Abstract—Servomotor uses feedback controller to control either the speed or the position or both. This paper discusses the performance comparisons of a modified genetic algorithm, named as the semi-parallel operation genetic algorithm (SPOGA) and the conventional genetic algorithm (GA), in optimizing the I/O scale factors, membership functions, and rules of a hybrid-fuzzy controller. Singleton fuzzification is used as a fuzzifier with seven membership functions for both input and output of the controller, whilst center of average is used as a defuzzifier. A 21-bit-30-population is used in SPOGA for both I/O scales and for membership functions. Two control modes are applied in cascaded: position and speed. Both the simulation and practical experiment results show that fuzzy-logic parallel integral controller (FLC) with SPOGA-optimized is better as compared to FLIC with GA-optimized and also the non-optimized FLIC, FLC, and PI in terms of performance and the reduction of the number of test runs for the optimization.

Index Terms—Hybrid-fuzzy controller, Genetic algorithm, servomotor, SPOGA

I. INTRODUCTION

Servomotors are used in a variety of applications in industrial electronics that includes precision positioning as well as speed control. Basically, any motor can be used in a servo system [1]. Although more expensive, DC motors have better starting torque than AC motors. The main components of a servo system are: motor, torque control system, and velocity control system [2].

Servomotors use feedback controller to control the speed or the position or both. The basic continuous feedback control is a PID controller. The PID controller has good performance but is not adaptive enough. To overcome the weakness of PID controller, Prof. Zadeh in 1968 proposed a fuzzy theory for control [3] and investigated by which owns good robustness [4]. Fuzzy logic idea is similar to the human being's feeling and inference processes. Unlike classical control strategy, which is a point-to-point control, fuzzy logic control is a range-to-point or range-to-range control [5].

Fuzzy systems are capable of handling complex, non-linear and sometimes mathematically intangible dynamic systems using simple solutions. It requires time, experience and skills of the designer for the tedious fuzzy tuning exercise [6] because it lacks a learning mechanism [7] and the response of a fuzzy logic controller is slower than a PID controller.

It has been reported in a number of papers that hybrid of PID or PI, with fuzzy logic in control system can overcome the set-back of a fuzzy logic controller, see for example [8]. GA is effective in acquiring the optimal or near-optimal for solving optimization problems [9]. The typical task of a GA in a control engineering application is finding the best values for a predefined set of free parameters which defining either a process model or a control law [10].

The contribution of this work is on investigation of the performance using SPOGA with the reductions of population size and generation number in which inflicts the test runs reduction to optimize the membership functions and rules of FLC, I/O scale parameters and integral constant, and the implementation on a servomotor control.

This paper aims to discuss the control methodologies and the optimization algorithm that is the focus of this work and to present some part of the simulation and practical experimental examples to illustrate the effectiveness of the proposed optimization method.

II. FLC, GA AND SPOGA

A. FLC

In fuzzy systems, the numerical input values should be first converted into the corresponding fuzzy representations by using a 'fuzzifier'. The fuzzy outputs are then provided by a fuzzy model, which could be a set of fuzzy logic rules, fuzzy relations or a simple fuzzy table, with or without deep fuzzy reasoning. Finally, the fuzzy output can be converted back into their relevant numerical (crisp) outputs through 'defuzzifiers' [11]. Basic configuration of fuzzy systems with fuzzifier and defuzzifier is shown in Fig. 1.

Singleton fuzzifier maps a real-valued point \( x^* \in U \) into a fuzzy singleton \( A' \) in \( U \), which has membership value 1 at \( x^* \) and 0 at all other points in \( U \) [10]. There are three criteria in designing fuzzifier [12]:

1) The fuzzifier should consider the fact that the input is at the crisp point.
2) If the input to fuzzy system is corrupted by noise, then the fuzzifier should help to suppress the noise.
The fuzzifier should help to simplify the computations involved in the fuzzy inference engine.

The singleton fuzzifier, is chosen over the Gaussian and triangle fuzzifiers due to its simplicity and can fulfill the above three criteria.

If $y$ is the center of the $i^{th}$ fuzzy set and $W_i$ is its height, then the center of average defuzzifier determines $y^*$ as [12].

$$y^* = \frac{\sum_{i=1}^{n} y_i W_i}{\sum_{i=1}^{n} W_i}$$

### B. GA and SPOGA

The thrust of the idea for proposing this algorithm in this work comes from the intrinsic parallelism architecture of the conventional GA (PGA) and the sub-partition of chromosomes in hierarchical GA (HGA), with the chromosomes separated into some sub-chromosomes according with the problem to be solved to reduce the test runs in the optimization process.

In its general structure, a GA approach would require the following three processes: 1) initialization; 2) evaluation; and 3) genetic operators which consist of three sub-processes namely the selection, crossover, and mutation, [13]. Basically, the GA would optimize the $KP_m$ and $KI_m$ of FLIC. The process of genetic algorithms is shown in Fig. 2. Twisted ring counter principle is used for initialization process instead of using random generations.

In brief, the SPOGA technique involves the following steps:
- Step 1: Divide the chromosome into sub-chromosomes.
- Step 2: Recombine into one chromosome.
- Step 3: Select populations to be crossover and mutated.
- Step 4: Conduct fitness evaluation.
- Step 5: Repeat Steps 1, 2, 3 and 4 until achieving best fitness using elitism process.

Following this, crossover and mutation are conducted for the next generations until getting the maximum generation (set by the user) and then finding the best fitness among the best fitness in a generation using elitism process [14].

### III. SIMULATION AND PRACTICAL EXPERIMENTS

#### A. Description of Simulation Experiment

Simulation experiment on a DC servomotor for the speed and position control was conducted using Simulink, and the block diagram is shown in Fig. 3. The speed control loop is in the position control loop [15]. Basically, the purpose of the control is to bring the position to its set point while the speed is limited to the speed set point.

The input, feedback, and output elements for position and speed are implemented in the Simulink diagram with specifications as follows:

- $A_p=1, A_v=0.002, K_v=9.5455, H_v=0.002, H_p=0.005, K_p=0.005$
- These specifications have been based on realistic assumptions. The hardware implementation for the block diagram is as follows:
- DC motor: 175 W, 1500 rpm, 240 V, 1.1 A
- Load: dynamometer load controller
- **Input elements:**
  - Tacho-generator with 500 rpm/volt
  - ADC: 1 channel 0 to 10 volts
- **FIR with 30 points**
- **Output elements:**
  - DAC: 2 channels 0 to 4 volts
  - Differential amp: HA-17741
  - Power amp: Chopper/Inverter and IGBT
- **Control elements:** Computer with Intel Pentium Core processor. Windows XP SP3 Home, MATLAB/Simulink software. The set point for position (rad in) is 3.5 rad, and for the speed (rpm in) is 275 rad for 90 sec.

The speed of DC motor is detected by a tacho-generator, sampling period of 0.01 sec, and filtered by 30-point finite impulse response (FIR) filter.

#### B Identification Process

The identification process consists of stages where the model structure is iteratively selected for the best model in the structure, and evaluation of the model's properties [16]. This cycle can be itemized, as follows:

1. Design an experiment and collect input-output data.
2. Examine and polish the data to remove trends and outliers. Apply filters to enhance important frequency ranges
3) Select and define a model structure.
4) Compute the best model according to goodness of fit.
5) Examine the properties of the model. If the model is not up to satisfactory, repeat the steps 2-4.

**C Conventional Controllers**

The conventional controllers, PI and PID are optimized using Ziegler-Nichols (ultimate cycle) method. During the experiments, there are some noises originate from the hardware components. Since the D-term of PID is too sensitive to any disturbances [17], the PI controller is selected. If \( K_{pu} \) is the minimum value of \( KP \) resulting in an undamped oscillation and \( Tu \) is the oscillation period, then

\[
K_p = 0.45 \cdot K_{pu} \quad \text{and} \quad K_i = \frac{1}{0.83 \cdot T_u}
\]  

**D. Design of Fuzzy Logic Parallel Integral Controller (FLIC)**

The fuzzifier used is a singleton mode with two inputs: error and change of error, and one output. There are seven uniform triangular membership functions for both inputs and output. As an inference engine, the Mamdani product is used whilst the center of average is used as the defuzzifier. The controllers use the initial rules as shown in Table I.

If \( E_i, D_j, \) and \( U_k \) are the linguistic names which characterize the fuzzy subsets of \( e, \Delta e, \) and \( \Delta u, \) respectively [18], then the fuzzy rule table can be coded into the fuzzy rule chromosome \( H_{(w,x,y)} \) which is formulated in the form of an integer matrix

\[
H_{(w,x,y)} = \begin{bmatrix}
h_{11} & \cdots & h_{1i} \\
\vdots & \ddots & \vdots \\
h_{wi} & \cdots & h_{wi}
\end{bmatrix}
\]  

In this experiment, \( i = 4, j = 2, k = 4, w = 7, x = 7, y = 7. \)

The fuzzy sets and their corresponding membership functions for input (error, and change of error) are shown in Fig. 5. The fuzzy sets and their corresponding membership functions for output rate are shown in Fig. 6. The output of FLC is then paralleled with integral controller to get the manipulated variable with the range value from 0 to 10. This value is then conditioned and becomes the input to the plant (DC motor).

**E Simulation of GA and SPOGA**

Simulation of GA is conducted to determine the ideal population size according to the bit length with the number of generation equals 20. The simulation is done for both the GA and SPOGA to obtain the maximum value of the function as in Eq. (4) and Eq. (5).

\[
f = \max(f_1(x) + f_2(x) + f_3(x) + f_4(x))
\]

where

\[
f_1(x) = -x^3 + 2x \quad \text{(5a)}
\]
\[
f_2(x) = -x^3 + 4x \quad \text{(5b)}
\]
\[
f_3(x) = -x^3 + 6x \quad \text{(5c)}
\]
\[
f_4(x) = -x^3 + 8x \quad \text{(5d)}
\]

Based on the functions in Eq. (4) and Eq. (5), with the range of \([0.00, 10.00]\) and the resolution of 0.01, the length of the string is 40 bit. The minimum population size is statistically 30. It is then gradually increased to 40, 50, and higher until fulfilling a criteria. When the population size is less than 100, the crossover rate is 0.9 and to avoid the premature convergence, the mutation rate is set to 0.01, see [19] and 0.1 see [20]. When the population size is greater than or equal to 100, the crossover rate is 0.6 and to avoid the premature convergence, the mutation rate is set to 0.01, see [19]. The results from GA and SPOGA to solve Eq. (4) are compared to the result from manual calculations.

The minimum criterion is as follows: If the % error is less than or equal to 5 then the GA result is considered as true. Otherwise, the GA result is false. The experiments are repeated 100 times and the accuracy is calculated by counting the false result within the 100 experiments. The accuracy is 100 % if there are no false results. The ideal population size for these criteria is the minimum population size in the resolution of 10 with 100 % accuracy and less than 2 % average error.
TABLE I
FUZZY RULES BASE FOR $Ke$ AND $Ku$ IN HYBRID [11]

<table>
<thead>
<tr>
<th>Change of Error</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Z</td>
</tr>
<tr>
<td>Z</td>
<td>S</td>
</tr>
<tr>
<td>P</td>
<td>S</td>
</tr>
</tbody>
</table>

$F$ SPOGA Optimization
Practically, fuzzy logic controllers need I/O scales to work in the control system loop. The structure of fuzzy logic controller is shown in Fig. 7.

![Fig. 7. Structure of fuzzy logic controller](image)

Prior to optimizing the parameters $K_e$, $Kce$, $K_u$ and $K_i$ for speed control, the first step is optimizing the triangular membership functions and rules. The fuzzy logic initially uses two inputs: error and change of error, seven uniform triangular membership functions, and one output with seven triangular membership functions in which the values were optimized by SPOGA as shown in Fig. 5 and Fig. 6. According to the FLC specification, there is 21 bits of chromosomes. Ideally, there will be $2^{21} = 2,097,152$ chromosomes in a population size, which will require too many computations. The fuzzy rules in Table I is coded into the fuzzy rule chromosome $H_{i,j,k}$, which is formulated in the form of an integer matrix. The genetic operation for rules is bit mutation with the formula as follows:

$$h_{ij} = h_{i+ai,j+aj}$$

where $ai$, $aj$ can be 1 or -1 with probability $0.01$, see [19].

The input values of error $e$ and change of error $\Delta e$ are scaled into $E$, $D$ and grouped into fuzzy sets. In other words, the input values are labeled and transformed into linguistic variables. The varying range of $E$ and $D$ is defined in the range of $[-10.00, +10.00]$, which are divided into seven grades, specifically the {NB, NM, NS, Z, PS, PM, PB}. Here, $Ke$ and $Kce$ are the transforming parameters in the fuzzifier. $K_u$ is a parameter, which transforms the computed control variable ($U$) into real time control variable [13].

The next step is optimizing the parameters $Ke$, $Kce$, $K_u$ and $K_i$ for speed controller. Experimentally, the $K_u$ had no effect to the performance. Consequently, the GA optimizes $Ke$, $K_u$ and the integral scale of integrator $K_i$. According to the specification, and there will be 21 bits of chromosome which eight bits binary for $Ke$, six bits binary for $K_u$, and seven bits binary for $K_i$. The population size is based on the simulation result of using SPOGA and the number of generation is 20. Roullete wheel is used as a selection method [18], [19], [22], the probability of crossover is 0.9, see [18] and the probability of mutation is 0.01, see [18].

If the parameter $K_e$ is defined to be within range of $[K_{e_{min}}$, $K_{e_{max}}]$, the relation between the binary code character string $st(i)$ and parameter $K_e$ is given as follows [13]:

$$K_e = K_{e_{min}} + \frac{G(bin(i))}{2^k - 1} (K_{e_{max}} - K_{e_{min}})$$

where the $bin(i)$ code of $K_e$ is shown as [21]:

$$G(bin(i)) = \left( \sum_{n=0}^{k-1} h_n 2^i \right)$$

Experimentally, the range of $K_e$ is [0.00, 1.50], $K_u$ is [0.00, 0.50] and $K_i$ is [0.00, 1.00].

$G$ Position Controller and Fitness Evaluation
It is sufficient to use Proportional (P)-based controller for the position control as the motor just rotates in one direction. The P- controller ($K_{pp}$) is tuned experimentally with no overshoot criteria.

Integral of time absolute value of error (ITAE) is used as a performance index for position control ($ITAE_p$) and the first 8-second of starting speed control ($ITAE_{sp}$) in which the formulas are as follows [24]:

$$ITAE = \frac{1}{2} \int_0^t [SP(t) - PV(t)] dt$$

where $x = vp$; $n=8$ for the first 8-second of starting speed control, and $x = p$; $n=90$ for the position control, and the fitness value is obtained from the basic formula [23]:

$$f(i) = \frac{ITAE_p - ITAE(i)}{\sum_{i=1}^{n} ITAE_{max} - ITAE (i)}$$

Using Eq. (10), the fitness values for first 8-second starting speed ($f_{sp}$), for position ($f_p$), for speed overshoot ($f_{os}$), for settling time of speed ($f_{ts}$), and for steady state error of position ($f_{pp}$) is formulated as follows:

$$f_{sp}(i) = \frac{(A_{max} - A(i))}{\sum_{i=1}^{n} (A_{max} - A(i))}$$

where $A = ITAE_{sp}$; $x = vp$ for first 8-second starting speed, $A = ITAE_p$; $x = p$ for position, $A = %O_s$; $x = os$ for speed overshoot, $A = T_s$; $x = ts$ for settling time of speed, and $A = S_p$; $x = sp$ for steady state error of position. If $fit_p$ is the total fitness function for speed control, $fit_p$ is the total fitness function for position control, and fit is the total fitness function for speed and position control, then these fitness functions are formulated as follows:

$$fit_p = \frac{f_{sp} + f_{os} + f_{ts}}{2}$$

$$fit_p = \frac{f_{sp} + f_{pp}}{2}$$

$$fit = \frac{2fit_p + fit_p}{2}$$

$$fit = \frac{2fit_p + fit_p}{2}$$
If $fit_{v,g}$ is the total fitness function of SPOGA-optimized hybrid-fuzzy controller for speed control and $fit_{v,h}$ is the total fitness function of non-GA-optimized hybrid-fuzzy controller for speed control, then the improvement value for speed control, $I_{pv}$, is formulated to be:

$$ I_{pv} = fit_{v,g} - fit_{v,h} \quad (13) $$

VI. SAMPLE RESULTS

Selecting the best process model in the experiment, the transfer function ($s$-domain model) of the best model is as shown in Eq. (14):

$$ \frac{\text{Speed}_{in}(s)}{\text{Voltage}_{in}(s)} = G(s) = \frac{456.3713}{s^2 + 9.5040s^2 + 80.7000s + 204.5000} e^{-0.1682s} \quad (14) $$

Graphical comparison of actual (real time) response and $s$-model (estimated) response in open loop analysis is shown in Fig. 8.

![Graphical comparison of actual (real time) response and s-model (estimated)](image)

Fig. 8. Graphical comparison of actual (real time) and estimation ($s$-model)

The simulation result of GA and SPOGA in the optimization of the function in Eq. (4) and Eq. (5) is as follows:

To fulfill the criteria, the GA needs 80 population size for $p_m=0.01$ and 90 population size for $p_m=0.1$ in 20 generations. This means GA needs (80x21) or 1,680 test runs for $p_m=0.01$ and 1,890 for $p_m=0.1$ to get the fitness value. Meanwhile, SPOGA needs 40 population size for $p_m=0.01$ and 50 population size for $p_m=0.1$ in 20 generations. This means that SPOGA needs 840 test runs for $p_m=0.01$ and 1,050 for $p_m=0.1$ to get the fitness value. Therefore, SPOGA can make the reduction of (1,680-840) or 840 test runs for $p_m=0.01$ and (1,890-1,050) or 840 runs for $p_m=0.1$. In other words, SPOGA can reduce 50% test runs for $p_m=0.01$ and 44.4% test runs for $p_m=0.1$. Notably, the SPOGA optimizes the error membership functions, change of error membership functions and output membership functions. The result of genetic operation for membership functions gives the best string in the 11th generation of a 26th chromosome as follows:

```
111111110001000000
```

Using this value, the rules, input and output membership functions are shown in Table II, Fig. 9 and Fig. 10 respectively. The result of genetic operation for I/O scales that gives the best string in the 10th generation as a 26th chromosome is as follows:

```
10111100111111000111
```

Using Eqs (8) and (9), the I/O scales and constant are given as: $K_e=1.50; K_ce=1.00; K_u=0.00; K_i=0.54$. Experimentally, the parameter of position controller is $K_{pp}=0.69$.

The PI and FLC have parameters as follows:

$K_p=10.58$ and $K_f=14.70$ for PI controller (Eq. (2))

$K_e=K_ce=K_u=1.00$ for FLC (experiment)

$K_{ce}=1.03; K_u=1.00; K_i=0.25; K_i=1.00$ (experiment for FLC)

$K_{pp}=0.61$ (experiment for PI); $K_{pp}=0.35$ (experiment for FLC); $K_{pp}=0.69$ (experiment for FLC);

![Graphical comparison of actual (real time) response and s-model (estimated)](image)

Fig. 9. Input membership functions of SPOGA-optimized FLC: (a) error, (b) change of error

![Graphical comparison of actual (real time) response and s-model (estimated)](image)

Fig. 10. Output membership functions SPOGA-optimized FLC

![Graphical comparison of actual (real time) response and s-model (estimated)](image)

Fig. 11. Step response of speed control of DC servomotor using SPOGA optimized FLC vs FLC

![Graphical comparison of actual (real time) response and s-model (estimated)](image)

Fig. 12. Step response of speed control of DC servomotor using SPOGA optimized FLC vs PI controller

Plot for the speed control in the test for SPOGA-FLIC vs. FLC is presented in Fig. 11. The FLC and PI are noisier than
SPOGA-FLIC, and the ITAE for FLIC and PI are larger than of SPOGA-FLIC. The result for speed control for SPOGA-FLIC vs. PI is presented in Fig. 12, where the settling time of FLIC is slightly faster than SPOGA-FLIC, albeit SPOGA is used for optimizing the speed controller.

The comparisons on the effectiveness of the PI, FLC, FLIC and FLIC-GA on the performance index are summarized in Table III.

<table>
<thead>
<tr>
<th>No.</th>
<th>PERFORMANCE ITEM</th>
<th>PI</th>
<th>FLC</th>
<th>FLIC</th>
<th>SPOGA-FLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ovs (%)</td>
<td>1.4448</td>
<td>14.5732</td>
<td>6.3121</td>
<td>2.2706</td>
</tr>
<tr>
<td>2</td>
<td>Ts (sec)</td>
<td>4.0600</td>
<td>6.0200</td>
<td>5.1000</td>
<td>2.0500</td>
</tr>
<tr>
<td>3</td>
<td>SSEP (%)</td>
<td>0.0885</td>
<td>0.8971</td>
<td>0.0284</td>
<td>0.0865</td>
</tr>
<tr>
<td>4</td>
<td>ITAE vp</td>
<td>2.27E+02</td>
<td>4.55E+02</td>
<td>1.80E+02</td>
<td>1.72E+02</td>
</tr>
<tr>
<td>5</td>
<td>ITAE p</td>
<td>3.92E+02</td>
<td>5.31E+02</td>
<td>3.78E+02</td>
<td>3.90E+02</td>
</tr>
<tr>
<td>6</td>
<td>fvp</td>
<td>0.2895</td>
<td>0.0000</td>
<td>0.3500</td>
<td>0.3605</td>
</tr>
<tr>
<td>7</td>
<td>fp</td>
<td>0.3216</td>
<td>0.0000</td>
<td>0.3537</td>
<td>0.3247</td>
</tr>
<tr>
<td>8</td>
<td>fitv</td>
<td>0.3216</td>
<td>0.0000</td>
<td>0.2431</td>
<td>0.4347</td>
</tr>
<tr>
<td>9</td>
<td>fip</td>
<td>0.3233</td>
<td>0.0000</td>
<td>0.3514</td>
<td>0.3253</td>
</tr>
<tr>
<td>10</td>
<td>fit</td>
<td>0.3222</td>
<td>0.0000</td>
<td>0.2792</td>
<td>0.3983</td>
</tr>
<tr>
<td>11</td>
<td>lvp</td>
<td></td>
<td></td>
<td></td>
<td>0.1917</td>
</tr>
</tbody>
</table>

For position control, the FLIC gives the smallest steady state error related to position (SSEP) and also the smallest fitness function based on $ITAE_p$ ($f_p$). The total fitness function for speed control and position control with the setpoint of 275 rpm, 3.5 rad is by SPOGA-FLIC. On overall, SPOGA-FLIC is the better controller than PI, FLC, or FLIC. The practical experimental results in the Table III reveal the improvement in the performance when having a SPOGA-optimized controller as compared to a non-SPOGA-optimized controller. The main findings are summarized as follows:

- FLC alone is not performing as good as conventional controller. Devising a hybrid-fuzzy controller improves the performance of FLC.
- SPOGA reduces up to 50% of test runs in the optimization process, and improves the performance of FLIC.

V. CONCLUSIONS

Within the servomotor speed and position control using hybrid-fuzzy controller, the aim of SPOGA is to improve the overshoot, settling time, IAE/ITAE and achieving zero steady state error. Detailed performance comparisons of controllers for a DC servomotor speed and position control have shown that SPOGA-optimized controllers perform better in reducing the test runs. These findings demonstrate the effectiveness of SPOGA as an optimization algorithm for a hybrid-fuzzy controller. Further improvement should include: 1) applying variable crossover and mutation rates to speed up convergence and restrain premature convergence, and 2) to evaluate SPOGA’s performance against other GA optimization algorithms to acquire better perception of its capability.

ACKNOWLEDGEMENT

The authors acknowledge the supports of the University Teknologi PETRONAS, Malaysia.

REFERENCES