RAPID DESIGN OF FUZZY SYSTEMS WITH XFUZZY

F. J. Moreno-Velo, I. Baturone, S. Sánchez-Solano, A. Barriga

Instituto de Microelectrónica de Sevilla - Centro Nacional de Microelectrónica
Avda. Reina Mercedes s/n, (Edif. CICA)
E-41012, Sevilla, Spain

IEEE International Conference on Fuzzy Systems (FUZZ-IEEE’2003),
Rapid Design of Fuzzy Systems with XFUZZY

F. J. Moreno Velo, I. Baturone, S. Sánchez Solano, A. Barriga

Instituto de Microelectrónica de Sevilla (IMSE-CNM)
Centro Nacional de Microelectrónica - CSIC
Avda. Reina Mercedes, s/n. Edif. CICA, E-41012, Sevilla, Spain
xfuzzy-team@imse.cnm.es

Abstract

The crecient use of fuzzy systems in complex applications has motivated us to develop a new version of Xfuzzy, the design environment for fuzzy system created at the IMSE (Instituto de Microelectrónica de Sevilla). This new version, Xfuzzy 3.0, offers the advantages of being entirely programmed in Java, and allows designing hierarchical rule bases that can interchange fuzzy or non fuzzy values as well as employ user-defined fuzzy connectives, linguistic hedges, membership functions, and defuzzification methods. Xfuzzy 3.0 integrates tools that facilitate the description, tuning, verification, and synthesis of complex fuzzy systems. This is illustrated in this paper with the design of a fuzzy controller to solve a parking problem.

1. Introduction

Both theoretical studies on fuzzy systems and practical applications have been increasing since Fuzzy Logic was introduced in 1965. As a result, the fuzzy systems we can employ nowadays can be very complex. The set of membership functions, connective operators, and defuzzification methods that have been introduced is large; fuzzy rules can be expressed with linguistic hedges as well as certainty grades (or rule weights); and hierarchical systems that implement multistage fuzzy reasoning have been investigated and developed to solve more challenging problems.

As happens in many science and engineering fields, the use of CAD tools provides many advantages for designing complex fuzzy systems. They not only reduce the cost and the time-to-market of a potential product, but also ease the exploration of new theoretical aspects. This is why several tools tailored to the fuzzy paradigm have been developed in the last few years. The limitation is that many of them are dedicated to specific realizations and/or have constraints on the set of fuzzy operations they support, the complexity of the systems they can design, and their capability for automatic tuning, simulation or synthesis.

In order to reduce this limitation, our research group has been working on a new version of our development environment Xfuzzy [1] with the aim of fulfilling a wide set of ambitious objectives. The result has been Xfuzzy 3.0, whose main advantages can be summarized as follows.

Xfuzzy 3.0 has been entirely programmed in Java. This means that: (a) it is a structured environment that follows the object-oriented scheme which makes it easy to maintain and extend, (b) it can be executed on any equipment with JRE (Java Runtime Environment) installed, and (c) it offers the safety advantages of Java programs.

Xfuzzy 3.0 integrates several tools to cover all the stages of a fuzzy system design: description, tuning, verification, and synthesis stages. The nexus between all these tools is the use of a common specification language, XFL3, which allows the user to express very complex relations between the fuzzy variables [2]. On one side, XFL3 allows designing hierarchical rule bases that can interchange fuzzy or non fuzzy values and can employ the same or different fuzzy operators. On the other side, it allows the user to define new fuzzy connectives, linguistic hedges, membership functions and defuzzification methods.

The environment integrates all the tools under an homogeneous graphical interface which eases the design process (Figure 1). Anyway, every tool can be executed as an independent program.

This paper is focused on describing the capabilities of Xfuzzy 3.0 to ease the design of complex fuzzy systems. Section II summarizes the tools available for the system definition (description stage). Section III describes the tool

Figure 1: Main window of Xfuzzy 3.0.
to adjust or tune the defined system (tuning stage). The tools for simulation, monitoring and graphical representations of the system behavior, required at the verification stage are briefly described in Section IV while Section V summarizes the synthesis tools. An application example is included in Section VI to illustrate the advantages of using these tools.

2. Description of the fuzzy system

Xfuzzy 3.0 contains two tools for describing fuzzy systems: *xfedit* and *xfpkg*.

The first one is dedicated to the logical definition of the system, that is, the definition of its linguistic variables and the logical relations between them. Although this description can be done by editing a file with a ".xfl" extension, the tool *xfedit* offers a graphical interface which avoids the need for a deep knowledge of the XFL3 language. The user defines the hierarchy of the system at the main window of *xfedit* (Figure 2). From this window, other edition windows for linguistic variable types (Figure 3), rule bases (Figure 4), and operator sets (Figure 5) can be displayed.

The variable type edition window (Figure 3) eases the definition of the linguistic labels and their membership functions by showing them graphically on the universe of discourse and giving facilities for modifying their parameters.

The window for editing rule bases on *xfedit* (Figure 4) presents three formats for the rule definition: matrix form, table form and free form. The matrix form constrains the rule base to two inputs and one output, and shows the rules in a compact form as a matrix. Each matrix element represents a rule like «if a is A & b is B then c is C». The table form is valid for any number of inputs and outputs, and each element of the table represents a rule like «if x0 is X0 & x1 is X1 & ... & xn is XN then z is Z». Finally, the free form allows exploiting the whole power of XFL3 to define complex relations like «if x0 is greater than X0 or not x3 is strongly equal to X3 then z is Z».

At the operator set edition window (Figure 5), the user selects which mathematical function is assigned to the logical connectives (and/or/not), the rule implication and aggregation operators, and to the linguistic hedges and defuzzification methods.

The mathematical functions that appear at the operator set edition window and the other edition windows are those defined in files named as *packages*. The tool *xfpkg* facilitates the edition of function packages, which can be used by all the environment tools. At the main window of *xfpkg* (Figure 6), the user can add freely new mathematical functions. A complete definition requires to provide the information that any tool of the environment could employ.
includes the definition of the function parameters and their constraints, the description of the behavior in the Java, C and C++ languages (needed by the software synthesis), and the description of the differential function (required by some learning algorithms of the tuning stage). Figure 6 shows the mathematical functions that we have included in a package named as “xfl”.

3. Tuning stage

Tuning a fuzzy system is usually one of the most complex tasks of the design process. Although the system behavior depends on the logical structure of the rule bases and on the fuzzy operators employed, the tuning process often deals with the modification of the parameters defining the membership functions. Since this should be performed simultaneously over a great number of parameters, a manual procedure happens to be an extremely complex task, making it necessary the use of automated tuning techniques.

Xfuzzy 3.0 contains a tool named xfsl dedicated to the tuning of fuzzy systems with supervised learning algorithms. Figure 7 illustrates the main window of xfsl. In order to configure the learning process, the first step is to select a training file that contains the input/output data of the desired behavior. A test file, whose data are used to check the generalization of the learning, can be also selected.

Xfsl admits a wide set of algorithms: (a) Steepest Descent, Backpropagation, Backpropagation with Momentum, Adaptive Learning Rate, Adaptive Step Size, Manhattan, QuickProp and RProp (as gradient descent algorithms), (b) Polak-Ribiere, Fletcher-Reeves, Hestenes-Stiefel, One-step Secant and Scaled Conjugate Gradient (as conjugate gradient algorithms), (c) Broyden-Fletcher-Goldfarb-Shanno, Davidon-Fletcher-Powell, Gauss-Newton and Marquardt-Levenberg (as second-order algorithms), (d) the Downhill Simplex and Powell’s method (as algorithms without derivatives), and (e) the Blind Search and Simulated Annealing with linear, exponential, classic, fast, and adaptive annealing schemes (as statistical algorithms).

Xfsl can be applied to any fuzzy system described by the XFL3 language, even to systems that employ particular functions defined by the user. An interesting feature of xfsl is that several error functions can be minimized: mean square error (the default function), mean absolute error, classification error (for classification problems), and advanced classification error (to consider not only the number of classification fails but also the distances from right classifications). If the system has several output variables, they can be weighted by different values so that the designer can select their relative influence on the global error. Another useful feature of xfsl is that the user can select the system parameters to tune by a graphical interface.

In addition, xfsl contains two processing algorithms to simplify the designed fuzzy system. The first algorithm prunes the rules and reduces the membership functions that do not reach a significant activation or membership degree. The second algorithm clusters the membership functions of the output variables. This is very useful for system identification problems.

4. Verification stage

The objective of the verification stage is to study the behavior of the system under development, detecting probable deviations on the expected behavior and identifying the source of these deviations. Xfuzzy 3.0 contains four verification tools for these purposes: xf2dplot, xf3dplot, xfmt, and xfsim.

The first two ones are dedicated to graphically represent the system behavior in two (xf2dplot) or three dimensions (xf3dplot) (Figure 8). The tool xfmt allows monitoring the system at all the hierarchical levels, showing the activation degree of every linguistic label and logical rule, as well as the value of the
different inner variables, for some determined input values. Figure 9a shows the main window of xfmt that corresponds to the top level of the system hierarchy, while Figure 9b illustrates the monitoring of one of the rule bases of the system.

The verification tool xfsim allows simulating the fuzzy system together with an external system in a closed-loop configuration. The external system can be either a model of the actual environment to which the fuzzy system will interact, or the actual environment itself if this is connected to the design equipment by some acquisition board. Xfsim requires the external system to be represented by a Java class with a given interface. The main window of xfsim (Figure 10) allows selecting this Java class, the initial input values to the fuzzy system, and the end conditions of the simulation. Two types of output formats are provided by xfsim: data files or graphical representations (Figure 11) wherein the user can show the evolution of any simulation variable or their relation.

5. Synthesis stage

The final stage of a fuzzy system design is to generate a system implementation, either software or hardware. Selecting the proper solution depends on the complexity of the system and the imposed requirements. When inference speed, area and power requirements are not restrictive, software implementations are preferred to offer a great flexibility and the possibility of incorporating the system into a more general software project. Xfuzzy 3.0 provides the user with three tools for software synthesis: xfc, xcpp, xfj, which generate descriptions of the fuzzy systems in C, C++ and Java languages, respectively.

Under study is the incorporation of a tool dedicated to hardware synthesis, with functionality similar to the one available with the previous version of Xfuzzy [1].

6. Application example

Let us consider as an example the typical problem addressed in the literature of backing up a vehicle so as to arrive at a desired loading dock at a right angle with the horizontal [3]. The input variables considered in this example are the x position of the vehicle and the vehicle’s orien-
The output control variable is the required wheel angle (Figure 12).

In order to design the rule base of this controller, we have generated a set of trajectories of ideal maneuvers which cover as much as possible the universes of discourse of the input variables (x from -10m to 10m, and the vehicle angle from -180° to 180°). This kind of trajectories are shown in Figure 13. The data generated are saved into a file for training our fuzzy controller.

The initial fuzzy controller we consider covers the input variable x with 7 linguistic labels and the vehicle angle with 9 linguistic labels, both of them represented by gaussian membership functions. The controller contains the 63 possible rules, each one with its own consequent, so that 63 singletons are considered for the output variable. The idea is to start from a system with enough complexity to solve our problem.

Since we do not apply any knowledge at first, all the membership functions of the input variables cover the universes of discourse uniformly, and the 63 singleton values are taken equal, thus providing a flat control surface.

After describing our initial fuzzy controller with the tool xfedit, we applied the tool xfsl to teach the ideal trajectories to the the fuzzy system. Applying the clustering process supported by xfsl, the number of output singletons are reduced from 63 to 7 and the RMSE obtained is 13.19%. The rule base learned is shown in Figure 14.

The flexibility of the language XFL3 allows summarizing these 63 rules by the following 6 rules which express very clear a linguistic knowledge:

1.- If x ≥ Right Medium and angle < RIght ⇒ wheel = -0.4.
2.- If x ≥ ZEro and angle < ZEro ⇒ wheel = -0.4.
3.- If angle < LEft ⇒ wheel = -0.4.
4.- If x ≤ Left Medium and angle > LEft ⇒ wheel = 0.4.
5.- If x ≤ ZEro and angle > ZEro ⇒ wheel = 0.4.
6.- If angle > RIght ⇒ wheel = 0.4.

A fuzzy controller with these 6 rules only needs 3 linguistic labels for the variable x, 3 for the variable angle, and 2 singleton values for the wheel angle, so that we removed the other labels and singletons and tuned again this simple controller with xfsl. The RMSE now obtained is 13.14%. The learned linguistic labels are shown in Figure 15.

The result is that we have designed a linguistically interpretable fuzzy controller from numerical data which makes the vehicle follow the ideal maneuvers employed for train-
This has been verified by simulation with the tool \textit{xfsim}, as illustrated in Figure 16.

A better solution for diagonal parking a car is to drive not only backward but also forward. Hence, a better fuzzy controller can be designed as a hierarchical system that performs a fuzzy decision making (deciding the driving direction) prior to choose the wheel angle and even taking into account smooth changes of driving direction to consider the constraints of a true car. The structure of this hierarchical controller is shown in Figure 17. Figure 18 illustrates the simulation results obtained with \textit{xfsim} for an starting position where the car is parallel to the parking line. The first decision taken by the controller is to drive forward until arriving a good position to then drive backward. The rule base \textit{brake} takes care of performing a soft transition between driving forward and backward, as shown in Figure 18b.

This controller has been applied to park an autonomous mobile robot. After verifying the behavior of this controller, the tool \textit{xfc} was used to generate its C code and include it into the general software that controls the robot [4].

7. Conclusions

The version 3.0 of Xfuzzy presented in this paper is a complete environment for the design of fuzzy systems. The main advantages over other fuzzy system design tools are the degree of complexity it can manage (defining systems with hierarchical rule bases; fuzzy rules with any combination of connectives and linguistic hedges; and user-configurable membership functions, defuzzification methods, fuzzy connectives, linguistic hedges, implication and aggregation operators) and a set of tools that cover efficiently the different stages of the design process, including the capability for tuning automatically these complex fuzzy systems. These issues have been shown by an illustrative example. The environment is distributed freely under the GNU General Public License from the Xfuzzy official web page (http://www.imse.cnm.es/Xfuzzy/) and, since it has been programmed entirely in Java, it can be executed on any platform containing the Java Runtime Environment (JRE).

References


