Using Xfuzzy Environment for the Whole Design of Fuzzy Systems

Iluminada Baturone, Francisco J. Moreno-Velo, Santiago Sánchez-Solano, Ángel Barriga, Piedad Brox, Andrés A. Gersnoviez, and María Brox

Abstract— **Since 1992, Xfuzzy environment has been improving to ease the design of fuzzy systems. The current version, Xfuzzy 3, which is entirely programmed in Java, includes a wide set of new featured tools that allow automating the whole design process of a fuzzy logic based system: from its description (in the XFL3 language) to its synthesis in C, C++ or Java (to be included in software projects) or in VHDL (for hardware projects). The new features of the current version have been exploited in different application areas such as autonomous robot navigation and image processing.**

I. Introduction

The research in fuzzy systems is so active that theoretical and practical advances are numerous. The first fuzzy systems were controllers with one or two rule bases with simple ‘if-then’ rules obtained from heuristic knowledge. Nowadays, fuzzy logic-based systems may contain fuzzy decision-making modules, fuzzy classifiers, and/or fuzzy controllers, which can be combined with non-fuzzy modules and interchange fuzzy or on-fuzzy values among them. The rules employed may be weighted by different values and may use different membership functions and operators (including linguistic hedges) to relate antecedents and consequents and to obtain the global conclusion. Besides, rule bases are usually obtained not only from heuristic knowledge but also from numerical data (the last method is particularly addressing a lot of attention for its relevance in the area of knowledge discovering), and the application of simplification as well as tuning methods to the obtained rules are also becoming usual practices.

This increase in complexity has motivated the evolution of the Xfuzzy environment. While Xfuzzy 1 focused on describing and simulating simple fuzzy controllers, the current Xfuzzy 3 uses a formal specification language, named XFL3, which facilitates the translation of complex rules expressed linguistically by allowing the use of rule weights, any kind of connective function to relate the antecedents, and linguistic hedges that may be applied to single or connected antecedents. In addition, this language allows the inclusion of new operators defined by the user as well as defining hierarchical modular systems [1].

The different versions of Xfuzzy have been distributed freely under the GNU General Public License. The last stable version, Xfuzzy 3.0, can be downloaded from its website: http://www.imse.cnm.es/Xfuzzy. This release contains a set of CAD tools which share the XFL3 language and offer Graphical User Interfaces to ease the design flow at the stages of description, tuning, verification, and synthesis. They are the following:

(a) *xfedit*, which eases describing the logical structure of a fuzzy system, that is, its inputs, outputs, groups of membership functions for each variable, sets of operators for each rule base, rule bases, and the system architecture (how rule bases are interconnected).

(b) *xfpkg*, which eases defining the function packages, that is, the code blocks describing the parameters, mathematical expressions and other features of membership functions, defuzzification methods, and unity and binary functions (related, respectively, to linguistic hedges and fuzzy connectives).

(c) *xf2dplot* and *xf3dplot,* to visualize graphically one of the outputs of the system against 2 or 3 of its inputs.

(d) *xfmt*, to monitor how the output values are obtained by inferring from the input ones.

(e) *xfsim*, to simulate how the fuzzy system behaves within the application domain.

(f) *xfsl*, which allows applying a wide set of supervised learning algorithms (gradient-descent, second-order, Gauss-Newton, and statistical algorithms).

(g) *xfc*, *xfcc*, and *xfj*, which, respectively, translate the description of the system in XFL3 to C, C++, and Java code.

The reader is referred to [2]-[3] to find a wider description of these tools.

This paper focuses on describing the new release, Xfuzzy 3.1, which is currently being tested and, hence, is not available yet at the website. The new features added to XFL3 and to some of the above mentioned tools are described in Section II. Section III summarizes the new tools incorporated to automate the process of extracting fuzzy rule bases from numerical data, to simplify the description of a fuzzy system, and to synthesize a VHDL-based description. All these capabilities are currently being exploited by our research group to design fuzzy controllers for autonomous robots and fuzzy processors for still image and video signal

This work was supported in part by the Spanish CICYT Projects DPI2005-02293 and TEC2005-04359, and by the Projects TIC2006-635 and TEP2006-375 from the Andalusian regional Government. The sixth author is supported by the Spanish Ministry of Education and Science under the program F.P.U. for PhD. students.

I. Baturone, A. Barriga, and A. A. Gersnoviez, are with the Instituto de Microelectrónica de Sevilla (IMSE-CNM-CSIC) and the Dept. de Electrónica y Electromagnetismo, Univ. de Sevilla, Seville, SPAIN (phone: +34-955-056-666; fax: +34-955-056-686; e-mail: {lumi, barriga, andres}@imse.cnm.es).

S. Sánchez-Solano, P. Brox, and M. Brox are with the Instituto de Microelectrónica de Sevilla (IMSE-CNM-CSIC), Seville, SPAIN (phone: +34-955-056-666; fax: +34-955-056-686; e-mail: {santiago, brox, maria}@imse.cnm.es).

F. J. Moreno Velo is with the DIESIA, Esc. Politécnica Superior, Univ. de Huelva, Huelva, SPAIN (e-mail: francisco.moreno@diesia.uhu.es).

processing, as described in Section IV. Finally, conclusions are given in Section V.

II. New Features of Existing Tools

A. *Defining families of membership functions*

A linguistic variable is defined in Xfuzzy 3 by using a *type* object. This definition includes the name of the type, the description of the universe of discourse (its limits and discretization), the list of associated linguistic labels, and their related membership functions. Until now, membership functions had to be "free" functions selected from a package, that is, they were defined independently and could not be explicitly related among them. For example, the variable 'x' shown in Figure 1 had to be defined by XFL3 as follows:

```
type Tx [0,100] {
very_small  xfl.triangle(-25,0,25);
small   xfl.triangle(0,25,50);
medium  xfl.triangle(25,50,75);
large   xfl.triangle(50,75,100);
very_large  xfl.triangle(75,100,125);}
```

A new feature added to XFL3 is that membership functions can be defined now as members of a family, that is, as functions explicitly related among them because share certain parameters. For example, the variable 'x' in Figure 1 could be also defined now as follows:

```
type Tx [0,100] {
family[]  xfl.triangular(25,50,75);
very_small  family[0];
small   family[1];
medium  family[2];
large family[3];
very_large  family[4];}
```

Several reasons have motivated the inclusion of this new feature. Firstly, the number of parameters to define a family of membership functions is smaller, which facilitates the tuning of the fuzzy system and permits the use of some automatic learning algorithms (such as simulated annealing) that are not appropriate with a large number of parameters. Secondly, it is easier to guarantee the linguistic meaning of the membership functions after applying an automatic modification process because, by construction, they cannot evolve to a state with highly overlapped or disordered functions. Finally, the use of certain families simplifies very much the hardware synthesis. As a disadvantage, systems that use families of membership functions cannot reach the optimization degree obtained with free functions, due to the imposed constraints. In general, free membership functions are more appropriate to describe output variables, while families of membership functions are specially indicated to describe input variables. A good practice is to use free membership functions for variables for which no much information is available, then perform an automatic tuning or identification process to acquire some knowledge about it, and, finally, use this information to employ a suitable family if possible.

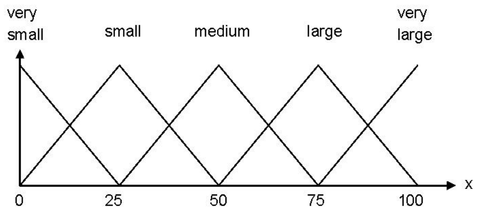
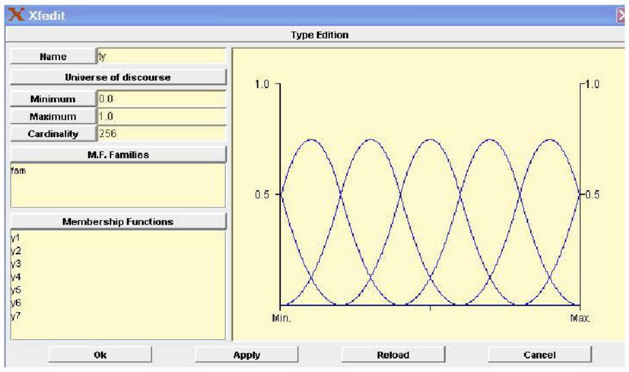


Figure 1. A linguistic variable and its membership functions.



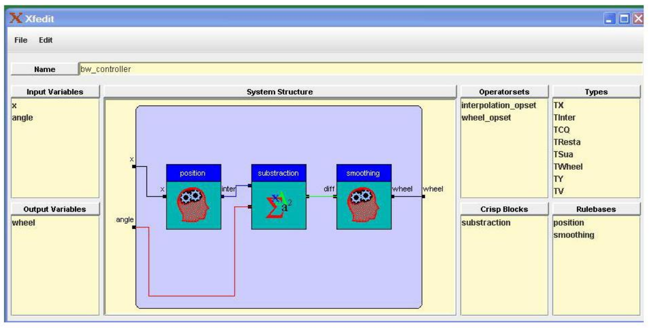
Name: ty
Universe of discourse
Minimum: 0.0
Maximum: 1.0
Cardinality: 256
M.F. Families
fam
Membership Functions
y1
y2
y3
y4
y5
y6
y7

Figure 2. A family of membership functions.

The tools *xfedit* and *xfpkg* (in particular) have been reprogrammed to admit this new feature. As example, Figure 2 shows the way how a family of membership functions based on second-order B-splines is shown by *xfedit*.

B. *Defining crisp modules*

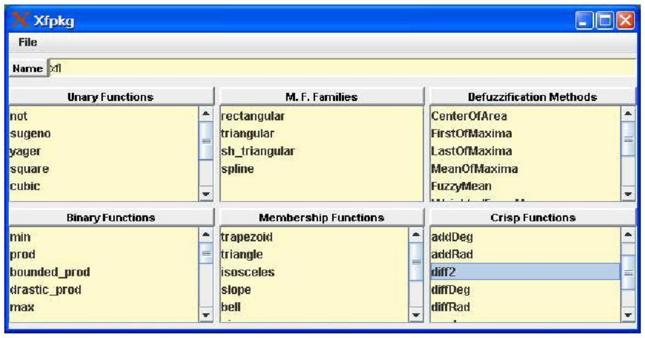
The structure of a system is defined in Xfuzzy 3 by using a *system* object. This definition includes the name of the global inputs and outputs and the list of interconnected modules. Until now, all the modules had to be rule bases containing 'if-then' rules. A new feature added to XFL3 is that now modules can be crisp, that is, they can implement any function on its inputs that is described mathematically by the Java code of its corresponding package. This allows designing complex systems in which crisp functions such as arithmetic operations or (de)multiplexers should be performed as intermediate steps between fuzzy inferences. As example, Figure 3 shows the main window of *xfedit* illustrating a system with fuzzy and crisp modules. The definition of this structure in XFL3 is as follows:



Name: bw_controller

Input Variables	Output Variables	Operatorsets	Crisp Blocks	Types	Rulebases
x	wheel	interpolation_opset	substraction	TX	position
angle		wheel_opset		TInter	smoothing
				TCQ	
				TResta	
				TSua	
				TWheel	
				TY	
				TV	

Figure 3. A system with fuzzy and crisp modules.



Name: xfl

Unary Functions	M. F. Families	Defuzzification Methods
not	rectangular	CenterOfArea
sugeno	triangular	FirstOfMaxima
yager	sh_triangular	LastOfMaxima
square	spline	MeanOfMaxima
cubic		FuzzyMean

Binary Functions	Membership Functions	Crisp Functions
min	trapezoid	addDeg
prod	triangle	addRad
bounded_prod	isosceles	diff2
drastic_prod	slope	diffDeg
max	bell	diffRad

Figure 4. Using *xfpkg* to define a crisp module.

```
rulebase position (TX x : TInter inter) using
wheel_opset { … }
rulebase smoothing (TSua diff : TWheel wheel)
using wheel_opset { … }
crisp { substraction xfl.diff2(); }
system (TX x, TCQ angle : TWheel wheel){
  position(x : inter);
  substraction(inter, angle : i0);
  smoothing(i0 : wheel);}
```

Figure 4 shows how the crisp function ‘diff2’ associated with the module ‘substraction’ is included in the package ‘xfl’.

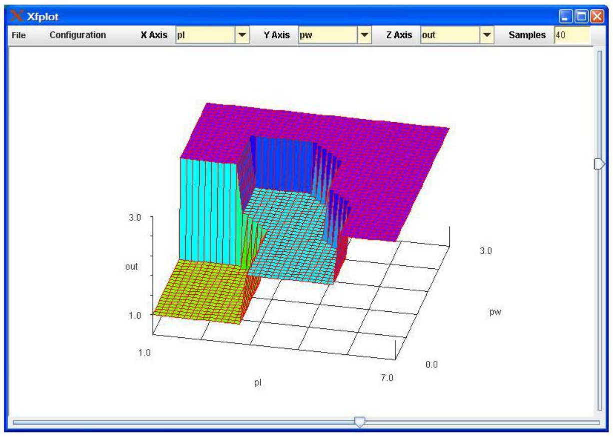
C. *Merging xf2dplot and xf3dplot into xfplot*

The previous tools *xf2dplot* and *xf3dplot* have been merged into a unique tool, named *xfplot*, in the new release of Xfuzzy. Figure 5 shows the main window of this new tool visualizing the output of a fuzzy classifier system. Advantages of this new tool are the inclusion of a ‘File’ menu (which allows saving data into a file) and a ‘Configuration’ menu (which allows selecting 2-D or 3-D graphic mode, a color palette to represent the output values, and loading or saving a configuration).

III. NEW TOOLS

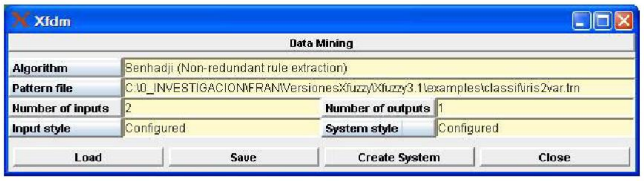
A. *Extracting fuzzy rule bases from numerical data*

Xfuzzy 3.0 does not allow obtaining rule bases from numerical data. They should be translated from linguistic knowledge, and numerical data were used only to tune the fine structure of the rule bases (the parameters of the membership functions). The new release, Xfuzzy 3.1 includes a new tool, named *xfdm*, developed to also employ these numerical data to obtain the coarse structure of the rule base (number of membership functions, number of rules, etc.). Hence, a new sub-menu ‘data mining’ has been included within the menu ‘tuning’ in the main window of Xfuzzy 3.1.



X Axis: pl
Y Axis: pw
Z Axis: out
Samples: 40

Figure 5. Graphical user interface of the tool *xfplot*.

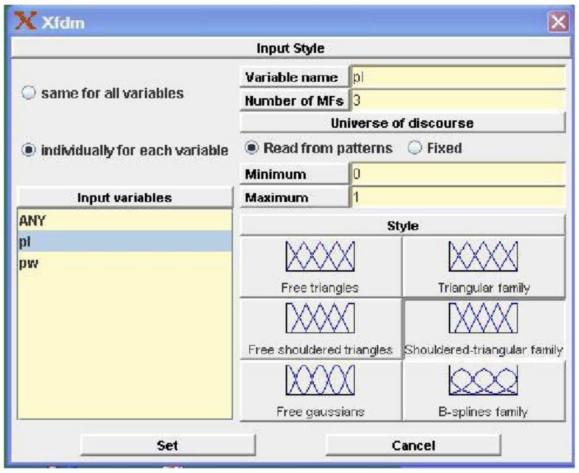


Algorithm: Senhadji (Non-redundant rule extraction)
Pattern file: C:\0_INVESTIGACION\FRAN\VersionesXfuzzy\Xfuzzy3.1\examples\classif\iris2var.trn
Number of inputs: 2
Number of outputs: 1
Input style: Configured
System style: Configured

Figure 6. Main window of the new tool *xfdm*.

Figure 6 shows the main window of the tool *xfdm*. It allows selecting: the grid- or clustering-based algorithm employed to extract the fuzzy rule base, the file with the numerical data, the number of inputs and outputs of the rule base to extract, and the input and system style of that rule base. The input style means the number and type of membership functions used to cover the input universes of discourse (free triangles or Gaussian functions as well as families of triangles and B-splines can be selected, as shown in Figure 7). The system style means to specify the name of the rule base to extract, the prefix used to name the output variables, the kind of conjunction operator used in the antecedents, and the type of inference-defuzzification method applied. The tool permits to identify different rule bases that could then be connected adequately with the tool *xfedit*, so as to describe the whole fuzzy system.

Following the same methodology adopted for the tuning tool *xfsl*, this tool *xfdm* includes a wide set of algorithms reported in the literature (one of them developed by our research group [4]) so as to cover as much as possible different application domains. Among the grid-based techniques (those which generate a grid partition of the input spaces prior to generate the rule base), three algorithms can



same for all variables [✓ individually for each variable]

Input variables: ANY, pl, pw

Variable name: pl
Number of MFs: 3
[✓ Read from patterns] Fixed
Minimum: 0
Maximum: 1

Figure 7. Window of *xfdm* to specify input style.

be selected which employ a fixed user-defined partition for the input variables (equal or different): *Wang&Mendel*, *Nauck*, and *Senhadji*. All of them evaluate which possible combination of input fuzzy sets is the most activated by each data (if there is any). In the *Wang&Mendel* algorithm no rule selection is implemented, so that the extracted rule base can be very large due to the curse of the dimensionality. *Nauck* and *Senhadji* algorithms avoid this problem since they allow selecting the maximum number of rules generated according to an efficiency measure [4]-[5].

The other grid-based algorithm that can be employed with *xfdm* is named *Incremental Grid*, which is based on the proposal in [6]. The tool allows obtaining the rules' consequents by applying or not learning. This method usually finds a better covering of the input variables than the other grid-based algorithms.

Cluster-based techniques generate simultaneously the rules and the membership functions of the variables from the clusters found. They usually generate simpler systems than those obtained by grid-based techniques but with less linguistic meaning. The tool *xfdm* includes four algorithms that employ a fixed user-defined number of clusters and are based on the *Hard C-means*, *Fuzzy C-means* [7], *Gustafson-Kessel* [8], and *Gath-Geva* [9] algorithms. They finish when reaching a maximum number of iterations or a minimum variation in the obtained clusters. The tool also includes an algorithm based on the proposal in [10], named *Incremental Clustering*, which finds the adequate number of clusters iteratively. This algorithm is configured by specifying the radius of influence of the obtained clusters and the maximum number of clusters to obtain.

B. Simplifying fuzzy rule bases

Rule bases obtained from heuristic knowledge and/or numerical data and possibly adjusted by supervised learning algorithms can often be translated into simpler systems by applying simplification algorithms. The new tool *xfsp* of Xfuzzy 3.1 allows applying simplification algorithms to either the variable membership functions or the rule bases.

Figure 8 shows the main window of *xfsp* when membership functions ('types') are selected to be simplified. It shows the three simplification processes which can be applied to them: purge mechanism, clustering and similarity-based merging method.

The purge mechanism looks for those membership functions which are not used in any rule base and eliminates them. This kind of membership functions may appear as a consequence of previous simplification processes or a non careful process of heuristic knowledge translation.

The clustering method looks for a reduced number of clusters (membership function prototypes) within the original functions. It applies the Hard C-Means algorithm, (the clusters found are crisp) on the space formed by the parameters that define the membership functions. The optimal number of membership function prototypes can be found automatically by applying validity indexes (Dunn Separation Index, Davies-Bouldin Index, and Generalized Dunn Indexes) [11] or can be fixed by the user after the

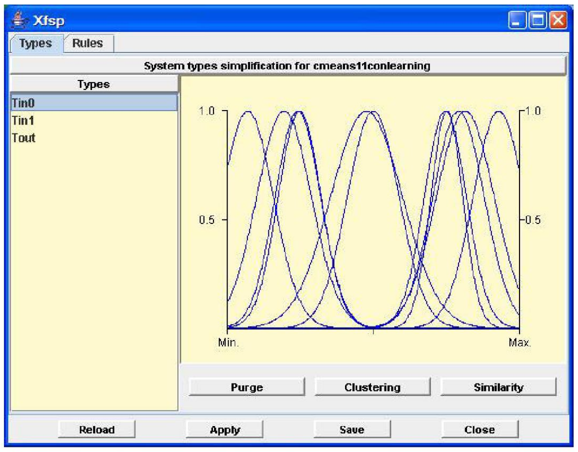


Figure 8. Window of *xfsp* for membership functions simplification.

visual inspection of the original functions. Figure 9a shows how the membership functions in Figure 8 are reduced from 11 to 5 (the user has selected 5 clusters).

The similarity-based merging process is an iterative process which looks for the pair of most similar functions and replaces them by a unique function if the similarity degree is over a threshold defined by the user. It finishes when no more functions can be merged. The similarity measure employed is the one defined by Dubois and Prade in [12]. Figure 9b shows how the functions in Figure 8 are reduced from 11 to 7 by applying a threshold of 0.7.

The results of applying clustering and similarity-based simplifications are similar. Advantages of using similarity-based method are that functions of different types (a triangle with a Gaussian, for instance) can be merged (which is not possible with clustering) and that the use of a threshold value can be more intuitive for the user. As a drawback, its computational cost is higher, although this cost is significant only for very complex systems.

The tool *xfsp* offers four methods to process the rules of a module: pruning, compression and expansion methods, and tabular simplification.

The compression method simply merges all the rules sharing the same consequent by connecting their antecedents disjunctively. In the other side, the expansion method implements the complementary process. They do not really perform simplification but only help the user to better understand the rule base. Simplification can be truly carried out by the pruning method and/or the tabular simplification.

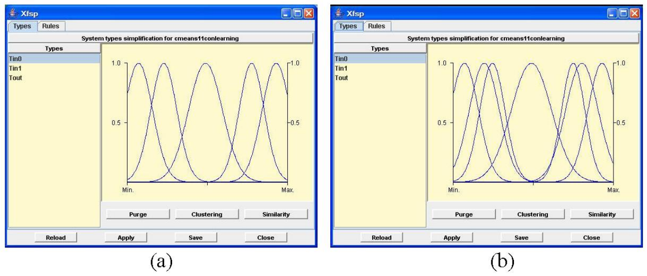


Figure 9. Results with (a) clustering and (b) similarity methods.

The pruning process allows reducing the number of rules by selecting the most significant ones to the application problem. Given a set of representative data, this process evaluates the activation degree of the rules and can eliminate: (a) the *n* worst rules, or (b) all the rules except for the *n* best rules, or (c) all the rules whose activation degree is below a threshold, where the parameter *n* or the threshold are established by the user.

The best rule simplification method available at *xfsp* is a tabular algorithm developed by some of the authors, which is based on an extension of the Quine-McCluskey algorithm of Boolean design. Tabular simplification is applied to each set of rules with the same consequent (although, for the case of *r* consequents, *r*-1 simplifications could be done by using the condition else for the *r*-th consequent). It selects the best 'prime implicants' to cover a consequent, which is equivalent to find the simplest rule associated with the considered consequent [13]. The linguistic hedges available at XFL3 are exploited to better express the resulting rule. In particular, the linguistic hedges 'not equal to' (!=), 'greater or equal to' (>=), and 'smaller or equal to' (<=) are used.

For example, the rule base expressed by XFL3 as follows:

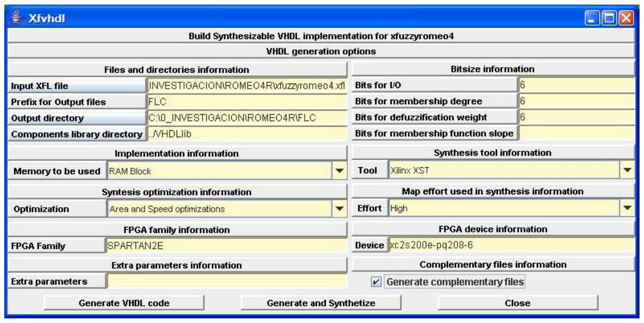
```
if(i0 == S & i1 == VS) -> out = low;
if(i0 == S & i1 == S) -> out = low;
if(i0 == S & i1 == M) -> out = low;
if(i0 == S & i1 == B) -> out = low;
if(i0 == S & i1 == VB) -> out = low;
if(i0 == M & i1 == VS) -> out = high;
if(i0 == M & i1 == S) -> out = high;
if(i0 == M & i1 == M) -> out = high;
if(i0 == M & i1 == B) -> out = high;
if(i0 == M & i1 == VB) -> out = low;
if(i0 == B & i1 == VS) -> out = high;
if(i0 == B & i1 == S) -> out = high;
if(i0 == B & i1 == M) -> out = high;
if(i0 == B & i1 == B) -> out = high;
if(i0 == B & i1 == VB) -> out = low;
```

is simplified by *xfsp* to the following rule base:

```
if(i0 != S & i1 != VB) -> out = high;
if(i0 == S | i1 == VB) -> out = low;
```

C. Generating VHDL code

One of the advantages of Xfuzzy 2.0 was its capability of generating the VHDL description of a fuzzy system described by XFL. The new release of Xfuzzy 3, Xfuzzy 3.1, has already the tool *xfvhdl* to automate the FPGA (Field Programmable Gate Array) implementation of a fuzzy system described by XFL3. The implementation follows a configurable active-rule driven architecture whose modules are defined and included into a library of parametric cells which meet the constraints of the employed synthesis tools (from Xilinx or Synopsys). The tool, whose main window is shown in Figure 10, allows the user to choose the bit sizes of the variables, and to select between: (a) memory-based or arithmetic circuits to implement the membership functions of the input variables, (b) knowledge base fixed (in a ROM) or programmable (in a RAM), (c) ROM implemented as combinational logic or distributed memory and RAM implemented as distributed or block memory.



Build Synthesizable VHDL implementation for xfuzzyromeo4

VHDL generation options

Files and directories information		Bitsize information	
Input XFL file	INVESTIGACION\ROMEO4R\xfuzzyromeo4.xfl	Bits for I/O	6
Prefix for Output files	FLC	Bits for membership degree	6
Output directory	C:\0_INVESTIGACION\ROMEO4R\FLC	Bits for defuzzification weight	6
Components library directory	../VHDLlib	Bits for membership function slope	
Implementation information		**Synthesis tool information**	
Memory to be used	RAM Block	Tool	Xilinx XST
Syntesis optimization information		**Map effort used in synthesis information**	
Optimization	Area and Speed optimizations	Effort	High
FPGA family information		**FPGA device information**	
FPGA Family	SPARTAN2E	Device	xc2s200e-pq208-6
Extra parameters information		**Complementary files information**	
Extra parameters		✓ Generate complementary files	

Figure 10. Main window of the tool *xfvhdl*.

IV. Application Examples

Xfuzzy environment is currently being used by our research group in the application areas of image processing and autonomous robotics.

Figure 11 shows several captions of the Xfuzzy tools (*xfedit* and *xfsim*) for designing a fuzzy system that de-interlaces video sequences. Description, learning, and verification tools have been specially employed to design this hierarchical system, which contains two fuzzy modules connected in cascade. More details about this application can be found in [14].

Figure 12 shows several captions of the Xfuzzy tools (*xfedit* and *xfsim*) for designing a fuzzy system that controls the traction and direction motors of a car-like robot so as to navigate towards a goal configuration with quasi-optimum paths and avoiding obstacles. Description, identification, simplification, and synthesis tools have been specially employed in this system that contains ten fuzzy modules and three crisp ones. The reader is referred to [15] to find more details on this application.

V. Conclusions

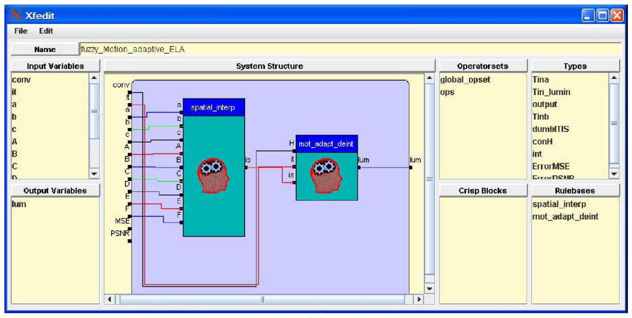
The new release of the Xfuzzy environment incorporates new useful features to define fuzzy systems (such as the capability of using families of membership functions and crisp modules) and to verify them (with the improvements to the tool *xfplot*). In addition, it includes new tools to generate fuzzy rule bases from numerical data (the tool *xfdm*), to simplify them (the tool *xfsp*), and to generate VHDL code (the tool *xfvhdl*). All these capabilities have allowed our research group to design efficient fuzzy systems for image processing and robotic applications.

References

[1] F. J. Moreno-Velo, S. Sánchez-Solano, A. Barriga, I. Baturone, D. R. López, "XFL3: A New Fuzzy System Specification Language", *Mathware & Soft Computing*, pp. 239-: 253 , December 2001.

[2] F. J. Moreno-Velo, I. Baturone, S. Sánchez-Solano, A. Barriga, "Rapid Design of Fuzzy Systems with XFUZZY", *Proc. 12th IEEE International Conference on Fuzzy Systems*, vol. 1, pp. 342-347, St Louis/MO, USA, May 2003.

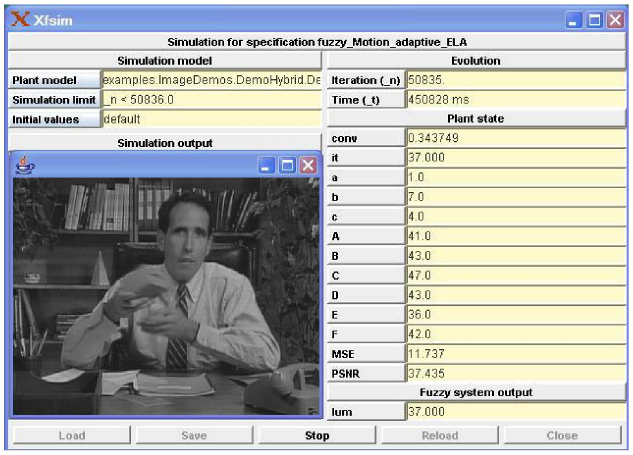
[3] F. J. Moreno-Velo, I. Baturone, R. Senhadji, S. Sánchez-Solano, "Tuning Complex Fuzzy Systems by Supervised Learning Algorithms", *Proc. 12th IEEE International Conference on Fuzzy Systems*, vol. 1, pp. 226-231, St Louis/MO, USA, May 2003.


Xfedit

Name	fuzzy_Motion_adaptive_ELA

Input Variables	Output Variables	Operatorsets	Types	Crisp Blocks	Rulebases
conv	lum	global_opset	Tina		spatial_interp
it		ops	Tin_lumin		mot_adapt_deint
a			output		
b			Tinb		
c			dumbITS		
A			conH		
B			int		
C			ErrorMSE		

(a)


Xfsim

Simulation for specification fuzzy_Motion_adaptive_ELA

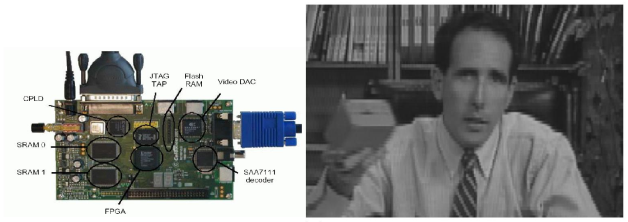
Simulation model	
Plant model	examples.ImageDemos.DemoHybrid.De
Simulation limit	_n < 50836.0
Initial values	default

Evolution	
Iteration (_n)	50835.
Time (_t)	450828 ms

Plant state	
conv	0.343749
it	37.000
a	1.0
b	7.0
c	4.0
A	41.0
B	43.0
C	47.0
D	43.0
E	36.0
F	42.0
MSE	11.737
PSNR	37.435

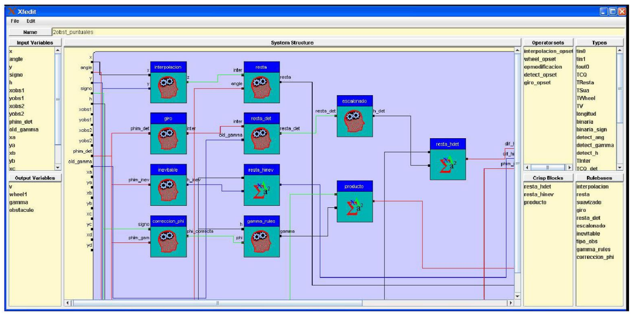
Fuzzy system output	
lum	37.000

(b)

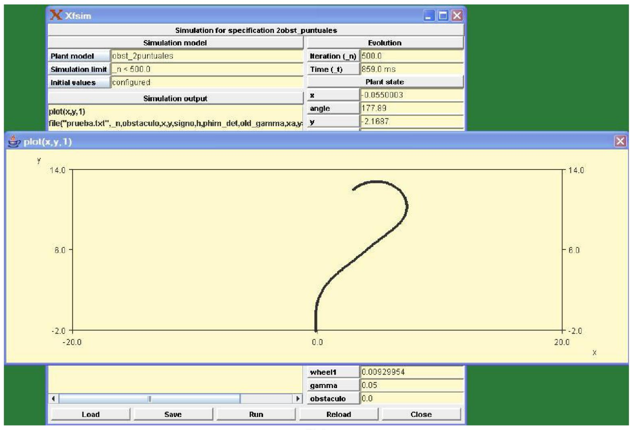


(c)

Figure 11. Image processing applications: (a) Description of the system. (b) Simulation. (c) Experimental results with a FPGA-based implementation.



(a)


Xfsim

Simulation for specification 2obst_puntuales

Simulation model	
Plant model	obst_2puntuales
Simulation limit	_n < 500.0
Initial values	configured

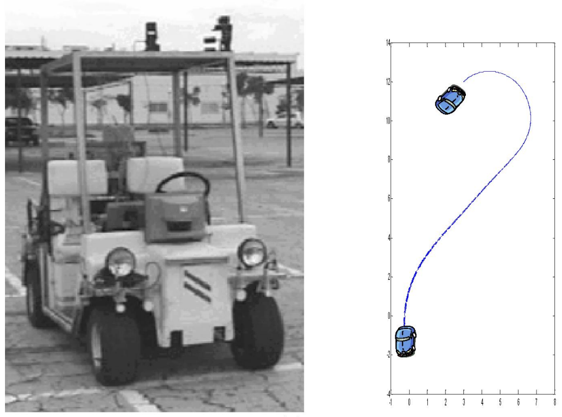
Evolution	
Iteration (_n)	500.0
Time (_t)	859.0 ms

Plant state	
x	-0.0550003
angle	177.89
y	-2.1687
wheel1	0.00929954
gamma	0.05
obstaculo	0.0

Simulation output

```
plot(x,y,1)
file("prueba.txt",_n,obstaculo,x,y,signo,h,phim_det,old_gamma,xa,y
```

(b)



(c)

Figure 12. Robotic applications: (a) Description of the system. (b) Simulation. (c) Experimental results with a PC-based implementation.

[4] R. Senhadji, S. Sánchez-Solano, A. Barriga, I. Baturone, F. J. Moreno-Velo, "NORFREA: An Algorithm for Non-redundant Fuzzy Rule Extraction", *Proc. IEEE SMC'2002*, vol. 1, pp. 604-608, Tunisia, Oct. 2002.

[5] D. Nauck, R. Kruse, "NEFCLASS - A Neuro-Fuzzy Approach for the Classification of Data", *Proc. 1995 ACM Symp. on Applied Computing*, Nashville, Feb. 26-28, pp. 461-465. ACM Press.

[6] C.H. Higgins, R.M. Goodman, "Fuzzy Rule-based Networks for Control", *IEEE Trans. on Fuzzy Systems*, vol. 2, n. 1, pp. 82-88, 1994.

[7] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York, 1981.

[8] D. E. Gustafson, W. C. Kessel, "Fuzzy Clustering with a Covariance Matrix", *Proc. IEEE Conf. on Dec. & Control*, San Diego, 1979, pp. 761-766.

[9] I. Gath, A. B. Geva, "Unsupervised Optimal Fuzzy Clustering", *IEEE Trans. on Pattern Analisis and Machine Intelligence*, vol. 11, pp. 773-781, 1989.

[10] S. L. Chiu, "A Cluster Estimation Method with Extension to Fuzzy Model Identification", *Proc. IEEE Int. Conf. on Decision*, pp. 1240-1245, 1994.

[11] E. H. Ruspini, P. P. Bonissone, W. Pedrycz, Eds., *Handbook of Fuzzy Computation*, Institute of Physics Pub., 1998.

[12] D. Dubois and H. Prade, *Fuzzy Sets and Systems: Theory and Applications*, New York Academic, 1980.

[13] I. Baturone, F. J. Moreno-Velo, A. A. Gersnoviez, "A CAD Approach to Simplify Fuzzy System Descriptions", *Proc. FUZZ-IEEE'2006*, pp. 2392-2399, Vancouver (Canada), July 2006.

[14] P. Brox Jiménez, I. Baturone, S. Sánchez-Solano, J. Gutiérrez-Ríos, F. Fernández-Hernández, "A Fuzzy Edge-dependent Motion Adaptive Algorithm for De-interlacing", *Fuzzy Sets and Systems* (Special Issue on Image Processing), vol. 158 (3), pp. 337-347, Feb. 2007.

[15] I. Baturone, F. J. Moreno-Velo, S. Sánchez-Solano, A. Ollero, "Automatic Design of Fuzzy Controllers for Car-Like Autonomous Robots", *IEEE Transactions on Fuzzy Systems* (Special Issue on Robotics), vol. 12, pp. 447- 465, Aug. 2004.