EVOLUTIONARY SEARCH FOR OPTIMIZATION OF FUZZY LOGIC CONTROLLERS

Rogério Neves, Marcio Lobo Netto

Polytechnic School, University of São Paulo

ABSTRACT

The specification of Fuzzy Logic Controllers (FLC) generally requires a specialist or close collaboration with a specialist who holds the linguistic operation rules of the system. Many times perhaps the rules may be not expressed efficiently in formal means, due to complexity or excessive number of variations and/or combinations involved, but sometimes there may be no specialist at all for the system in study. For these cases, we suggest an autonomous method of definition and optimization of Fuzzy Logic Controllers, making use of heuristics, concepts of Evolutionary Search, Genetic Algorithms and Multi-Agents in determination and optimization of parameters for an automated control system.

1. INTRODUCTION

Since the introduction of Fuzzy Logic by Zadeh [1], many attempts have been made in order to apply the uncertainly associated with human thinking, expressed by natural language, to automated control systems. From the most fruitful, Fuzzy Logic Controllers (FLC) and Fuzzy Algorithms [2][3] are largely applied in the specification of automated control systems for early human-controlled systems. As long nothing is perfect, we still need a specialist to define verbally how to manually operate the system. Sometimes, perhaps the specialist may not be available or even exist for the system in study.

We suggest here an autonomous method of definition and optimization of Fuzzy Logic Controllers, by applying concepts of Artificial Life, Evolutionary Search/Genetic Algorithms and Multi-Agents. The method consists in express the system in terms of input, output and rule parameters, using computer simulations to search and refine the parameters for optimum values of objective functions, making use of the massive processing capability of modern computers to test and qualitatively evaluate the parameters for multiple instances at once. The method can be divided in two main steps:

- System study: the relevant system attributes are identified; input and output parameters are defined;
- Computer simulations: the system is simulated; the parameters are evaluated and optimized by genetic algorithms.

To demonstrate the method a simple case is presented, where the input and output parameters are clearly recognizable and the connections can be intuitively defined for analogies with the autonomous method. The method is described as the problem is proposed in section 2, the experiment conduction is presented in section 3, the results obtained are presented in section 4 and final conclusions are made in section 5.

2. PROBLEM OUTLINE

Supposing that the system we wish to control is an autonomous robot. For now, the only knowledge required about the system is the input and output parameters. Studying the system we may find the input variables associated with available sensors, and output variables depending on the control structure involved (switches, step-motors or power regulators, etc.).

Once the system outline is clear, it’s time to define the purpose of the robot, which will model the fitness function and simulation rules. Supposing it will be designed to be a resource collector, ignoring details such as resource kind, terrain type, operating conditions, it does basically identify certain kinds of objects, rotate, move to, stop and collect. The main operations will be defined in the simulation as functions, which will accept parameters and return some feedback. So, for this experiment, the functions will be defined as follows:

- TRACK: Read scene, expressing objects in terms of linguistic variables.
- ROTATE (angle): Rotate angle in degrees.
- MOVE (speed): Move with desired speed.
- CATCH: Call routine to catch the object, returns success or fail.

In each program cycle (instant), the possible operations are: track, move and rotate or track and catch.

2.1. Tracking

When called the function performs calculations in the scene, locating objects and expressing them in terms of linguistic variables (representing sensor readings). Real sensors may have limitations in range, so let's define max-angle and max-distance bounding the covered area.

As input parameters, the linguistic variables *D* and *L* represents respectively distance and lateral displacement of the object relative to the robot's position, the variables gives pertinence values to fuzzy sets of all objects in the robot's view range (between -max-angle/2 and +max-angle/2 with distance < max-distance). The definition of fuzzy sets must comply with sensor's reliability, considering factors such resolution, error level and other physical limitations. Here we will minimize the number of fuzzy sets, describing by five sets the displacement and by four sets the distance.

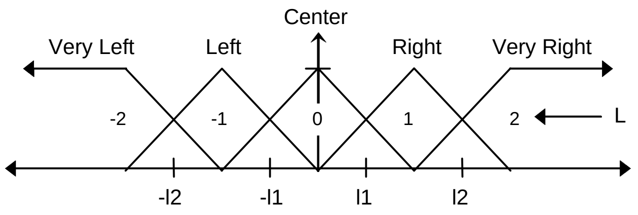

Figure 1: Fuzzy sets belonging to the linguistic variable lateral displacement (L), the variables l1 and l2 demarks the crossover points and are used to describe de distribution.

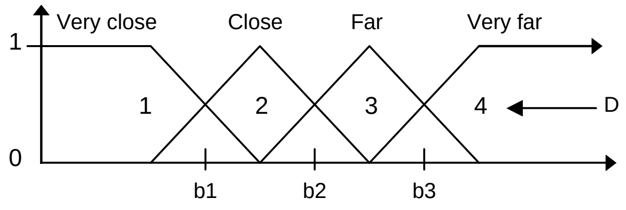

Figure 2: Fuzzy sets in the linguistic variable distance (D), the variables b1, b2 and b3 here demarks the crossover points.

The function needs to return only the pair of variables (*D*, *L*) associated with the closer object in range, or notify in case nothing was detected (no objects in range). If we are dealing with crisp controls, which need precise values to operate, this is the time to select the defusification strategy to apply. Some applicants are: Max criterion, mean of maximum method, center of area, center of mass, or greater pertinence [3].

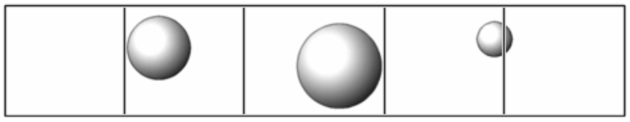

Figure 3: Objects in the scene described in Table I.

Table I: Description of the scene in Figure 3

Object	Distance	Lateral displacement
1 (closest)	0.36/1, 0.64/2	0.86/0, 0.14/1
2	0.94/2, 0.06/3	0.75/-1, 0.25/-2
3	0.29/3, 0.71/4	0.56/1, 0.44/2

2.2. Moving and rotating

The output refers to the system's control capabilities. Here will be imposed that the control assumes only "crisp" values, in case, only certain values of Speed and Angle are allowed, just like in a control panel, where switches (or buttons) activates the actions performed.

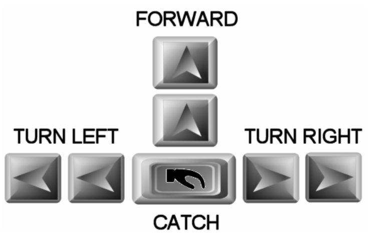

Figure 4: The "Crisp" control panel

We may denote that only certain combinations are allowed in the control proposed, and no selection implies the robot stand still. The experiment algorithm shall avoid inconsistent combinations.

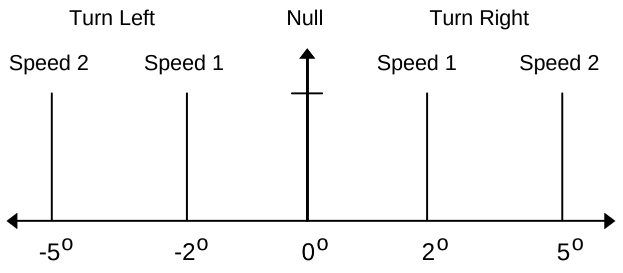

Figure 5: Representing the possible outputs for "rotate" function as Fuzzy Singletons.

2.3. Catching

Catching executes the call to the capturing mechanism (ex. Suction pipe, mechanic arm, etc.), which supposes that the object is in the right place at time called. Once activated, the function executes the call and returns true or false, according to catch success.

The function checks if the object is in the right position relative to the robot and below a specified tolerance threshold, if so, it executes the exclusion of the object from the scene adding to the robot's score. Softening the threshold in the early generations, hardening during experiment until the desired tolerance is reached, grants a faster adaptation of the breed, severally reducing the evolution time.

The provided feedback influence decisions in the next instants (ex. If FAIL then retry, leave or move a bit and retry, etc.), in fact, feedback is an important part of the process, increasing the system's intelligence as it knows more about what's happening around.

3. EVOLVING THE EXPERIMENT

According to what was presented, the main problem is to map a multi-dimensional space of inputs into another multi-dimensional space of outputs. This can be done using functions, connections or inference rules. Dealing with fuzzy logic, is natural to express actions using inference rules, but many times, combination of strategies may present better results. The fact is that all the possible strategies must be available to the algorithm to choose, first randomly, and then improved by genetic algorithms, changed, tried and discarded, until an efficient strategy is found. The inference rules have the form:

IF D is FAR and L is CENTER then MOVE FAST
IF D is CLOSE and L is LEFT then TURN LEFT
IF D is VERY CLOSE and L is CENTER then CATCH

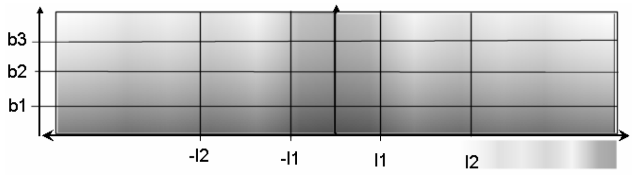

Figure 6: The two-dimensional input variable space, assuming symmetry for lateral displacement.

From scratch, we don't know any of the rules, just the combinatory explosion of the inputs and outputs. Excluding the forbidden combinations, the remaining will be available to the instances "creation" algorithm.

3.1. Creating instances

Each robot is described by a genetic code or, for simplicity, just "DNA", which is nothing more than a string containing all the robot's specific parameters including input/output rule connections for the inference rules and delimiters l1, l2, b1, b2, b3. Simulation parameters such position, direction vectors, robot's energy, state variables, and number of captured targets can also be stored into the string, despite it classically shouldn't belong to the DNA, it will simplify the message passing to functions, once the string contain all the information about the robot. Here, the DNA represents the string containing the robot's individualistic parameters.

In the beginning of the simulation, a number *n* of robots are created with random parameters and put together into a virtual arena, which contains *m* target objects scattered all over. The parameters for the robots must be draw from allowed values, and the DNA tested to exclude invalid combinations of parameters in order to prevent creation of useless and not-working units. Some previous test concerning the generated DNA's functionality may severally reduce the experiment time. In the start of the simulation, control parameters are reset, setting all robots initial energy to *Ei*, zeroing their captured targets and placing them randomly into the arena with random direction vectors.

3.2. Evaluating

The simulation accuracy is an important matter concerning the system efficiency, and must express the dynamics of the system, such physical conditions and competition rules, in order to evaluate the parameters. To simplify the implementation, second order effects can be ignored in the simulation (such friction, shape, etc.), but more details about the real operating conditions imply best fit of found parameters in the physical model.

The objective functions (represented by the fitness expression) may represent the desired result of the evolution process. In the case, we must want to reward robots that score more objects in less time, spending the minimal amount of energy. So, the expression for the objective function can be stated as:

$$Satisfaction = \frac{objectsScored}{(Ei - Ef).timeElapsed}$$

Expressing the degree of satisfaction in terms of captured objects, initial energy *Ei*, final energy *Ef* and the time elapsed in the process. Some considerations shall be made concerning the simulation dynamics:

- Energy consumption: Movements, rotations and call to internal function (such catch) consumes variable amount of energy, which need not to express the real consumption, but the desired behavior, imposing energy losses for unwanted operations. Once the available energy is exhausted the robot is no longer processed and its elapsed time is stopped.

- Time penalties: Robot's call for functions must be succeeded by a period of inactivity, in which the robot will wait for some action to be performed. Once again, objectives can be expressed by time penalties, implying large time losses for successive ineffective or undesired calls.
- Damage control: Actions that may result in some damage or hazardous movements (such collisions and bumps) shall be penalized with energy loss, inactive time, or by inserting a proper "damage" variable into the satisfaction function for further evaluation.

Several stop criteria must be set in order to minimize the simulation round time, for the experiment, the chosen stop conditions are:

- Timeout: A maximum experiment time is reached;
- Resources over: All objects captured;
- Energy shortage: All robots stopped due lack of energy.

3.3. Evolution

Once one of the stop conditions is reached, the round is ended and the time to express the selection rules has come, like in natural selection, the most adapted will survive, some will evolve and many will decease. The first step is to generate a list of instances ordered by greater *Satisfaction* value, selecting the instances that will be kept from the top of the list.

There are many evolutionary strategies than can be used to evolve the selected instances, such mutation, cross-over, reproducing bests, all are found in referred literatures [4][5][6]. The evolution strategy determines how some instance can possibly achieve the optimal value, some strategies may take longer and some may never escape from local maximums, but best results can be reached using a combination of several strategies.

One possible evolution strategy, supposing we have initially hundred instances, for the best twelve:

- Save the first place, identifying in the name the fitness value. This will grant the possibility to recur to some previous solution with better score in case it is needed;
- Keep the best ten unchanged in the next round, plus, reproduce them with mutation factor of 1% to 10%. This will give the chance to generate better instances. Lower mutation rates are recommended in this case;
- Apply mutation to the remaining instances, granting the chance to improve its performance. Higher mutation rates (5% to 20%) are recommended here;
- Cross-over instances can be generated from the top selected instances, combining the DNA from successful pairs randomly, granting the chance to escape local maximums;
- New instances can be generated to fill the required number for a new simulation, giving the chance to reach off local solutions in space;
- Combination of cross-over and mutation can be performed. Greater the number of instances in the simulation, greater the chance to reach the global maximum for the satisfaction expression.

Once all the new instances are generated, the arena must be rebuilt, robots randomly distributed with states reset and objects reposed, so the simulation restarts.

4. RESULTS

In the performed experiment, for 400 generations, a value close to the experiment's maximum was achieved in the first 100 generations, and the maximum in around 300 generations. The competition gets more challenging as the instances generated are better adapted, causing the simulation time to decrease as the resources are consumed faster by specialized breeds and sometimes the overall satisfaction falls slightly due to more efficient instances competing for the available resources. Results and animations of the experiment can be found in [7].

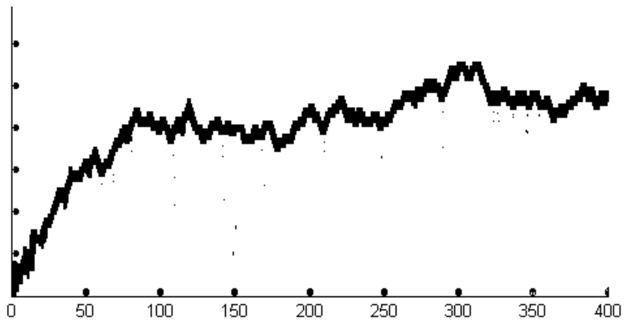

Figure 7: Qualitative analysis plot. Satisfaction value for the best 10 from100 instances in 400 generations

Better instances are progressively generated as the experiment evolves indicating that at some time the parameters may converge to some optimal value, but nothing grants that the global maximum for the *Satisfaction* will be ever achieved (except for exhaustive search, where all combinations are tested, when experiment time is very long or in cases where analytic methods can be applied, leading to specific points in the space of solutions), the only guarantee is that the chance to achieve a maximum value increases slightly with the number of generations evolved.

5. CONCLUSIONS

The described experiment demonstrates that the method generates good solutions for the problem in study. The method can be extended to systems where the control process is unknown, as well as to systems where the control process is to complex for intuitive deduction, where rules involve a large number of implications, input and output parameters.

Once resolved, the optimal parameters can be expressed in formal rules, algorithms or fuzzy algorithms to be programmed into an ordinary microprocessor and inserted into the hardware to perform the specific task.

6. REFERENCES

[1] Zadeh, Lotfi A., *Outline of a New Approach to the Analysis of Complex Systems and Decision Processes*, IEEE Transactions on Systems, Man, and Cybernetics, volume SMC3, no 1, January 1973.

[2] Lee, Chuen Chien, *Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I*, IEEE Transactions on Systems, Man, and Cybernetics, volume 20, no 2, March/ April 1990.

[3] Lee, Chuen Chien, *Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part II*, IEEE Transactions on Systems, Man, and Cybernetics, volume 20, no 2, March / April 1990.

[4] Bentley, P. J. (Editor), *Evolutionary Design by Computers*, Morgan Kauffmann ISBN: 1-55860-605-X (1999)

[5] Michalewic, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag ISBN: 3-540-60676-9 (1996)

[6] Mitchell, M., *An Introduction to Genetic Algorithms,* MIT Press ISBN: 0-262-13316-4 (1997)

[7] Neves, Rogério *(rponeves@lsi.usp.br), home page http://www.lsi.usp.br/~rponeves/work/robot*