manufacturing Processes," 8 05 2003. [Online]. Available:

http://www.iue.tuwien.ac.at/phd/heitzinger/node27.html. [Accessed 20 May 2015].

[10] D. S. Touretzky, *Forward and Inverse Models*, Pittsburgh: Carnegie Mellon School of Computer Science, 2013.