Third International Conference
Modelling and Development of Intelligent Systems
October 10 - 12, 2013
"Lucian Blaga" University
Sibiu - Romania

A Better Genetic Representation of a Fuzzy Controller Enabling the Determination of Its Parameters with the Help of a Genetic Algorithm

Stelian Ciurea

Abstract

Since 1975, fuzzy controllers have fully proved their usefulness in the most diverse applications. The design of such a controller involves setting up inference rules and values for a large number of parameters. There are situations where this is possible either through the expertise of a human operator or through a knowledge stock. If we cannot rely on such information, genetic algorithms are a good alternative to determine these values. The first condition in solving a problem by means of a genetic algorithm is the genetic representation of the solution. In this paper, we present the genetic algorithm that we designed with a view to determining the parameters of a fuzzy controller for the Truck Backer-Upper Problem (this problem is considered an acknowledged benchmark in nonlinear system identification). The genetic representation used in this algorithm belongs to us.

1 The Fuzzy Controller for the Truck Backer-upper Problem

1.1 The Truck Backer-Upper Problem

This problem, made famous by [3], has been investigated by many researchers. On the other hand, it is difficult not to notice that nearly anyone is able to drive the truck to the desired position if given some time to get used to the controls.

The truck corresponds to the cab part of the Nguyen-Widrow's truck and trailer, referred to as the simplified Nguyen-Widrow problem. The truck position is determined by the three state variables $x \in [-50, 50]$, $y \in [0, 80]$ and $\varphi \in [-90°, 270°]$ - the angle between the truck's onward direction and the x-axis (Fig. 1). The width and length of the truck are 5 and 2 meters, respectively.

The truck sets out from an initial position with the three state variables xi, yi and φi and must reach the loading dock with $xf = 0$, $yf = 0$, $\varphi f = 90°$. The truck only moves backwards with the fixed speed. To control the truck at every stage, an appropriate steering angle $\theta \in [-45°,45°]$ must be provided. Thus, the controller is a function of state variables $\theta = f(x, y, \varphi)$.

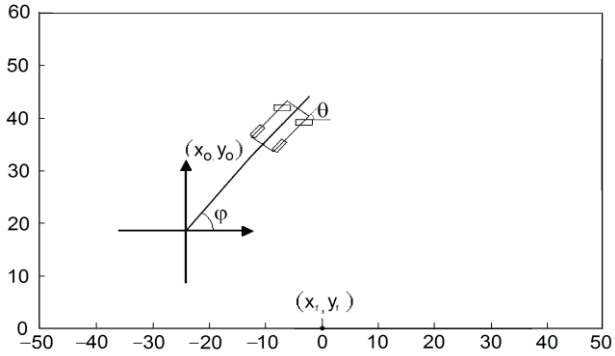

Figure 1. Truck backer-upper system

Typically, it is assumed that there is enough clearance between the truck and the loading dock so that the truck position coordinate y can be ignored, simplifying the controller function to: $\theta = f(x, \varphi)$. For obvious reasons, such a controller does not perform very well if the distance between the truck position and the loading dock is small.
The movements of the truck are described by the following system of equations:

$$\begin{cases} \dot{x} = -v\cos\varphi \\ \dot{y} = -v\sin\varphi \\ \dot{\varphi} = -\frac{v}{l}\tan\theta \end{cases} \quad (1)$$

where l is the length of the truck and v is the backing up speed of the truck. These equations are applied to the current state, and the truck moves on until one of the following stopping conditions is met:

- $y \leq 0$ (the truck reached the loading dock);
- x, y or φ have an unacceptable value: $y>100$, $x \notin [-50, 50]$ or $\varphi \notin [-90°, 270°]$

1.2 The Fuzzy Controller

We implemented a Mamdani-type fuzzy controller. The input data are *x* and φ, and the output data is the steering angle θ. For *x*, we have opted for 5 fuzzy sets with the following linguistic variables: left - LE, left center - LC, center - CE, right center- RC, and right – RI. For φ, we have settled on 7 sets: RB (right below), RU (right upper), RV (right vertical), VE (vertical), LV (left vertical), LU (left upper), and LB (left below). For θ, we have also selected 7 sets: NL (negative large), NM (negative medium), NS (negative small), ZE (zero), PS (positive small), PM (positive medium), and PL (positive large). The membership functions we have employed are trapezoidal or triangular (Fig. 2, Fig. 3 and Fig. 4).

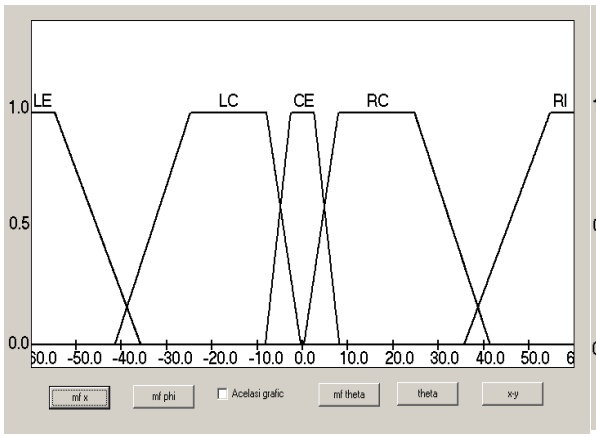

Figure 2. Example of trapeziodal for variable *x*.

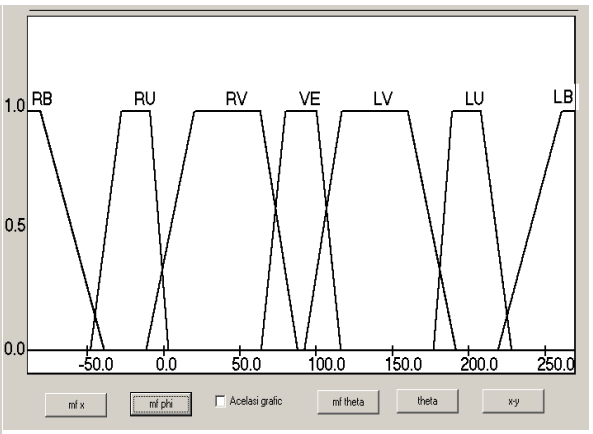

Figure 3. Example of trapezoidal membership functions for variable φ.

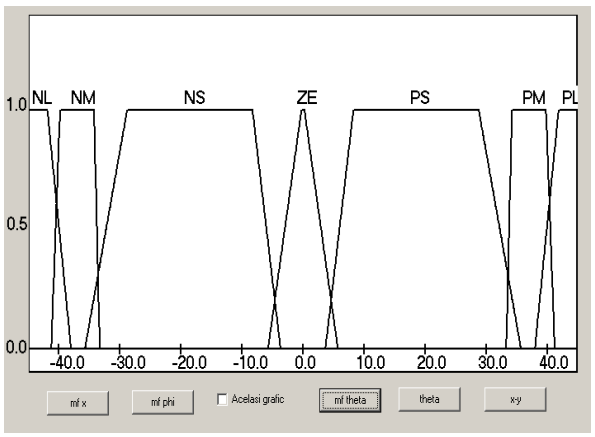

Figure 4. Example of trapezoidal and triangular membership functions for θ.

Starting from these sets, a fuzzy controller needs 5×7=35 inference rules based on fuzzy arguments. For example, if the *x* position is right centre, and the angle φ is vertical, then we want to steer positive medium. Symbolically, IF x is RC AND φ is VE, THEN θ is PM. Table I illustrates a possibility of defining these 35 rules.

Table I: Matrix of the Rules for the control of the Truck Backer-Upper System:

φ / *x*	**RB**	**RU**	**RV**	**VE**	**LV**	**LU**	**LB**
LE	NL	NL	NL	NM	NM	NS	PS
LC	NL	NL	NM	NM	NS	PS	PM
CE	NM	NM	NS	ZE	PS	PM	PM
RC	NM	NS	PS	PM	PM	PL	PL
RI	NS	PS	PM	PM	PL	PL	PL

For the two input data variables – *x* and φ, the membership function is calculated for each of the 35 rules; we group the two results of these memberships function by means of an AND fuzzy operation, and then by means of an implication operation we obtain fuzzy sets for θ. We will have 35 θ sets corresponding to the 35 rules. These are grouped by means of the aggregation operation

to get the output θ-set. Then, to find the actual control value, we must convert the output fuzzy set into a numerical value for θ by means of the defuzzification operation. There are various formulae for the fuzzy AND, implication, aggregation and defuzzification operations. It follows that, in order to fully define fuzzy controller, we need 35 inference rules and parameters that define the type and positioning of the fuzzy functions on the universe of discourse axis for the three variables (x, φ, θ), as well as the implementation of the fuzzy operations that occur in the calculating the response of the controller. There are no mathematical formulae to provide the values for these parameters.

3 The Genetic Algorithm

Genetic algorithms belong to the category of probabilistic algorithms. Such an algorithm starts from a set (population) of possible solutions (chromosomes). The performance of each chromosome is calculated by means of an evaluation function which appraises the accuracy of the solution provided by that chromosome to the studied problem. The new population is formed by selecting the fitter individuals. Some members of the new population recombine by means of "genetic" operators to form new solutions. There are unary transformations like mutations, which create new individuals by a small change in single individual and binary transformations, such as the crossover, which create new individuals by mixing traits from the two parents. After a number of iterations (generations) the search converges and is successful if the best individual obtained at a given time represents the optimum solution.

3.1 Genetic representation of the controller

The purpose of the genetic algorithm we have designed is to determine the parameters of the fuzzy controller optimal for solving the truck backer-upper problem. Because of the intrinsic symmetry of the problem, we have selected the member functions that are symmetrical to the median axis of the universe of discourse for each of the three variables. Thus, in a chromosome, for x, we have retained parameters for variables CE, RC and RI; variable LE mirrors RI symmetrical to axis x=0 and LC mirrors RC. Similarly, for φ we have represented parameters for variables VE, LV, LU and LB, and for θ, variables ZE, PS, PM and PL.
We coded the following parameters that typify a fuzzy controller:

- The interference matrix as a matrix with 5 rows and 7 columns, where each item can range between 1 and 7 corresponding to the 35 interference rules (1 means NL, 2 means NM, etc.);
- the type of fuzzy operations through 5 integer numbers for the type of fuzzy operations AND (0=min, 1=product), OR (0=max, 1=a+b-ab), the type of implication operation (0=min, 1=product), the type of the aggregation operation (0=max, 1=sum, 2=a+b-ab) and the type used in the defuzzification operation: a value between 0 and 3 corresponds to the methods based on integral calculus, while values between 4 and 6 are for elitist methods;
- The shape and position of the fuzzy sets in the universe of discourse. We point out that we chose the trapezoidal for the trapezoidal membership functions corresponding to these sets (we assumed that the triangles are special cases of trapeziums with smaller bases of negligible length).

3.1.1 Representing classical trapezoidal membership functions

With the release of MATLAB 6, the "Fuzzy Logic Toolbox" library has been implemented [14]. Within it, a number of membership functions were defined for fuzzy sets, among which trapezoidal and triangular-shaped ones. Due to the popularity of MATLAB, the method by which

fuzzy sets are represented in this software has become traditional, and most papers determining the parameters of fuzzy controllers by means of genetic algorithms use it. Thus, a trapezoidal membership function is represented with the help of four parameters, marked a, b, c and d, with the mathematical expression (2) and shape illustrated in Figure 5:

$$f(x;a,b,c,d) = \begin{cases} 0 & x \le a \text{ or } x \ge d \\ \dfrac{x-a}{b-a} & a \le x \le b \\ 1 & b \le x \le c \\ \dfrac{d-x}{d-c} & c \le x \le d \end{cases} \quad (2)$$

where

$$a < b \le c < d \quad (3)$$

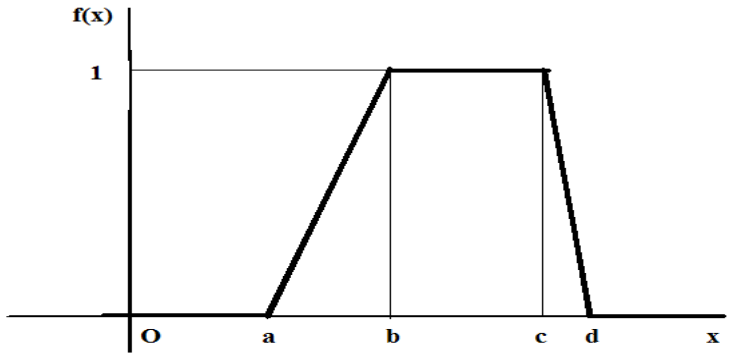

Figure 5. Trapezoidal membership function.

This representation is useful when we do the calculations required to determine the value of the output size of the controller or when we graphically represent memberships functions. However, it is not helpful when we apply the specific operations of the genetic algorithm because we need to introduce a number of restrictions where in the case of both mutation and crossover, so that for the resulting individuals relationship (3) is maintained, which burdens and slows down the genetic algorithm. We implemented this method of representation by means of the structure termed **cromozom**:

```
struct cromozom
{  double parammfx[5][4];      /* parameters for the 5 memberships functions of x */
   double parammfphi[7][4];    /* parameters for the 7 memberships functions of phi*/
   double parammfteta[7][4];   /* parameters for the 7 memberships functions of theta*/
   int typeand, typeor, typeimplication, typeagregation, typedefuzz, rules[5][7] ;
};
```

3.1.2 Our own representation of trapezoidal membership functions

We implemented the representation of trapezoidal membership functions as follows:

- for the membership functions for *x* 9 real parameters:
  - 3 values representing the ratio between the larger base of the trapezium and the average value of the universe of discourse – a value within the range [0.25 ; 2.0] for variables CE, RC and RI.
  - 2 values representing the percentage of the larger base of the trapezium overlapping the larger base of the trapezium placed on its left - a value within the range [0.05 ; 0.4] for pairs CE and RC, and respectively RC and RI

- 3 values representing the ratio smaller base/larger base – a value within the range [0.01; 0.65] for CE, RC and RI.
- a value representing the position of the smaller base for CE – a value within the range [0.05 ; 0.95] out of the available interval calculated depending on the large basis for RC (for RI the shape of the function is a rectangular trapezium, and for CE the trapezium is isosceles).

- 13 real parameters that are determined in a similar manner, but adding one value to each parameter to represent the functions for the other two variables (φ and θ).

The advantage of this representation is that, regardless of the method chosen for the operation of mutation or crossover, the resulting values will belong to the intervals considered, so no further validation tests are needed. In our application, we used a structure called **cromozom2** corresponding to this way of representation:

```
struct cromozom2
{  int typeand, typeor, typeimplication, typeagregation, typedefuzz, rules[5][7] ;
   /* memberships functions for x*/
   double rapx[3];        /* ratio larger base / average value of the universe of discourse */
   double suprax[2];      /* percentage of the overlapping of larger bases: 0.05-0.4 */
   double procx[3];       /* ratio smaller base/larger base: 0.05-0.65*/
   double pozrelx;        /* position of the smaller: 0.05-0.95 */
   double rapphi[4], supraphi[3], procphi[4], pozrelphi2,pozrelphi1;     /*rapphi1 results */
   double raptheta[4], supratheta [3], proctheta [4], pozreltheta2,pozreltheta1;
};
```

3.1.3 Conversion from our own representation of transfer functions in their traditional form

In our application, we used our own representation for the operations that are specific to the genetic algorithm, whereas for simulating fuzzy controllers, for the graphic representation and for the output files generated by the application, we used the traditional form. It was necessary to implement a function converted into the traditional. For reasons of space, we only present the conversion for member functions of input x (those for φ and θ are similar).

```
struct cromozom conversie(struct cromozom2 c2, double xmin, double xmax, double phimin, double
phimax, double tetamin, double tetamax)
   {  struct cromozom c1;
      double bazamare[7], a[7][4], bazamica; int i,j;
      c1. typeand = c2. typeand; c1. typeor = c2. typeor; c1. typeimplication = c2. typeimplication;
      c1. typeagregation = c2. typeagregation; c1. typedefuzz = c2.typedefuzz;
      for (i=0;i<5;i++)
            for (j=0;j<7;j++)
                  c1.rules[i][j] = c2.rules[i][j];
      double medianax = (xmax + xmin)/2; double mediax = (xmax - xmin) / 5;
      c1.parammfx [2][3] = medianax + mediax * c2.rapx[0] / 2;                    /* CE */
      c1.parammfx [2][2] = medianax + c1.parammfx [2][3] * c2.procx[0];
      c1.parammfx [2][0] = medianax - c1.parammfx [2][3];
      c1.parammfx [2][1] = medianax - c1.parammfx [2][2];
      bazamare[2] = a[2][3] - a[2][0];
      c1.parammfx [4][0] = xmax - mediax * c2.rapx[2];                            /* RI */
      bazamare[4] = xmax - c1.parammfx [4][0];
      c1.parammfx [4][1] = xmax - bazamare[4] * c2.procx[2];
      c1.parammfx [4][2] = xmax + xmax/4;
      c1.parammfx [4][3] = xmax + xmax/2;
      c1.parammfx [3][0] = c1.parammfx [2][3] - bazamare[2] * c2.suprax[0];   /* RC */
      c1.parammfx [3][3] = c1.parammfx [4][0] + bazamare[4] * c2.suprax[1];
      bazamica = (c1.parammfx [3][3] - c1.parammfx [3][0]) * c2.procx[1];
      c1.parammfx [3][1] = c1.parammfx [3][0] + (c1.parammfx [3][3] - c1.parammfx [3][0] - bazamica) *
c2.pozrelx;
      c1.parammfx [3][2] = c1.parammfx [3][1] + bazamica;
      for (i=0;i<=1;i++)                                                          /* LE(i=0) and LC(i=1) */
            for (j=0;j<=3;j++)
                  c1.parammfx [i][j] = medianax - c1.parammfx [4-i][3-j];
      ….
      return c1 ;}
```

3.2 The Genetic Algorithm

In the genetic algorithm that we have implemented, one chromosome is a controller and it is a structure comprising a matrix with five rows and seven columns of integers values, 5 integer values and 35 real values that correspond to the parameters described in the previous paragraph. The genetic algorithm attempts to determine the optimum values of all these 75 parameters.

Input data:

for the fuzzy controller: the length and speed of the truck, the minimum and maximum values for x, y, φ, θ and set of fuzzy rules;

for the proper genetic algorithm: the number of ages, the number of chromosomes, the selection method (we have implemented three methods: Monte Carlo, "Tournament", and Michalewicz [5]), the probability of crossover, the probability of selecting a chromosome for the mutation, the probability of mutation for a gene, the number of tests for the fitness of chromosome and the three state variables x0, y0 and φ0 for each test. To assess a chromosome, we have simulated its route from the initial position to its final position (xf, yf, φf), and we compute the fitness using the following function:

$$fitness = \sum_{\text{test cases}} [2x_f^{\,2} + y_f^{\,2} + 5(\varphi_f - \frac{\pi}{2})^2] \tag{4}$$

Since the aim of the controller is to bring the truck to the coordinates point (0, 0, π/2), the function we have used is a penalizing one in relation to each of the three parameters that characterize the final state of the controlled system: the lower the value of the function, the better the chromosome.

Output data: the average performance of the population and the performance of the best chromosome following each generation; the parameters of all the controllers represented by the last generation chromosomes.

The implemented algorithm:

```
1. Read initial data
2. Initialize the parallel work mode
3. If the process is root then
      Randomly generate the initial population
4. For each age do
5.    If the process is root then
6.       For id←1, nr_procs do
7.             Send to the id process the data needed to assess the id chromosome
8.       For id←1, nr_procs do
9.          Receive fitness value calculated for the id process
10.      Selection; Crossover; Mutation
11.      If it is the last generation then
12.         Write the output data          //in text files
13.   Else                                 //the process is slave
14.      Receive the data needed to assess the chromosome
15.      For each assessment test do
16.         Compute the fitness
17.      Send the fitness value calculated at the root
18. Close the parallel work mode;
19. Stop
```

4 Experimental Results

We wrote the application in the C language, and we have used the mpich2-1.4.1 library for the parallel mode. We have run the application on an Intel HPC System, located at Lucian Blaga University, in Sibiu, comprising 14 nodes, each equipped with 4 Dual Core Intel Xeon processors. As parameters of the genetic algorithm, we have used the following: number of generations = 400, number of chromosomes = 200, crossover probability = 0.50, probability of selecting a chromosome for mutation = 0.25 and probability of mutation for a gene = 0.05. For the truck, we have chosen the following characteristics: length l = 5 m and backing up speed v = 1.4 m/s. The genetic algorithms with the parameters thus established were run for 14 randomly generated populations.

To assess the controllers we have obtained, we have used other 60 items of test data. The best controllers obtained in each of the 14 cases have been assessed with these tests. As assessment function, we have used the following measure of error from [6]:

$$\varepsilon = \sum_{i=1}^{11}\left(\left|x_{fi}\right| + \frac{0.4}{15}\left|90 - \varphi_{fi}\right|\right) \tag{5}$$

where x_{fi} and φ_{fi} are the values of x and φ in the final state from each of the 60 tests.
In Table II we present the average error of the best controllers resulted from the Genetic Algorithms using the three methods of selection we mentioned above.

Table II: Average error of the best controllers

Initial pop.	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14
Monte-Carlo	1.10	1.46	1.41	1.28	1.44	1.21	1.16	0.92	1.27	2.06	1.08	1.27	0.96	0.92
Tourna-ment	0.79	1.21	**0.60**	1.44	1.05	1.21	1.54	1.21	2.26	0.66	1.29	0.98	0.79	1.04
"Micale-wicz"	1.20	1.02	1.51	1.30	1.31	0.87	2.0	1.23	1.13	0.80	1.38	1.25	**0.60**	1.29

The results of each simulation for these tests were included into one of following three categories [9]: Good if $\varepsilon \leq 0.4$; average if $\varepsilon > 0.4$ but the controller led the truck to the loading dock (y=0); missed if the truck failed to reach the loading dock. Table III presents the number of tests for which the result was "good". The number of "missed" tests was zero for each best controller.

Table III: Number of "good" tests

Initial pop.	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14
Monte-Carlo	8	10	8	8	0	6	4	11	6	6	10	8	12	14
Tourna-ment	12	6	**30**	4	8	13	4	8	0	24	6	14	22	10
"Micale-wicz"	4	12	2	8	8	12	2	7	2	16	8	0	28	6

Fig. 6 illustrates the average fitness of the population that provided the best controller. Fig. 7 illustrates the fitness of the best chromosome obtained.

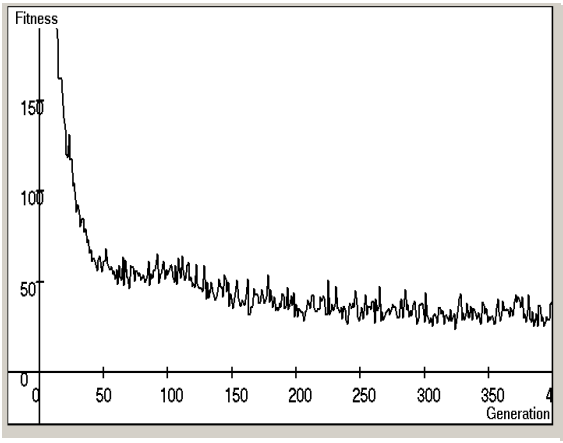

Figure 6. The average fitness of the population #1 "Tournament"

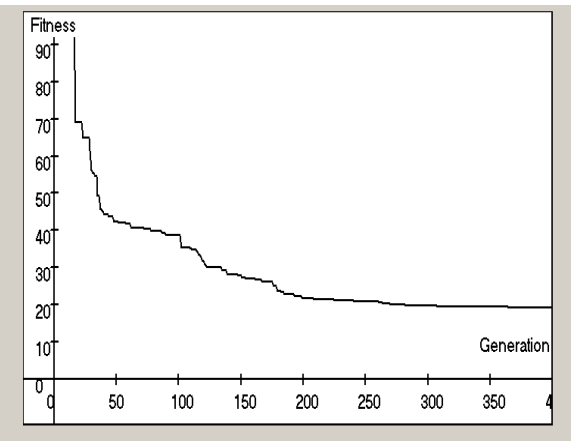

Figure 7. Best fitness of the population #3, "Tournament"

For 6 of the 60 starting items of test data we have illustrated in figure 8, the trajectories obtained by simulating the behavior of the best controller obtained.

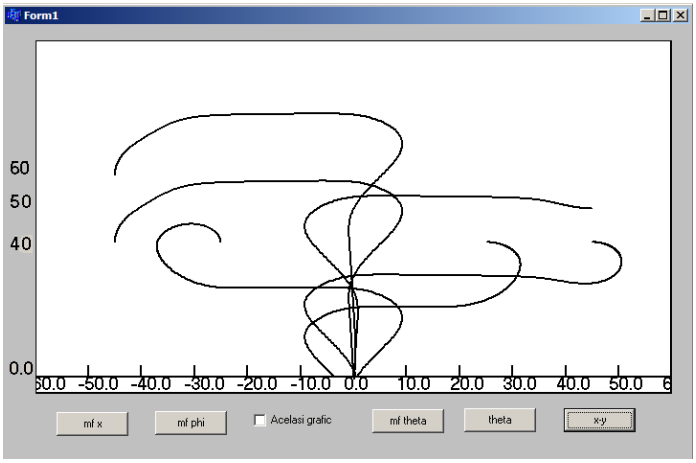

Figure 8. Truck trajectories for 6 initial positions obtained with best

The parameters of the best controller obtained are the following: product for fuzzy-AND, maximum for fuzzy-OR, product for the fuzzy implication, sum for the aggregation of the fuzzy rules and defuzzification using the sub-unitary weighted centroid method. The member functions for this controller are those represented in Fig. 2, Fig. 3 and Fig. 4. The Matrix of the Rules is similar to the one illustrated in Table I.

5 Conclusion

We can notice that:

The genetic algorithm proved its efficiency in all 42 tests we have performed. Thus, the ratio between the average fitness of the initial population and the average fitness of the final population equaled:

- 18.96 with the "Monte Carlo" method;
- 47.98 with the "Tournament" method;
- 46.03 with the "Micalewicz" method;

The ratio between the fitness of the best controller of the initial population and the fitness of the best controller of the final population equaled:

- 6.29 with the "Monte Carlo" method;
- 8.73 with the "Tournament" method;
- 7.83 with the "Micalewicz" method;

The best results have been obtained in the case of the algorithms that have used the "Tournament" selection method. The error of the best overall chromosome is 0.598 (coresponding to the best chromosome obtained from the initial population #3).

References

[1] Masood Anzar, Mohammad Fazle Azeem, Tanveer Chauhan & Anil Kumar Yadav, Generalized Approach for GA Based Learning of FLC Design Parameters, IICPE-2010
[2] Danilo Pelusi, Optimization of a Fuzzy Logic Controller using Genetic Algorithms, 2011 Third International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC) 2011
[3] D. Nguyen, B. Widrow, "The truck Backer Upper: An example of self learning in neural networks, in Neural networks for Control", The MIT Press, Cambridge MA, 1990.
[4] Z. Michalewicz, "Heuristic Methods for Evolutionary Computation Techniques", Journal of Heuristics, Vol.1, No.2, 1995, pp.177-206
[5] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs", Springer, 1998
[6] A. Riid, and E. Riistem, "Fuzzy logic in control: Truck Backer-Upper problem revisited," Proc. IEEE 10" Int. Conf. Fuzzy Systems, Melboume, Vol. 1, pp. 513-516, 2001.
[7] J. R. Koza, "A Genetic Approach to The Truck backer-upper Problem and the Inter-Twined Spiral Problem", IJCNN Intl. Conf. on Neural Networks, vol. 4, pp. 310-318, NY: IEEE Press, USA, 1992.
[8] S. Ciurea, I. Mihut, „Fuzzy Controller Design Using Genetic Algorithm Optimization", 8th International Symposium on Automatic Control and Computer Science - SACCS 2004, October 22 - 23, 2004, Iasi, Romania.
[9] S. Ciurea, „Determining the Parameters of a Sugeno Fuzzy Controller Using a Parallel Genetic Algorithm", Proceedings of the „9th International Conference on Control Systems and Computer Science, University Politehnica of Bucharest, Romania, 2013", ISBN 798-0-7685-4980-4, pg 36-43.
[10] Pintu Chandra Shill, Kishore Kumar Pal, Md. Faijul Amin, Kazuyuki Murase, "Genetic Algorithm Based Fully Automated and Adaptive Fuzzy Logic Controller", IEEE International Conference on Fuzzy Systems, June 27-30, 2011, Taipei, Taiwan
[11] Zhi-Long Wang, Chih-Hsiung Yang, Tong-Yi Guo*, The Design of An Autonomous Parallel Parking Neuro-Fuzzy Controller for A Car-like Mobile Robot, SICE Annual Conference 2010, Taipei, Taiwan
[12] http://www.mpitutorial.com.
[13] http://www.open-mpi.org/doc
[14] "Fuzzy Logic Toolbox™ User's Guide R2011b", The MathWorks, Inc., 2011.

STELIAN CIUREA
"Lucian Blaga" University of Sibiu
Faculty of Engineering, Department of Computer and Electrical Engineering
E. Cioran Str, No. 4, Sibiu-550025, ROMANIA,
E-mail: stelian.ciurea@ulbsibiu.ro