Chapter

Fuzzy Logic Control with PSO Tuning

Jeydson Lopes da Silva

Abstract

Several applications of artificial intelligence in the area of control of dynamic systems have proven to be an efficient tool for process improvement. In this context, control systems based on fuzzy logic - Fuzzy Logic Control (FLC) are part of a series of advances in the areas of control systems. Fuzzy control is based on natural language and therefore has the ability to make approximations closer to the real nature of the problems. The use of metaheuristic algorithms such as the particle swarm optimization (PSO) allows it to provide adequate adjustments to the fuzzy controller in an optimized manner. This technique allows to adjust the FLC in a simple way according to the performance desired by the designer, without the need for a long time of conventional tests.

Keywords: FLC, PSO, artificial intelligence, controller, optimization

1. Introduction

FLC represents a family of intelligent controllers with a lot of potential for use in the world for industrial control systems. Its popularity is mainly due to its performance being robust in several operating conditions and its functional simplicity, in addition to its ease of implementation, allowing engineers to operate them in a simple and direct way. Even with the emergence of new control techniques, FLC controllers will still be on the market for a long time in industrial plants [1].

A good parameterization of the FLC inference functions is essential to allow a good performance of this type of closed loop controller. The tuning of the controller is a persistent problem in the area of control and automation, from the initial approach of this topic to the present day; a definitive solution has not yet been reached, being a subject constantly addressed in several works in the field of control engineering. However, it is important to note that despite the various techniques that produce adjustments in the FLC parameters, it is still necessary to assess the designer regarding the result of the parameterization of this controller [2].

In recent years, the computational capacity available allows optimization techniques developed in the field of artificial intelligence to gain space in the solution of several engineering problems [3, 4]. In this context, algorithms based on metaheuristics can provide adequate solutions to the FLC parameterization problem. Since the parameters necessary for the proper functioning of the FLC can be numerous and often complex, techniques based on intelligent computing provide an alternative solution to this type of problem.

2. Particle swarm optimization

Particle Swarm Optimization (PSO) algorithm is a population metaheuristics created from models of the collective and social behavior of animals, in their coordination of movements in the tasks of searching and obtaining food. These models were simplified, losing the requirement to maintain a minimum distance between its neighbors. In addition, the communication architecture was transformed, which was initially inspired by spatial proximity and was changed to use a topology defined by a graph. Therefore, PSO ends up having, nowadays, more similarities with models of mutual influence between human beings in their ways of thinking and acting [5].

In the PSO there is a fixed amount of agents, called particles. This set of agents is called a swarm, designed in such a way that each agent is able to communicate with its neighbor, which is a subset of its peer, and can be defined in a static or dynamic way. Each of the particles moves in the solution space with a certain speed, always evaluating the solution corresponding to that occupied position in each iteration. The speed of the particles must be influenced by your own experience (cognitive factor), and also influenced by your neighbors (social factor). Such influences are implemented as two attractors, the first located in the best position already evaluated by the particle itself and the second located in the best position visited by the neighboring particles [6, 7].

In general, the position of a particle i at a time k is represented by a vector x_k^i and its respective speed as a vector v_k^i. Both vectors are stored during the learning process in generation k and used for updating in the next generation. In addition to the vectors x_k^i and v_k^i, PSO uses the best position of the particle i (p_k^i), as well as the position of the best particle of the whole swarm (p_k^g), both evaluated throughout the process until the moment k evaluated.

The equations that govern the PSO can be defined as

$$v_{i+1} = \omega v_i + c_1 r_1 \left(p_i - x_i\right) + c_2 r_2 \left(p_g - x_i\right) \tag{1}$$

$$x_{i+1} = x_i + v_{i+1} \tag{2}$$

where x is the position of the particle; v is the velocity of the particle; w is the inertial weight, controlling the impact of the previous speed at the current speed; c_1 and c_2 are positive constants, controlling the social and individual behavior of each particle; r_1 and r_2 are random numbers in the interval [0.1], contributing to diversify the exploration of the problem search space.

The basic PSO algorithm can be defined as follows.

Randomly initialize the particle positions x_k^i and speeds v_k^i within the search space at k = 0.

Algorithm 1: PSO algorithm

Randomly initialize the particle positions x_k^i and speeds v_k^i within the search space at k= 0;
while *Unsatisfied stop conditions* **do**
  Evaluate the objective function of each of the population particles;
  Update the best position of each particle individually p_k^i and the best position of the swarm p_k^g;
  Update the position of each particle in time *k+1* based on position and speed in time *k*, based on equations (1) and (2);
end

3. Fuzzy logic controller

From the concepts of fuzzy logic, which include fuzzification, defuzzification, inference system and compositional rules, it was possible to build a fuzzy logic controller, where control actions are generated from a set of linguistic knowledge. In general, the FLC is conceptually easy to understand. Despite its easy understanding, its design requires a greater amount of parameterization than normal in other controllers [8].

Several previous works attest that the application of FLC can overcome the effectiveness of traditional controllers in several tasks. In fact, the FLC works perfectly in a dynamic control environment, its steps can be similar to other controllers. In the fuzzification stage, the signals from the plant's sensors and any other information that is declared important are mapped to the pertinence functions and appropriate truth values. Then, the processing step makes use of each appropriate rule, thereby generating a particular result individually and then combines the results of those rules. Finally, the result of the operation in the previous stage is then converted into the defuzzification to an output value according to the system in which the FLC is inserted [9, 10].

Regarding the FLC memberships functions (MFs), it should be noted that the most applicable functions are the triangular, trapezoidal and Gaussian MFs type. This is due to the fact that. In the FLC project there is no exact number of how many curves, or what types of curves must be inserted in order to improve the FLCs performance, this work is part of the specialist's commitment.

A fuzzy controller can contain dozens of pertinent functions and associated rules. The rules have an empirical character, since they are elaborated from the knowledge of a process specialist. This fact is especially important and makes the FLC different and advantageous compared to other controller architectures. The reason for this is that empirical rules are especially useful in plants that contain complex processes or even with insufficient information that would decrease the effectiveness of more conventional control techniques.

3.1 FLC designer

Similar to what happens in fuzzy logic applications in other areas, the FLC project requires several operational requirements that must be adequately dimensioned for the correct functioning of this type of control. An important choice as to how the final control signal will take place is the decision between the TSK or Mamdani FLC. The general rule model is given as:

$$\textbf{IF } x \textit{ is } A \textbf{ AND } y \textit{ is } B \textbf{ THEN } u \textit{ is } C, \tag{3}$$

where, A is a fuzzy set of X, B is one of the fuzzy set of Y and C is fuzzy set U (signal of controller).

In the case of the FLC of the Mamdani type, each rule is a conditional fuzzy proposition, and different fuzzy relationships in A x B x C can be derived from it. The implementation of each rule is done by defining operators to process the rule's antecedent and the implication function that will define its consequent. In this case, the action of the FLC is defined based on the aggregation of rules that make up the algorithm. This aggregation results in the fuzzy set C, which defines the output of controller C. The effective output of the controller is then obtained through a defuzing process applied to set C.

The TSK FLC is a simplified model of the Mamdani controller, where the consequent of each rule is defined as a function of the linguistic input variables.

Each rule results in no longer a numerical value but a fuzzy set. The weight assumes the pertinence value resulting from processing the rule's antecedent. The value of **u** can be defined as a constant (single value with relevance equal to one) - **singleton**. From a singleton it is possible define rules with output values that represent a classification of the controller response, without changing the simplified way of determining the final controller response.

In many practical situations of real controllers trapezoidal MFs are used for the function inputs. The reason is that several sensors installed in the system can present different noise in the measurement and as a consequence change some conditional rule in the operation of the FLC.

Regarding the design of the basic FLC rules, the expert needs to decide the total number of rules and how these operational conditions are related.

3.2 FLC optimization

The central idea of using metaheuristics in the pursuit of optimizing FLC performance is in the design characteristic of this controller. The various parameters and variables that are necessary for the good functioning of the FLC do not present general rules, making the role of the operator essential in the analysis and dimensioning of these terms [11].

Despite the specialist's knowledge, the resulting FLC may not perform as optimally. Metaheuristics, on the other hand, can function as an intelligent search engine for the various possible architectures for the FLC, without the need for the exhaustive work of trial and error on the part of an expert control engineer [11–13].

The use of a metaheuristic for the optimization of the FLC needs to be correctly dimensioned for its proper and effective functioning. There are different architectures to be explored when using these techniques despite an FLC [13]. **Figure 1** presents an FLC architecture using metaheuristics for its optimization.

From the observation of **Figure 1**, it is possible to notice that metaheuristics can help in the solutions of different areas of the FLC. There are several options for the parametric optimization of the FLC, therefore, the designer needs to define what one wants to obtain or which are the important parameters to be defined in his control project.

3.2.1 FLC membership function optimization

An application of FLC parametric optimization with metaheuristics is based on determining the positions and characteristics of MFs. In this case, algorithm

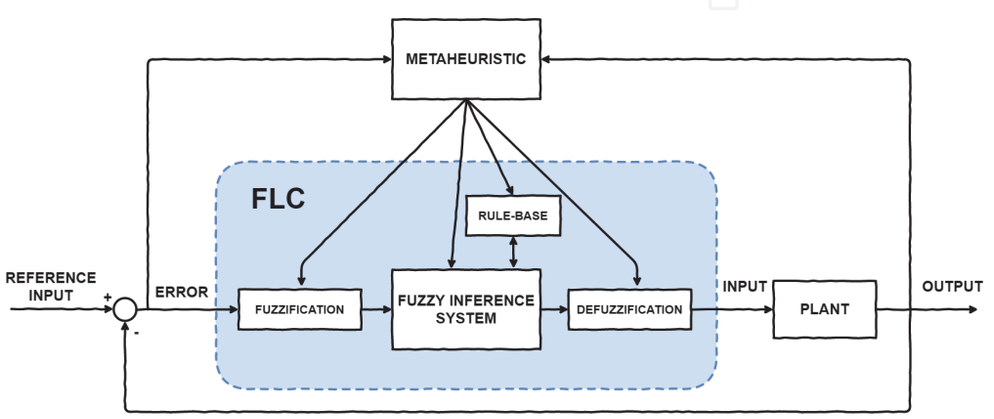

Figure 1.
FLC optimization scheme.

objective is to find the values of each MF individually that allows to obtain a better performance of this controller [14].

Assume that you have determined a set of rules and chosen all the MFs in your project. A common application of a metaheuristic algorithm in this case is to determine the ideal positioning of the MFs. In this particular context, the MF designer is a time-consuming exercise and it is usually possible to achieve good results. For this reason, some techniques offer benefits to develop such functions. In this case, it is necessary to turn the positions of each MF into optimization variables of the algorithm. **Figure 2** presents a representation of this application.

Figure 2 presents a possible model for the application of metaheuristic optimization of MFs. From a set of previously selected MFs, it is assumed that the positions of these functions are the variables (x_i) of the optimization problem. In this way, the algorithm seeks to select such parameters, increasing or decreasing the space of each one in the possible set. The position and size of each MF is important since this has a direct influence on the output of the FLC system.

In this FLC optimization application, it is necessary to define which MFs are involved and the respective definition limits. In practice, it is necessary to limit the range of values for each variable (x_i), thus avoiding that the optimization does not result in inadequate results.

3.2.2 FLC rules optimization

The optimization of the FLC rules is also an interesting application on this topic. Determining the rules of an FLC is naturally an empirical process, since trial and error are very necessary in order to be successful in these rules [15].

The application of metaheuristics in this case is configured to determine the best rules that, when combined in the logic of the FLC, optimize its operation. The variables in this situation are the defining set of MFs and can be combined in different ways to compose the rules. In this case, based on a previous set of information, it is possible to determine which variables will be part of a given rule and even how many rules should drive the FLC. **Figure 3** shows a metaheuristic scheme for the determination of a specific rule of the **IF-THEN** type.

Figure 3 shows an optimization of rules based on IF - THEN. From the set of MFs (NB, NS, NM, Z, PS, PM, PB), the subsequent terms $y_1(i)$ (de/dt) and $y_2(j)$ (e), as well as the consequent term $z(k)$ can be defined appropriately for the formation of FLC rules. In this example, i = 7 and j = 7, totaling 49 possible rules to be determined by metaheuristic optimization.

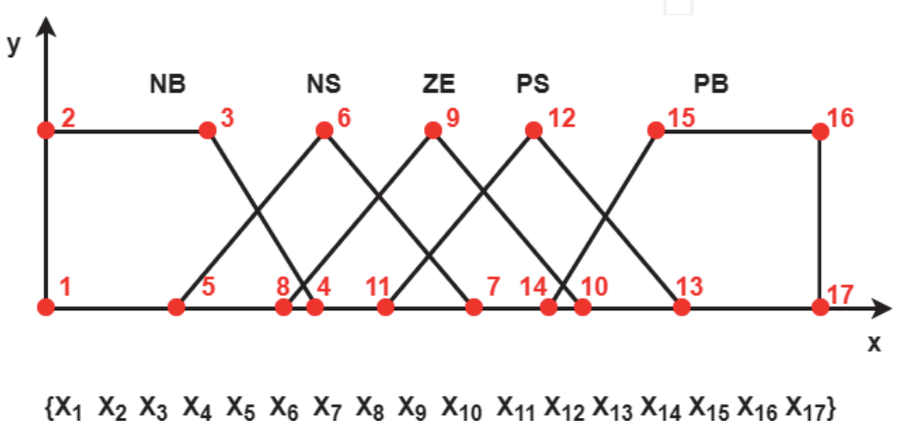

Figure 2.
MF optimization scheme.

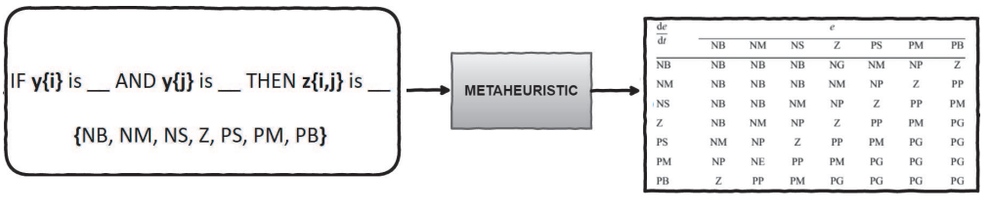

Figure 3.
Optimization scheme for fuzzy rule conditionals.

Metaheuristic algorithm tests all possible combinations of the IF-THEN condition simultaneously, thus allowing to speed up this process of determining the FLC rules.

In fact, there are several possibilities for the elaboration of FLC rules, which makes this problem complex for the use of traditional optimization methods.

4. Fitness function modeling

One or more performance requirements can be used to compose the performance index of the system, which will serve as the objective function (fitness) of the optimization problem used to obtain the tuning of the controller. The performance and robustness requirements act as restrictions on the tuning of the controller, that is, they allow imposing limits on the behavior of the control loop. The performance index to be useful to the problem must be composed of system parameters, in addition to that, it must be computed with some ease, analytically or experimentally [16].

In order to make the optimization functional, the definition of a fitness function is extremely necessary. This function aims to assess the quality and behavior of the control system, allowing the designer to check how well his project is performing. In general, dynamic performance quality indices are adopted as variables of an objective function in control systems.

The composition of the objective function is based on the dynamic characteristics that the operator wants the FLC to achieve in the optimization. This definition happens empirically based on the observation of the system and execution tests.

If the designer wishes the controller response to have a low overshooot value, this variable should be added with some gain in the composition of the fitness function. This gain depends on how much this variable is intended to influence the optimization goal. Moreover, if there is a desire to minimize some error rate in general, ITAE, ISE or IAE can be adopted in the function.

In such cases, the goal of metaheuristics is to minimize the objective function and, since this function is directly associated with a specific parameter, the specific objective can be achieved.

In order to demonstrate the construction of the objective function, consider the following general formulation

$$J = C_1X_1 + C_2X_2 + \ldots + C_nX_n, \tag{4}$$

where C_i represents the constants that are associated with the variables of interest X_i. The designer must determine how many variables one wishes to compose the function such that it achieves its final FLC performance objective. However, some variables in the function can be redundant and not affect the overall performance of the optimization. In this way, the designer's expertise is essential to obtain a function that allows an effective optimization of the FLC.

5. FLC optimization example

In order to present a practical application of FLC optimization via metaheuristic algorithm, this section presents an example of using a metaheuristic (PSO) for positional optimization of MFs in an FLC.

An air flow system is presented as a transfer function and an FLC is connected to it, which is parameterized manually and also by the PSO. Since the PSO algorithm is relatively simple and easy to apply, consider it for use in this application example.

The FLC is of the TSK type (control output is a specific value). In addition, the error and its derivative are considered as input to the system. The reason for this is that in addition to the error, it is also necessary to understand the speed and direction of the error variation.

In order to facilitate application, an example in MATLAB is used. **Figure 4** shows the FLC scheme associated with a metaheuristic.

From **Figure 4**, it can be seen that the FLC has the error and the error derivative as inputs. The error signal is used to calculate the ITAE and IAE indices. The FLC model has two outputs, one for the control signal and one for the rules. The control signal output passes through an integrating signal to compose the control signal. Moreover, the other output of the FLC is being used for the optimization of this controller. In this example five MFs are used (NB, NS, Z0, PS, PB) for the input error e (triangular), input error derived d/dt (triangular) and for the controller *output*. The optimization focuses on determining the position of the MFs (section 3.2.1) for the inputs and output.

First, the optimization focuses on the triangular functions, choosing one of the three positions that make up the triangle to carry out the modification. This is done by adding a variable X_1 and X_2 to this position while the others remain with a fixed value. Second, we seek to find the value of the output using the optimization variables X_3 and X_4. In addition, the rules have already been pre-established and are not targets for optimization. The source code for the definition of the FLC, including the MFs and the respective rules are presented in **Figure 5**.

Regarding the fitness function to be used in this example, the system uses the ITAE and IAE error indexes to formulate this function (**Figure 6**).

$$J = 0.5 * ITAE + 0.5 * IAE \tag{5}$$

For each new population resulting from the PSO swarm, the variables from the algorithm are stored and inserted in the FLC and, from that, the result of the fitness function (J) is estimated. This is done until reaching the stopping criterion. In this example, add the value of $J < 10^2$ as the main stop criterion for the algorithm. The code that presents the implementation of the objective function is shown below.

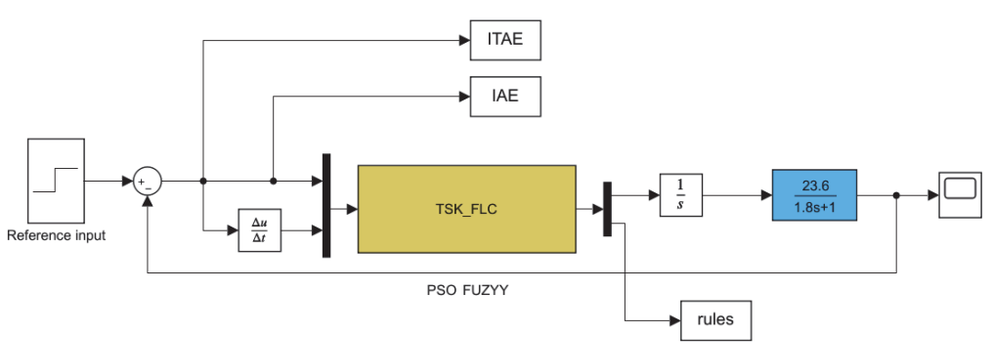

Figure 4.
FLC scheme for optimization by PSO in MATLAB.

```
function out=functions(input)
X=evalin('base','X');
warning off;
error=input(1);
d_error=input(2);
fuz=newfis('fuz','sugeno');        %Sugeno Type FLC

Sa1=[0.5,0.5];      %Input and Output membership functions are equallt distributed
Sb1=[0.5,0.5];      %Input and Output membership functions are equallt distributed
Sc1=[0.5];          %Input and Output membership functions are equallt distributed

fuz = addvar(fuz,'input','erro',X(1)*[-1 1]);
fuz = addvar(fuz,'input','d_erro',X(2)*[-1 1]);
fuz = addvar(fuz,'output','u1',[-1 1]);
% MF for input1
fuz = addmf(fuz,'input',1,'NB','trimf',[-1 -1     -Sa1(2)]);       %Five Triangular Membership functions
fuz = addmf(fuz,'input',1,'NS','trimf',X(1)*[-1 -Sa1(1) 0 ]);
fuz = addmf(fuz,'input',1,'ZO','trimf',X(1)*[-Sa1(1) 0   Sa1(1)]);
fuz = addmf(fuz,'input',1,'PS','trimf',X(1)*[0    Sa1(1) 1]);
fuz = addmf(fuz,'input',1,'PB','trimf',X(1)*[Sa1(2) 1     1   ]);
% MF for input2
fuz = addmf(fuz,'input',2,'NB','trimf',X(2)*[-1 -1     -Sb1(2)]);    %Five Triangular Membership functions
fuz = addmf(fuz,'input',2,'NS','trimf',X(2)*[-1 -Sb1(1) 0 ]);
fuz = addmf(fuz,'input',2,'ZO','trimf',X(2)*[-Sb1(1) 0   Sb1(1)]);
fuz = addmf(fuz,'input',2,'PS','trimf',X(2)*[0    Sb1(1) 1]);
fuz = addmf(fuz,'input',2,'PB','trimf',X(2)*[Sb1(2) 1     1   ]);
% MF for output1
fuz = addmf(fuz,'output',1,'NB','consXnt',[-1]);                    %Five Triangular Membership functions
fuz = addmf(fuz,'output',1,'NS','consXnt',[-Sc1(1)]);
fuz = addmf(fuz,'output',1,'ZO','consXnt',[0]);
fuz = addmf(fuz,'output',1,'PS','consXnt',[Sc1(1)]);
fuz = addmf(fuz,'output',1,'PB','consXnt',[1]);

matrix_rules =[ 1    1    1    1    1; 1    2    1    1    1; 1    3    1    1    1; 1    4    2    1    1;
                1    5    3    1    1; 2    1    1    1    1; 2    2    1    1    1; 2    3    2    1    1;
                2    4    3    1    1; 2    5    4    1    1; 3    1    1    1    1; 3    2    2    1    1;
                3    3    3    1    1; 3    4    4    1    1; 3    5    5    1    1; 4    1    2    1    1;
                4    2    3    1    1; 4    3    4    1    1; 4    4    5    1    1; 4    5    5    1    1;
                5    1    3    1    1; 5    2    4    1    1; 5    3    5    1    1; 5    4    5    1    1;
                5    4    5    1    1;];

fuz=addrule(fuz,matrix_rules);
out= evalfis([error d_error],fuz);
```

Figure 5.
FLC code in MATLAB.

```
function OptFUZZY =OptmiFUZZY(x1,x2,x3,x4)
%clear all
%close all

%Objective function calculation - partially calculated inside of the model

assignin('base', 'X', [ x1 x2 x3 x4]);

sim('Fuzzy_PSO.mdl');
OptFUZZY =0.5*max((Sys_Out1.Data(end)))+0.5*max((Sys_Out2.Data(end))); %ITAE + IAE


end
```

Figure 6.
Fitness function code in MATLAB.

For the application of the PSO, it is first necessary to initialize the algorithm parameters. In this case, initially the value of the particle positions in the swarm is randomly defined. The position of all particles is usually started with some speed, in this example all speeds are started at zero. Finally, it is defined as a minimum value of the best initial particle in 1000. The particles in the swarm must reach and exceed at least this value of the "best initial particle". **Figure 7** shows the code used.

The operation of the PSO is simple, the population is evaluated and its position is updated based on its position and previous speeds. The swarm positions are then entered in the *X* variables, which in turn are the inputs for the "*OptFuzzy*" optimization function. The *X* values are actually the values that will adjust the positions of the FLC's input MFs. If the value obtained in the optimization is less than the current value of the objective function, the best individual particle values and the best global value are increased. This is done until reaching the stopping criterion. **Figure 8** shows the PSO operation code in this FLC optimization example.

In order to test the PSO in the optimization of the FLC MFs of the present example, we try to test the algorithm with 10, 20, 50, 100, 200 and 500 particles in the swarm. **Figure 9** shows the behavior of the objective function "*OptFuzzy*" over 50 iterations.

In general, the greater the amount of swarms in the PSO, the faster the minimization of the objective function will occur, since this way there will be a greater exploration of the search space of the algorithm.

```
%% Particle Swarm Optimization Simulation

iterations = 5000;
inertia = 1.0;
correction_factor = 1.0;
swarm_size =500;

% ---- initial swarm position -----
%index = 1;
for index = 1 : iterations

        swarm(index, 1, 1) = randi([0,10]);
        swarm(index, 1, 2) = randi([0,10]);
        swarm(index, 1, 3) = randi([0,10]);
        swarm(index, 1, 4) = randi([0,10]);
        %index = index + 1;
end

swarm(:, 4, 1) = 1000;          % best value so far
swarm(:, 2, :) = 0;             % initial velocity
```

Figure 7.
PSO parameters.

```
%% Iterations
for iter = 1 : iterations

    %-- evaluating position & quality ---
    for i = 1 : swarm_size
        swarm(i, 1, 1) = swarm(i, 1, 1) + swarm(i, 2, 1)/1.3;     %update x1 position
        swarm(i, 1, 2) = swarm(i, 1, 2) + swarm(i, 2, 2)/1.3;     %update x2 position
        swarm(i, 1, 3) = swarm(i, 1, 3) + swarm(i, 2, 3)/1.3;     %update x3 position
        swarm(i, 1, 4) = swarm(i, 1, 4) + swarm(i, 2, 4)/1.3;     %update x4 position

        x1 = swarm(i, 1, 1);
        x2 = swarm(i, 1, 2);
        x3 = swarm(i, 1, 3);
        x4 = swarm(i, 1, 4);

        value = OptFUZZY(x1,x2,x3,x4);
        %(x1,x2,x3,x4);

        %OptFUZZY =0.5*max((Sys_Out1.Data(end)))+0.5*max((Sys_Out2.Data(end)))
        % fitness evaluation (you may replace this objective function with any function having a global minima)
        Fitness = val;
        if val < swarm(i, 4, 1)                % if new position is X2_ter (fitness value of this particle)
            swarm(i, 3, 1) = swarm(i, 1, 1);   % update best x,
            swarm(i, 3, 2) = swarm(i, 1, 2);   % best y postions
            swarm(i, 3, 3) = swarm(i, 1, 3);
            swarm(i, 3, 4) = swarm(i, 1, 4);
            swarm(i, 4, 1) = val;              % and best value
        end
    end
```

Figure 8.
PSO code in MATLAB.

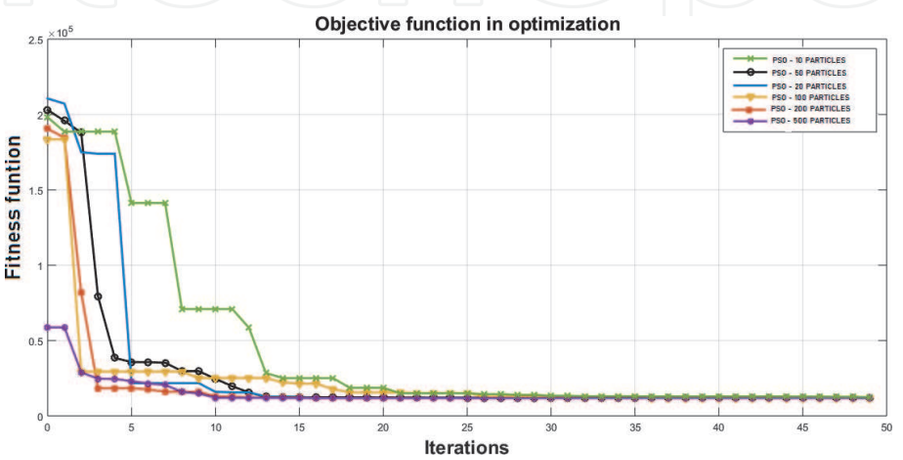

Figure 9.
Fitness function optimization.

In addition to optimizing the FLC's input (*e* and *de/dt*) and output via PSO, manual tuning is also applied to these MFs. The manual tuning basically consists of determining "*manually*" the positional values of the triangular functions of the MFs

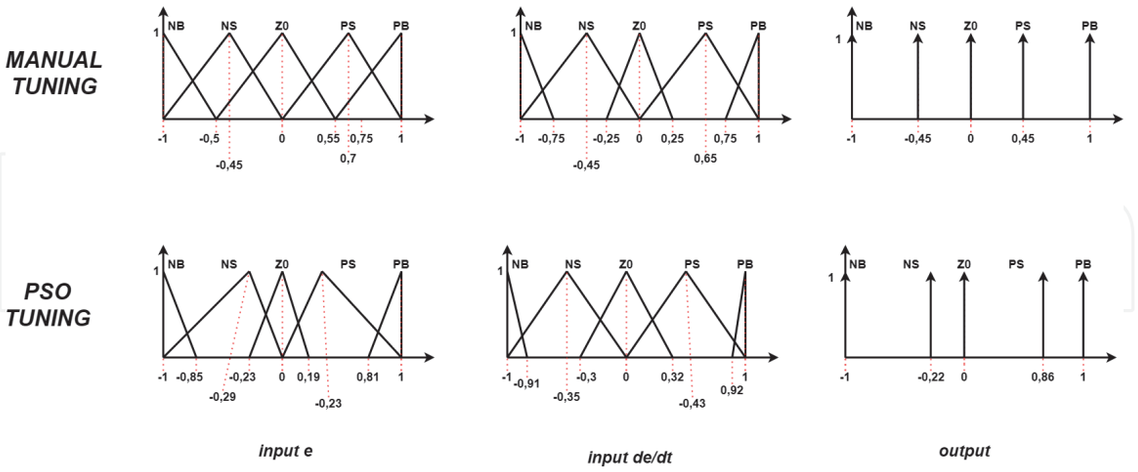

Figure 10.
MFs manual tuning vs. MFs PSO tuning.

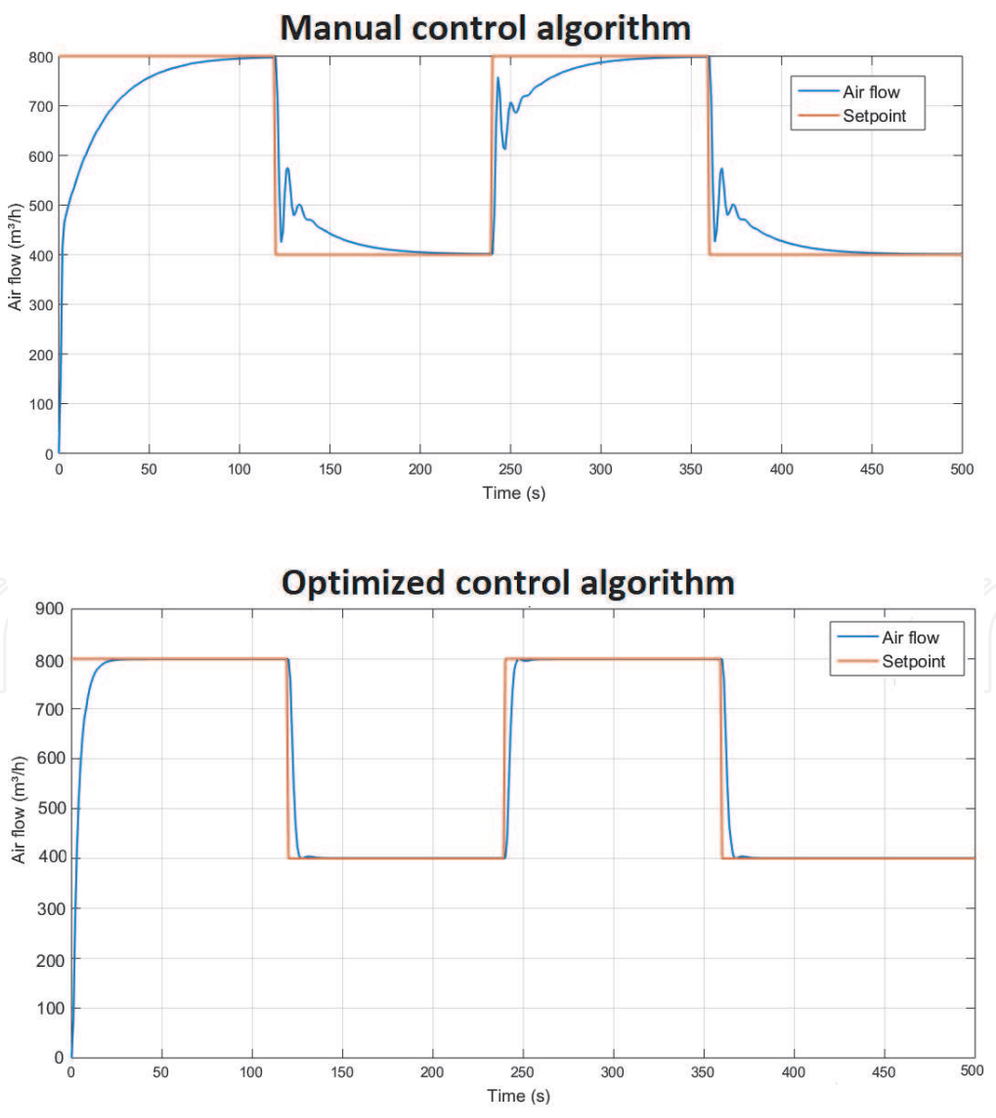

Figure 11.
FLC manual tuning vs. FLC PSO tuning.

FLC Tunning	Rise Time (s)	Settling Time (s)
Manual	33,94	72,10
Metaheurístic	3,56	6,99

Table 1.
Comparison between the different tunings of the FLC.

and the value of the MFs at the output. This choice is empirical and is based on the operator's knowledge and experience regarding the control system. The result of the MFs tuning process in different ways is shown in **Figure 10**.

From the MFs resulting from the tuning processes discussed above (**Figure 4**), the FLC controller is simulated in both tuning situations (Manual and PSO) by applying different steps to the control system input. **Figure 11** shows the result of the behavior of the FLC resulting from the different synotnias mentioned above.

From **Figure 11** it is possible to notice that the FLC obtained from the PSO presented a much more satisfactory performance in terms of dynamic behavior. **Table 1** presents detailed values of the performance indices for both processes.

6. Conclusions

The FLC is an important controller in the area of intelligent systems of control engineering, however its design aspects represent a challenge to the specialist, as its operational requirements require a characteristic knowledge based on linguistic resources.

An alternative to the development of the FLC is the use of optimization resources to aid the development of this type of controller. Metaheuristics can clearly contribute to this issue, as they represent an effective way of exploring solutions to complex problems. The use of this feature involves the definition of an optimization problem, that is, the definition of a fitness function and which variables are associated in the process.

An FLC optimized by a metaheuristic algorithm represents a viable alternative for several applications, as the solutions can be adapted to the optimization criteria of these algorithms, thus allowing the specialist to develop an FLC to meet their specific performance needs.

Abbreviations

FLC fuzzy logic control
MF membership function
PSO particle swarm optimization
GA genetic algorithm
TSK Takagi–Sugeno–Kang

Author details

Jeydson Lopes da Silva†
Federal University of Pernambuco, Recife, Brazil

*Address all correspondence to: jeydson.lopes@ufpe.br

† These authors contributed equally.

References

[1] Chuen L. Fuzzy logic in control systems: fuzzy logic controller. I. IEEE Transactions on Systems, Man, and Cybernetics, vol. 20, no. 2, pp. 404–418. DOI: 10.1109/21.52551

[2] Yahaya, S Z, Hussain Z, Boudville R, Ahmad F, Taib M N. Optimization of FLC parameters for optimal control of FES-assisted elliptical stepping exercise using GA and PSO. 2014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2014), Batu Ferringhi, 2014, pp. 663–667. DOI: 10.1109/ICCSCE.2014.7072801

[3] Ronaldo A, Luis C, Davidson M, Jeydson S, Camila B, Andre A. An Emotional Controller PLC Implementation for an Industrial Fan System. Neural Networks (IJCNN), 2016 International Joint Conference on, 2016.

[4] Arrofiq M, Saad N. A PLC-based Self-tuning PI-Fuzzy Controller for Linear and Non-linear Drives Control. 2nd IEEE International Conference on Power and Energy (PECon 08), Malaysia, 2008.

[5] Kennedy J, Eberhart R. Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks, p. 1942–1948, 1995.

[6] Saptarshi S, Sanchita B, Richard A P. Particle Swarm Optimization: A Survey of Historical and Recent Developments with Hybridization Perspectives. Journal of Machine Learning and Knowledge Extraction. DOI: 10.3390/make1010010.

[7] Juan L F M. A Brief Historical Review of Particle Swarm Optimization (PSO). Journal of Bioinformatics and Intelligent Control 1(1):3–16 DOI: 10.1166/jbic.2012.1002.

[8] C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controller. I," in IEEE Transactions on Systems, Man, and Cybernetics, vol. 20, no. 2, pp. 404–418, March–April 1990, doi: 10.1109/21.52551.

[9] Zhou, et al.Optimized Fuzzy Logic Control Strategy for Parallel Hybrid Electric Vehicle Based on Genetic Algorithm. Applied Mechanics and Materials, 274, 345–349. https://doi.org/10.4028/www.scientific.net/amm.274.345

[10] P. Rajakumar, R. Saravanakumar. Design of Fuzzy Logic Controller for DSTATCOM. Advanced Materials Research, vol. 905, Trans Tech Publications, Ltd., Apr. 2014, pp. 401–405. Crossref, doi:10.4028/www.scientific.net/amr.905.401.

[11] Lamkhade P, Parvat B, Kadu C. Design and implementation of fuzzy logic controller for level control. 2015 International Conference on Energy Systems and Applications, Pune, 2015, pp. 475–479, DOI: 10.1109/ICESA.2015.7503395.

[12] Katbab A. Fuzzy logic and controller design-a review. Proceedings IEEE Southeastcon '95. Visualize the Future, Raleigh, NC, USA, 1995, pp. 443–449. DOI: 10.1109/SECON.1995.513133.

[13] Shahraki A, Shahraki M, Mosavi M. Design and simulation of a fuzzy controller for a busy intersection. 2013 International Conference on Computer Applications Technology (ICCAT), Sousse, 2013, pp. 1–6. DOI: 10.1109/ICCAT.2013.6521986.

[14] Rahma A, Khemliche M. Combined approach between FLC and PSO to find the best MFs to improve the performance of PV system. 2014 International Conference on Electrical Sciences and Technologies in Maghreb (CISTEM). doi:10.1109/cistem.2014.7077038

[15] Li P, Du X. A GA Optimization for FLC with Its Rule Base and Scaling Factors Adjustment. In: Huang DS., Li K., Irwin G.W. (eds) Computational Intelligence. ICIC 2006. Lecture Notes in Computer Science, vol 4114. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-37275-2_1

[16] Ogata K. Modern Control Engineering, Prentice-Hall, 2003. , Prentice-Hall, 2003.