APLICACIÓN DE XFUZZY 3 AL PROCESADO DE IMÁGENES BASADO EN REGLAS

Iluminada Baturone Piedad Brox Rosario Arjona

Instituto de Microelectrónica de Sevilla, (IMSE-CNM-CSIC) y Dpto. Electrónica y Electromagnetismo, Universidad de Sevilla (España); {lumi, brox, arjona}@imse-cnm.csic.es

Resumen

Los entornos de desarrollo de sistemas fuzzy se han empleado normalmente para diseñar sistemas de control y de toma de decisiones pero apenas para diseñar sistemas de procesado de imágenes, a pesar de que este campo cuenta ya con numerosas soluciones basadas en Lógica Fuzzy. En este artículo se muestra cómo el entorno Xfuzzy 3 desarrollado en el Instituto de Microelectrónica de Sevilla posee la versatilidad necesaria para abordar el diseño de estos sistemas, facilitando su descripción, verificación, ajuste y síntesis.

Palabras Clave: Sistemas fuzzy, procesado de imágenes, herramientas de CAD.

1 INTRODUCCIÓN

Al poco tiempo de introducirse la Lógica Fuzzy en 1965, se presentó el fundamento del control fuzzy en 1972 [4] y la primera realización práctica en 1976 [10]. Desde entonces, las aplicaciones en control han sido las más numerosas y las que más atención han recibido a nivel académico y comercial. Este enorme interés ha dado lugar al desarrollo de distintos entornos software especialmente orientados al diseño de controladores difusos (como FIDE de Aptronix, Inc., rFLASH, de Rigel Corporation y FuzzyTECH, de Inform).

Más tarde, las ventajas de la Lógica Fuzzy empezaron a explotarse en otros campos de aplicación. En relación al procesado de imágenes, los primeros trabajos se publican en los 80 [14]. La imprecisión y ambigüedad inherentes a las imágenes las hacen adecuadas para su procesado mediante Lógica Fuzzy. Estas características se dan a bajo nivel debido al ruido y la posible baja resolución y calidad de las imágenes. También se dan a alto nivel porque las fronteras de los objetos y regiones de una imagen así como sus distancias, posiciones relativas, etc. se pueden modelar con conceptos fuzzy. Una línea de trabajo seguida por muchos autores ha sido desarrollar una morfología matemática fuzzy como extensión de la morfología clásica, definiendo nuevos operadores morfológicos fuzzy [13]. Otra línea con gran aceptación ha sido emplear reglas fuzzy basadas en conocimiento heurístico. Varios autores han propuesto bases de reglas para el suavizado, aumento de contraste, extracción de bordes y filtrado [16]. También a alto nivel se han empleado reglas fuzzy para reconocer y localizar objetos en aplicaciones de visión por ordenador o por robot [6]. Sin embargo, pocos entornos software para Lógica Fuzzy han sido empleados en estas aplicaciones. Específicamente para el procesado de imágenes tan sólo encontramos en la literatura la herramienta propuesta por Russo, que se aplica a un problema de extracción de bordes [15].

El entorno Xfuzzy 3[1] desarrollado en el Instituto de Microelectrónica de Sevilla para el diseño de sistemas fuzzy ha sido empleado típicamente para resolver problemas de control como la mayoría de los entornos [1]. Sin embargo, la versatilidad que le proporciona su lenguaje de especificación XFL3, que soporta reglas cercanas a las expresadas en lenguaje natural, con operadores que pueden ser definidos libremente por el usuario y estructuras jerárquicas con módulos fuzzy y no fuzzy, permite la utilización del entorno en muchos otros campos de aplicación, en particular en el del procesado de imágenes.

En este artículo se presenta cómo Xfuzzy 3 permite el diseño de sistemas de procesado de imágenes basados en reglas. En la Sección 2 se resumen las características y herramientas que facilitan dicho diseño. En la Sección 3 se ilustran 3 ejemplos de aplicación en: (a) extracción de bordes, (b) aumento de resolución y (c) desentrelazado de

[1] https://forja.rediris.es/projects/xfuzzy
http://www.imse-cnm.csic.es/Xfuzzy

secuencias de vídeo. Las conclusiones se indican en la Sección 4.

2 XFUZZY 3 PARA PROCESADO DE IMÁGENES

El entorno Xfuzzy 3 posee una serie de herramientas de CAD que comparten un lenguaje común de especificación de sistemas fuzzy denominado XFL3 [12]. Una de las características de este lenguaje es que separa la descripción de una base de reglas de la definición de los operadores que intervienen en ella. Esto significa que el cálculo de los grados de pertenencia (definición de las funciones de pertenencia), el de los grados de activación de las reglas (definición de las conectivas, de los operadores complemento y modificadores lingüísticos) y el cálculo de la salida no fuzzy (definición del método de concreción) se hace en base a la elección de funciones matemáticas definidas en ficheros externos al que define la estructura de la base de reglas. Estos ficheros, denominados "paquetes", pueden ser definidos libremente por el usuario, lo cual es una gran ventaja para extender la aplicabilidad de Xfuzzy porque según el campo de aplicación los operadores pueden ser diferentes.

En el caso del procesado de imágenes, son varios los trabajos en que aparecen operadores particulares que no son habituales en control. Así por ejemplo, en [8] se utiliza el modificador lingüístico "la mayoría de", empleándose una función particular para definirlo, y en [7] se define un operador de "intensificación" y de concreción particulares. Otra diferencia con respecto al control es que el número de entradas que se contempla en las bases de reglas suele ser más elevado por lo que suelen especificarse sólo algunos casos posibles y el resto se engloba en una regla del tipo "else", cuyo grado de activación se calcula de forma diferente según los autores [2, 8, 16].

Para estos casos, el usuario de Xfuzzy 3 puede definir paquetes específicos con funciones particulares de tipo: (a) binarias (operando sobre dos argumentos, como t-normas y s-normas), (b) unarias (sobre un argumento, como c-normas y modificadores lingüísticos), (c) funciones de pertenencia y (d) métodos de concreción. La definición de una función incluye su nombre (y alias posibles), los parámetros que especifican su comportamiento (así como posibles restricciones sobre estos parámetros), la descripción de su comportamiento en Java (y en C y C++ si se quiere generar síntesis a esos lenguajes) e incluso la descripción de sus derivadas si se quiere emplear esta función en métodos de aprendizaje guiados por gradiente. El usuario puede introducir toda esta información de forma gráfica mediante la herramienta *xfpkg* (Figura 1). Con esta utilidad se genera automáticamente una clase Java que incorpora todas las

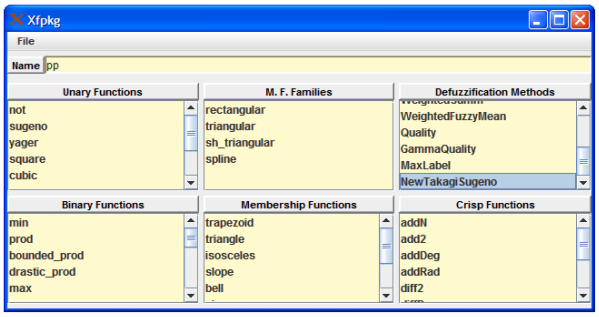


Figura 1: Ventana principal de la herramienta *xfpkg*.

características de la función y que puede emplearse por cualquier especificación XFL3.

Por otro lado, el procesado de imágenes suele encadenar razonamientos. Por ejemplo, en [9] se emplea una base de reglas fuzzy para seleccionar los parámetros de un filtro gaussiano (no fuzzy) y en [5] la inferencia fuzzy determina el tipo de filtro (fuzzy o no) a seleccionar.

Para estos casos, el usuario de Xfuzzy 3 puede definir los distintos módulos constitutivos tanto fuzzy como no fuzzy (o *crisp*) y conectarlos con la jerarquía adecuada. Esto lo puede hacer de forma gráfica mediante la herramienta *xfedit*. En la ventana principal de esta herramienta, el usuario puede definir las distintas bases de reglas o módulos fuzzy (cada una con sus funciones de pertenencia y operadores propios) así como seleccionar los distintos módulos crisp. Los módulos crisp se definen en Java en un paquete que deberá estar cargado en el entorno (para esta definición de módulos crisp también es muy útil la herramienta *xfpkg* comentada anteriormente). Pueden definirse para aplicar cualquier función matemática sobre sus entradas. Otra versatilidad interesante que permite XFL3 es que, a la hora de interconectar módulos fuzzy, se pueden intercambiar valores tanto crisp como fuzzy, según que se empleen o no métodos de concreción antes de intercambiar los valores.

Para analizar cómo el sistema definido procesa las imágenes, el usuario puede emplear la herramienta de simulación *xfsim*. Esta herramienta conecta el sistema con un modelo del contexto de operación (una planta, en la nomenclatura de control), que el usuario definirá como una clase Java. En el caso del procesado de imágenes, esta clase Java generará las entradas al procesado a partir de una(s) imagen(es) dada(s) y tomará las salidas del procesado para generar la(s) imagen(es) resultante(s) y los ficheros de datos (*log*) necesarios.

Si en la simulación no se obtienen buenos resultados, suele ser muy útil la herramienta *xfmt*, que permite monitorizar las entradas y salidas de cada módulo del sistema de procesado visualizando las funciones de pertenencia en cada nivel y los grados de activación de las

reglas. Así puede detectarse si se ha cometido algún error en la traducción a reglas del conocimiento heurístico.
Para mejorar de forma automática los resultados de un sistema de procesado se pueden aplicar métodos de aprendizaje que ajusten los parámetros del sistema. Para ello puede emplearse la herramienta *xfsl*, que permite aplicar una gran variedad de algoritmos de aprendizaje supervisado. Incluso si no se dispone de conocimiento heurístico para definir el sistema pero sí de datos numéricos, se puede emplear la herramienta *xfdm* para extraer automáticamente una base de reglas. En este caso, suele ser útil emplear a continuación la herramienta *xfsp*, que permite simplificar las funciones de pertenencia y/o las reglas de la base extraída.

Por último, el sistema de procesado definido en XFL3 puede traducirse de forma automática a Java, C y C++ utilizando las herramientas de síntesis *xfj*, *xfc* y *xfcc*, respectivamente.

3 EJEMPLOS DE DISEÑO

3.1. EXTRACCIÓN DE BORDES

Existen gran variedad de algoritmos que resuelven el problema de la extracción de bordes. Entre los que emplean técnicas de Lógica Fuzzy encontramos como referente el trabajo de Russo [16], en el que se propone un sistema basado en reglas. Se trata de un sistema que admite como entradas las diferencias de luminancia de un píxel con sus ocho vecinos más cercanos. De esta manera, los antecedentes de la base de reglas tienen en cuenta los valores positivos o negativos de estas diferencias y las posibles combinaciones de los bordes que se pueden dar (Figura 2). Tanto los conceptos “positivo”, “negativo”, como los posibles valores de la salida “negro”, “blanco”, se representan por funciones de pertenencia triangulares.

Estudios posteriores han modificado la base de reglas propuesta en [16]. Así por ejemplo, en [11], se añaden tres reglas más que cubren los casos en los que no existe borde y, por tanto, establecen el píxel de salida como

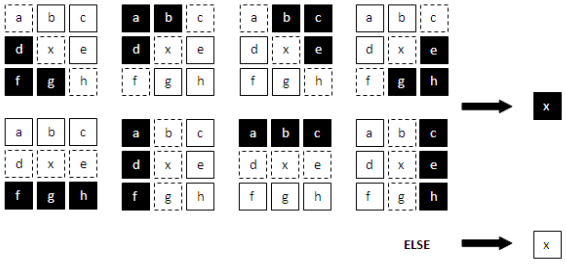


Figura 2: Reglas para la extracción de bordes en [16].

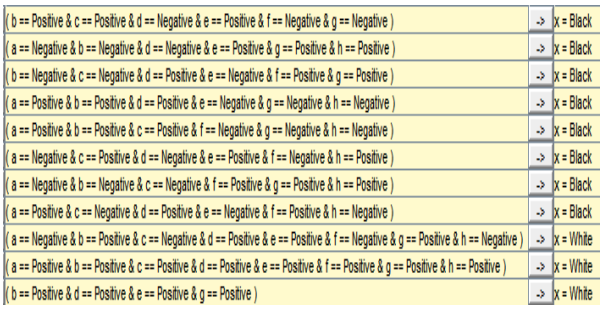


Figura 3: Base de reglas en *xfedit* para la extracción de bordes.

blanco. Esta base de reglas más extensa ha sido la descrita en Xfuzzy mediante *xfedit* (Figura 3).

Como operadores se han utilizado el producto para definir los conectivos de los antecedentes, min-max para la implicación y agregación, y el centro de área como método de concreción. Tanto los operadores como las funciones de pertenencia empleadas están ya incluidas en el paquete por defecto que carga el entorno por lo que, para este ejemplo, no ha sido necesario definir un nuevo paquete.

La simulación del sistema descrito se puede realizar haciendo uso de la herramienta *xfsim*. El modelo en Java construido para dicho fin nos permite seleccionar una imagen para calcular la extracción de bordes y ofrecer una imagen de salida como resultado de la aplicación del sistema difuso (Figura 4).

Puesto que las cuatro funciones de pertenencia involucradas en las reglas se han definido de forma heurística, lo más probable es que sus parámetros no sean óptimos. En vez de probar de forma manual diferentes valores, podemos emplear los algoritmos de aprendizaje supervisado de la herramienta *xfsl* y realizar un ajuste automático de parámetros (Figura 5a). En general, los resultados se mejoran, como se muestra en la Figura 5b.

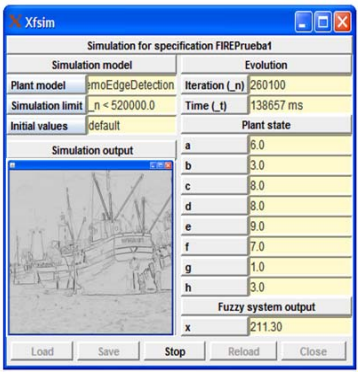
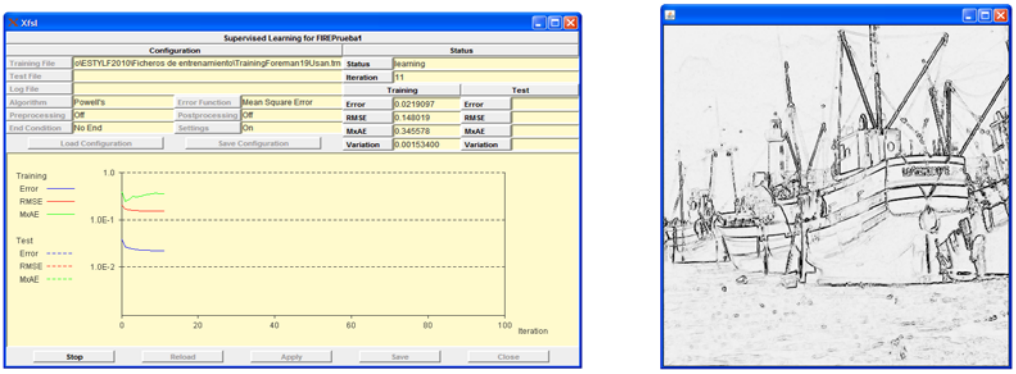


Figura 4: Simulación del sistema de extracción de bordes con *xfsim*.



(a) (b)

Figura 5: (a) Ventana de la herramienta de aprendizaje *xfsl*. (b) Resultado de la simulación tras el aprendizaje.

3.2. AUMENTO DE RESOLUCIÓN

Para agrandar o aumentar la resolución de una imagen hay que introducir nuevos píxeles que constituyan nuevas líneas y columnas. Esto se ilustra en la Figura 6 para un factor de amplificación igual a 2.

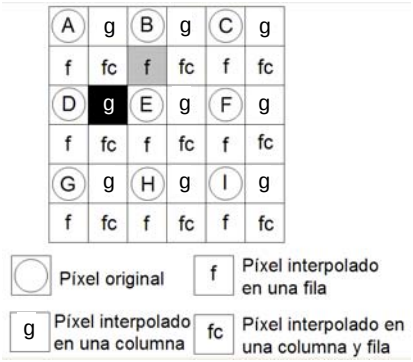


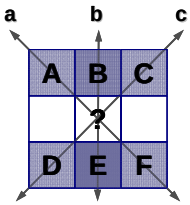
Figura 6: Aumento de resolución por un factor de 2.

El algoritmo propuesto en [2] consta de una primera fase en la que se interpolan los píxeles de las nuevas filas y columnas, que se muestran con los símbolos 'f' y 'g' respectivamente, en la Figura 6. Por ejemplo, para interpolar el píxel 'f' sombreado en gris se utilizan los píxeles originales {A, B, C, D, E, F}, mientras que para el píxel 'g' sombreado en negro, se emplean los píxeles originales {A, B, D, E, G, H}. La base de reglas fuzzy toma como entradas las 3 diferencias de luminancias en valor absoluto mostradas en la Figura 7a e infiere la luminancia del píxel a interpolar con las reglas mostradas en la Figura 7b. Se trata de reglas heurísticas que evalúan la posible existencia de bordes para interpolar adaptándose a ellos. Los píxeles mostrados con el símbolo 'fc' en la Figura 6 se interpolan en una segunda fase como el valor medio de los resultados obtenidos al aplicar la base de reglas anterior en los 3+3 píxeles de las líneas superior e inferior, y los 3+3 píxeles de las columnas izquierda y derecha (utilizando los cuatro píxeles originales más cercanos y los cuatro píxeles más cercanos interpolados en la fase previa).

El sistema fuzzy fundamental en este algoritmo es el que contiene las reglas en la Figura 7b. Éste es el que se ha descrito en Xfuzzy 3 con la herramienta *xfedit* (Figura 8a). Para ello se ha tenido que definir un nuevo operador de inferencia con la herramienta *xfpkg* puesto que la regla 'otherwise' tiene la particularidad de activarse como el complemento de la suma de los grados de activación de las otras 3 reglas.

Las funciones de pertenencia para los conceptos 'small', 'strongly small' y 'large' que aparecen en la base de reglas pueden describirse heurísticamente, sin embargo, puesto que podemos disponer de un conjunto de datos numéricos correspondientes a pares de imágenes con distinta resolución, estas funciones se han ajustado en Xfuzzy 3 con la herramienta *xfsl* aplicando el algoritmo de Levenberg-Marquardt.

Para simular el procesado se ha creado una clase Java que pide al usuario seleccionar una imagen. La clase va recorriendo los píxeles de la imagen proporcionando las entradas necesarias para ir interpolando los diferentes píxeles. Esta clase genera la imagen agrandada permitiendo su visualización (Figura 8b) y escritura en un

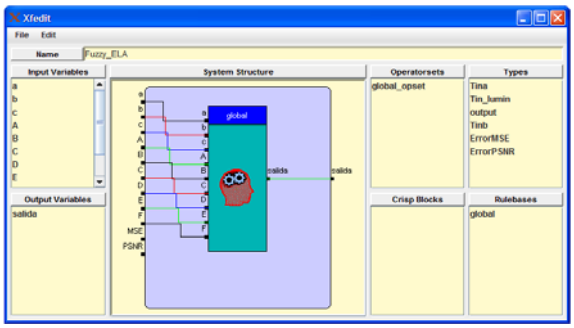


(a)

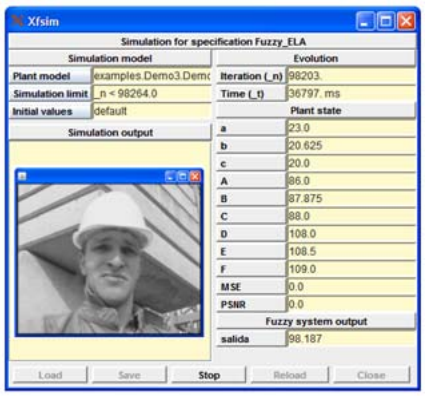
if	*antecedents*	*consequent*
1)	(a is SMALL) and (b is LARGE) and (c is LARGE)	(A+F)/2
2)	(a is LARGE) and (b is LARGE) and (c is SMALL)	(C+D)/2
3)	(a is strongly SMALL) and (b is LARGE) and (c is strongly SMALL)	(A+F+C+D)/4
4)	otherwise	(B+E)/2

(b)

Figura 7: (a) Direcciones evaluadas para aplicar interpolación adaptativa a los bordes según las reglas en (b).



(a)



(b)

Figura 8: (a) Sistema fuzzy para agrandado de imágenes descrito con *xfedit* (b) Simulación con *xfsim*.

fichero. Además, la clase permite seleccionar al usuario una imagen ya agrandada con respecto a la que calcular errores. Si se selecciona dicha imagen, la simulación compara con ella la imagen resultante del procesado y calcula el MSE (Mean Square Error) y el PSNR (Peak Signal-to-Noise Ratio).

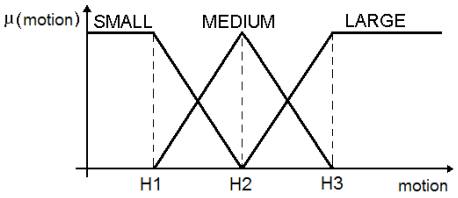
3.3. DESENTRELAZADO DE VÍDEO

En los estándares de TV analógica y en varios digitales no se envían las imágenes completas sino que se van intercalando imágenes con sólo las filas pares con otras con sólo las filas impares. Si el aparato donde se va a reproducir la secuencia no admite formato entrelazado es necesario desentrelazar las imágenes interpolando las filas que falten.

El algoritmo fuzzy descrito en el apartado anterior puede emplearse para calcular las filas que faltan a partir de las existentes. En la nomenclatura del desentrelazado se trataría de un interpolador espacial porque sólo considera las filas existentes en la imagen actual pero no en las precedentes (que sería una interpolación temporal). Sin embargo, el desentrelazado es mejor si se consideran ambos tipos de información espacial y temporal. Los algoritmos que se adaptan al movimiento se apoyan en la heurística para seleccionar interpolación espacial cuando el movimiento es grande e interpolación temporal cuando el movimiento es pequeño. El algoritmo propuesto en [3]

if	*antecedents*	*consequent*
1)	Motion is SMALL	I_T
2)	Motion is MEDIUM	$\gamma I_T + \lambda I_S$
3)	Motion is LARGE	I_S

(a)



(b)

Figura 9: (a) Reglas para adaptar la interpolación al movimiento. (b) Funciones de pertenencia.

usa la base de reglas mostrada en la Figura 9a para evaluar de forma fuzzy el posible movimiento que haya en torno al píxel actual y seleccionar un tipo de interpolador u otro. Para implementar la interpolación espacial se emplea el algoritmo fuzzy comentado anteriormente. Como interpolación temporal se recurre a copiar las líneas de la imagen entrelazada anterior.

En esta aplicación se ha aprovechado la versatilidad de Xfuzzy para trabajar con sistemas jerárquicos. La Figura 10 muestra la ventana de *xfedit* donde se aprecia la estructura del sistema formado por dos módulos fuzzy que intercambian información no fuzzy. El módulo que se adapta al movimiento se implementa como un sistema Takagi-Sugeno de orden 1, de modo que no requiere definición de nuevos operadores.

Las funciones de pertenencia para los conceptos ‘small’, ‘medium’ y ‘large’ referidos al movimiento (Figura 9b) pueden describirse heurísticamente. No obstante, se obtienen mejores resultados si se aprenden empleando la herramienta *xfsl*. Conviene aprender también los parámetros γ y λ de los consecuentes e incluso ajustar los antecedentes del interpolador espacial, aprovechando la capacidad de Xfuzzy para ajustar sistemas jerárquicos. En este caso, hemos aplicado el algoritmo de Levenberg-Marquardt pero estimando, en vez de calculando, las derivadas.

Para simular este procesado se ha creado una clase Java que pregunta al usuario por las 4 imágenes a partir de las cuales se va a obtener la imagen desentrelazada actual. La clase va generando las entradas correspondientes al sistema y toma el valor del píxel inferido para construir la imagen resultante que se visualiza gráficamente y puede salvarse a un fichero (Figura 11). Las imágenes de partida son progresivas por lo que la clase las entrelaza como paso previo. Esto permite que la clase calcule los errores MSE y PSNR entre la imagen progresiva perfecta y la desentrelazada.

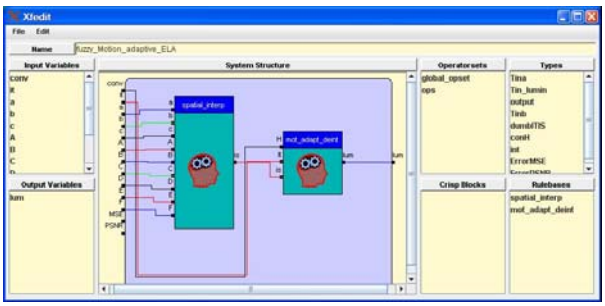


Figura 10: Sistema fuzzy para desentrelazado descrito en *xfedit*.

Figura 11: Simulación del sistema de desentrelazado con *xfsim*.

4 CONCLUSIONES

El entorno Xfuzzy 3, tradicionalmente usado en aplicaciones de control, también facilita el diseño de sistemas fuzzy para procesado de imágenes. Esto se ha ilustrado con tres ejemplos diferentes: la extracción de bordes, el aumento de resolución y el desentrelazado de imágenes.

Agradecimientos

Los autores agradecen el trabajo de Alejandro Valdivia, el primero en probar Xfuzzy 3 para procesar imágenes.
Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia y Tecnología mediante el proyecto TEC2008-04920 y por la Junta de Andalucía mediante el proyecto P08-TIC-03674.

Referencias

[1] 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*. Vol. 12, Pág. 447- 465, Aug. 2004.

[2] P. Brox, I. Baturone, S. Sánchez Solano, A. Barriga. “Image enlargement using the Fuzzy-ELA algorithm”. *Proc. IPMU 2006*, Pág. 866-873, Julio 2006.

[3] P. Brox, 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*. Vol. 158 (3), Pág. 337-347, Feb. 2007.

[4] S. S. L. Chang y L. A. Zadeh. “On fuzzy mapping and control”. *IEEE Trans. on Syst., Man and Cybern*. Vol. 2, Pág. 30-34, 1972.

[5] Y. S. Choi and R. Krishnapuram. “A robust approach to image enhancement based on fuzzy logic”. *IEEE Trans. Image Processing*. Vol. 6, No. 6, Pág. 808-825, 1997.

[6] K.-J. Choi, Y.-H. Lee, J.-W. Moon, C.-K. Park, F. Harashima. “Development of an automatic stencil inspection system using modified Hough transform and fuzzy logic”. *IEEE Trans. Ind. Electr*. Vol. 54, No. 1, Pág. 604-611, Feb. 2007.

[7] T. K. De and B. N. Chatterji. “An approach to a generalized technique for image contrast enhancement using the concept of fuzzy set”. *Fuzzy Sets and Systems*. Vol. 25, Pág. 145-158, 1988.

[8] F. Farbiz, M. B. Menhaj, S. A. Motamedi, and M. T. Hagan. “A new fuzzy logic filter for image enhancement”. *IEEE Trans. Syst., Man and Cybern*. Vol. 30, No. 1, Pág. 110-119, Feb. 2000.

[9] T. Law, H. Itoh, and H. Seki. “Image filtering, edge detection, and edge tracing using fuzzy reasoning”. *IEEE Trans. Pattern Analysis and Machine Intelligence*. Vol. 18, No. 5, Pág. 481-491, 1996.

[10] E. H. Mamdani. “Advances in the linguistic synthesis of fuzzy controllers”. *Int. Jour. Man-Machine-Stud*. Vol. 8, No. 6, Pág. 669-678, 1976.

[11] S. Mathur, A. Ahlawat. “Application of fuzzy logic on image edge detection”. *Int. Conf. Intelligent Information and Engineering Systems, INFOS 2008*, Varna, Bulgaria. June-July 2008.

[12] F.J. Moreno-Velo, S. Sánchez-Solano, A. Barriga, I. Baturone, D.R. López. “XFL3: An specification language for fuzzy systems”. *Mathware & Soft Computing*, Vol. VIII, No. 3, Pág. 239-253, 2001.

[13] M. Nachtegael and E. E. Kerre. “Classical and fuzzy approaches towards mathematical morphology”. Chapter 1 in *Fuzzy techniques in image processing*, Physica-Verlag, 2000.

[14] S. K. Pal and R. A. King. “Image enhancement using smoothing with fuzzy sets”, *IEEE Trans. Syst., Man and Cybern*. Vol. 11, No. 7, Pág. 494-501, 1981.

[15] F. Russo. “A user-friendly research tool for image processing with fuzzy rules”. *Proc. First IEEE Int. Conf. Fuzzy System*, Pág. 561-568. 1992.

[16] F. Russo and G. Ramponi. “Edge extraction by FIRE operators”. *Proc. of the Third IEEE Int. Conf. on Fuzzy Systems*, Pág. 249-253, 1994.