UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN

APLICACIÓN DE LA LÓGICA DIFUSA AL CONTROL DE PRESIÓN Y TEMPERATURA EN UN SISTEMA DE AIRE ACONDICIONADO, DE UN EDIFICIO INTELIGENTE

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERA MECANICA ELECTRICISTA

P R E S E N T A

CAROLINA GUTIERREZ BARCENAS

ASESOR: ING. JORGE DE LA CRUZ TREJO

CUAUTITLAN IZCALLI, EDO. DE MEX. **2003**

TESIS
CON
FALLA DE
ORIGEN

PAGINACION

DISCONTINUA

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN
UNIDAD DE LA ADMINISTRACION ESCOLAR
DEPARTAMENTO DE EXAMENES PROFESIONALES

U. N. A. M.

ASUNTO: VOTOS APROBATORIOS

DEPARTAMENTO DE EXAMENES PROFESIONALES

DR. JUAN ANTONIO MONTARAZ CRESPO
DIRECTOR DE LA FES CUAUTITLAN
P R E S E N T E

ATN: Q. Ma. del Carmen García Mijares
Jefe del Departamento de Exámenes
Profesionales de la FES Cuautitlán

Con base en el art. 28 del Reglamento General de Exámenes, nos permitimos comunicar a usted que revisamos la TESIS:

"Aplicación de la Lógica Difusa al Control de Presión y Temperatura en un Sistema de Aire Acondicionado, de un Edificio Inteligente".

que presenta la pasante: Carolina Gutiérrez Bárcenas
con número de cuenta: 8812067-0 para obtener el título de :
Ingeniera Mecánica Electricista

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el EXAMEN PROFESIONAL correspondiente, otorgamos nuestro VOTO APROBATORIO.

A T E N T A M E N T E
"POR MI RAZA HABLARA EL ESPIRITU"
Cuautitlán Izcalli, Méx. a 19 de Febrero de 2003

PRESIDENTE	Ing. Enrique Jiménez Ruíz	
VOCAL	Ing. Juan Rafael Garibay Bermúdez	
SECRETARIO	Ing. Jorge de la Cruz Trejo	
PRIMER SUPLENTE	Ing. Juan Contreras Espinosa	
SEGUNDO SUPLENTE	Dr. Armando Aguilar Márquez	

DEDICATORIA

A mis padres, Consuelo Bárcenas y Antonio Gutiérrez, gracias por la vida que me han dado y por tanto amor, sacrificio y apoyo. Por el gran trabajo hecho con todos nosotros y por proporcionarme el mejor de los legados: mi educación.

A Omar David, me enseñaste que la vida es más que solo vivir y dejarla pasar, a darle un valor real y apreciar las personas y cosas, eres el mejor de los hombres, la felicidad y las ganas de ser mejor cada día, gracias por todo tu apoyo, y tu amor incondicional, siempre que pensé que la Ingeniería no era para mi, dijiste lo contrario y me consolaste y gracias por ser ante todo un buen amigo....Te amo.

A Maria Luisa, mi madre, que siempre has cuidado de mi con cariño y dedicación.

A Anita, mi hermana y compañera de toda la vida.

A mis hermanos, María de la Luz, Aurora, Antonia, Guadalupe y Jaime, mi familia y la base de mi vida.

A todos mis niños, Nati, Con, Irma, Migue, Lalito, Arturo, Blanca, Armando, Chucho, Griselda, Ricardo, Jessica, José Antonio y Alma, que son mi alegría.

AGRADECIMIENTOS

Al Ing. Jorge de la Cruz, por su atención y ayuda para la conclusión de mis estudios, así mismo por la revisión y las facilidades ofrecidas durante la realización de esta tesis.

A los profesores Francisco Rojas, Filiberto Leiva y José Antonio Sánchez, por que la UNAM seguirá siendo grande por profesores como ellos.

A Aurelio González, quien además de ser un gran amigo fue un profesor para mi, gracias por ser tan paciente "Inge".

A Eloisa Moreno, por ser la mejor de las amigas y enseñarme a valorarme como persona y como profesionista.

A Teodoro Melo, por ser tan especial, el mejor de los amigos y por tantos buenos consejos.

A Alejandro Chávez, Mario Alberto Dorantes y Miguel Ricalde, por ser unos excelentes amigos y compañeros.

A Julia Jiménez y Enrique Becerril, gracias por tantos años juntos.

INDICE

INTRODUCCIÓN.

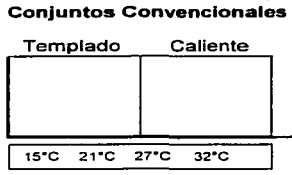
Un edificio inteligente es aquel que cuenta con un sistema administrador de recursos, el cual es una conjunción de elementos de software y hardware dedicados a supervisar una red de controladores inteligentes dirigido a:

1. Lograr el óptimo uso de los recursos y servicios con los que cuenta un edificio, procurando la máxima comodidad.
2. seguridad y plena satisfacción de todos sus ocupantes, al menor costo de operación posible.

Uno de los principales servicios que se controlan mediante esta red es el aire acondicionado. El software que se utiliza actualmente cuenta con una programación que hace que un controlador sea capaz de manipular las condiciones o variables principales: temperatura y presión; obviamente, dicho sistema tiene que estar sujeto a las órdenes de un operador, el cual indicará al software las acciones a seguir. Dicho operador suele tener cierto costo, además de errores al comandar el sistema, por lo que la integración de una red en un edificio inteligente no termina con una inversión inicial si no que se invierte también en gastos de operación, que a la larga conllevan a una inversión mucho mayor que la inicial.

Pensando en eliminar gastos y evitar errores de operación, se muestra el presente trabajo; el cual mediante una innovadora y exacta forma de control pretende monitorear y dar instrucciones a un sistema electromecánico para obtener la presión y la temperatura requeridas por el usuario. Esta nueva forma de control es llamada *Lógica Difusa* y es una rama de la inteligencia artificial gracias a la cual las computadoras pueden diluir el blanco y negro de la lógica ordinaria.

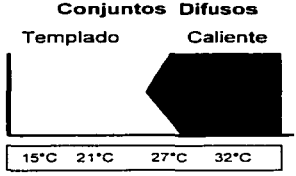
Por ejemplo, si se habla de 27°C, ¿Se está refiriendo a una temperatura caliente o templada?. En la Lógica Difusa, y en la forma que se clasificaría esta temperatura, podría decirse que se trata de una temperatura "*medio templada y medio caliente*".



Metodología Convencional

Esta respuesta representada en un conjunto convencional muestra que la transición entre el conjunto templado y caliente se da de manera instantánea (un elemento es miembro de un conjunto o no lo es).

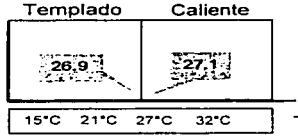
Sin embargo, la respuesta en un conjunto difuso, la transición se da de un manera gradual (un elemento puede pertenecer en parte a un conjunto).



Metodología Difusa

En la lógica clásica, usando los conjuntos convencionales, 26.9 °C puede ser clasificado como templado y 27.1°C puede ser clasificado como caliente, pequeños cambios de temperatura pueden causar variaciones significantes en el sistema.

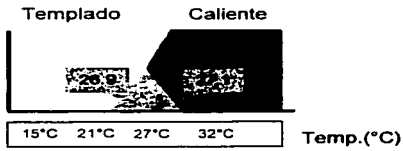
Conjuntos Convencionales



Aquí 29.9 y 27.1 pertenecen a dos conjuntos diferentes

En un sistema difuso, pequeños cambios en la temperatura causan un cambio gradual en el comportamiento del sistema.

Conjuntos Difusos



Aquí 29.9 y 27.1 pertenecen al mismo conjunto

La lógica difusa no solo reconoce las alternativas del blanco y negro de la lógica tradicional, sino también la infinita gradación entre ellos.

El algoritmo de control difuso consiste en escalar la señal de entrada de la variable a controlar y la asignación a sus correspondientes regiones de equivalencia lingüística; se busca en la base de conocimientos las reglas que se pueden aplicar a las condiciones prevalecientes del estado del sistema; se determina el grado de pertenencia de cada antecedente de las reglas y su efecto en las partes consecuentes de cada regla.

Por último, se hace la combinación de los consecuentes de las reglas para fijar el valor de salida.

El control difuso ofrece amplias posibilidades para mejorar el control de procesos industriales. En la medida que los sistemas de cómputo sean más versátiles y baratos y se tenga mayor aceptación del control inteligente, se podrán cumplir las exigencias del control de procesos por muy complejos que sean.

ANTECEDENTES

A continuación se detalla el funcionamiento del control convencional que opera actualmente en un Edificio Inteligente:

Existe un recinto donde se genera calor por diversas causas, ya sea por la presencia de personas, iluminación, diversas máquinas, ganancia térmica por insolación, etc., en el cual se pretende inyectar aire frío para abatir el incremento de temperatura hasta llegar a un valor deseado.

La inyección de Aire Acondicionado se lleva a cabo mediante un sistema conectado a una unidad central (PC), en la cual se llevan a cabo la totalidad de cambios en el sistema mediante un operador.

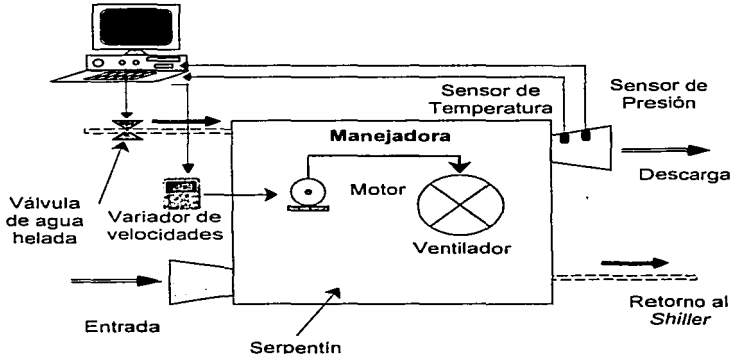


Diagrama de funcionamiento mediante control convencional

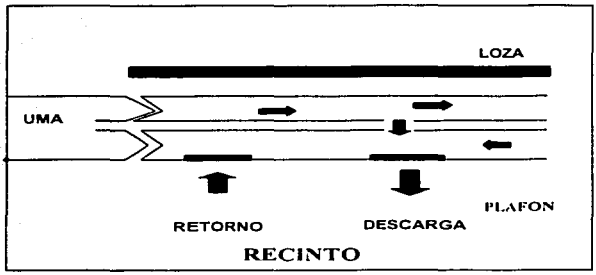
La unidad central va ordenar al sistema dos acciones de suma importancia, las cuales van a determinar la presión y temperatura a la descarga:

1. La velocidad a la que va a girar el motor mediante un variador de velocidades.
2. Un porcentaje de apertura en la válvula de agua helada.

Estos dos parámetros van a establecer las condiciones finales del sistema.

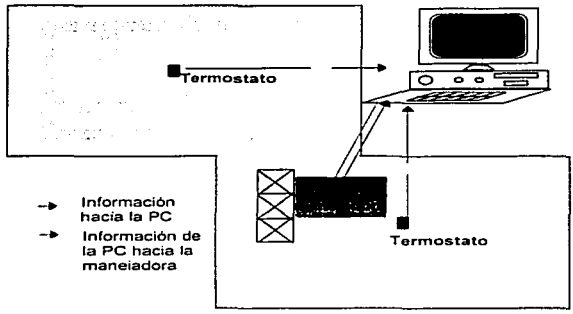
Según el diagrama, el aire con ganancia térmica entra por el serpentín el cual funciona como intercambiador de calor debido a la temperatura (7 u 8°C) del agua que circula a través de el, la cantidad de flujo determinará la apertura de la válvula de agua helada, el aire es succionado y descargado por el ventilador ubicado dentro de la manejadora, en el ducto de descarga existen dos sensores uno de presión y otro de temperatura conectados a la unidad central, las mediciones realizadas por ambos sensores reportan lecturas a la PC y esta sigue variando la velocidad en el variador y la apertura de la válvula hasta alcanzar las condiciones programadas, el agua utilizada en el sistema recircula hacia la unidad enfriadora de agua helada (shiller), en la cual por procesos de refrigeración recupera la temperatura que perdió en la manejadora, de esta manera el agua lleva a cabo un proceso de circulo cerrado.

El aire inyectado tiene una temperatura menor que la censada en ese momento dentro del recinto. Por convección el aire caliente sube a las partes más altas y pasa por el interior del plafón(cámara) en forma natural por medio de rejillas de retorno, en forma forzada succionado por ventiladores de extracción o bien a través del sistema de succión de la misma Unidad manejadora de aire, para bajar su temperatura y volver a ser inyectado.



Circulación de aire Acondicionado

Dentro de las oficinas existen termostatos conectados a la PC, los cuales al igual que los sensores ubicados en la descarga de la manejadora envían constantes lecturas de temperatura, de esta manera el operador tiene una visión del comportamiento del sistema, dicha información va a determinar si los parámetros seleccionados son los adecuados o el operador necesita modificarlos nuevamente.



Distribución de ducteria de Aire Acondicionado, e intercambio de información dentro del sistema.

Los cambios en los valores de temperatura y presión son realizados de acuerdo a las necesidades del usuario, sin embargo en ocasiones suele suceder que los valores actuales son totalmente opuestos a los requeridos, obviamente al tratar el sistema de proporcionar estas condiciones ocasiona malestares e incomodidad en los usuarios, lo cual es muy grave ya que el sistema que opera actualmente fue diseñado para proporcionar el máximo de comodidad.

El sistema que se maneja en este tipo de edificios tiene un costo inicial considerable, posterior a este costo se invierte en personal capacitado para operar y monitorear el sistema, el cual como ya se ha mencionado anteriormente representa un costo elevado y continuo.

El presente trabajo, aplicando el control con Lógica Difusa pretende corregir este tipo de problemas, ya que esta nueva forma de programar reconoce muchas más alternativas que el control tradicional, gracias a esto los cambios se llevaran a cabo de manera gradual y en forma autónoma.

CAPITULO I

LÓGICA DIFUSA

Control, desde el punto de vista de la Ingeniería es el proceso que causa que una variable del sistema tienda hacia un valor deseado denominado valor de referencia. A la aplicación de la lógica difusa al control de procesos se le dió el nombre de control difuso.

En el caso de sistemas con cierta complejidad el razonamiento difuso proveé una forma de comprender el funcionamiento del sistema permitiendo interpolar aproximaciones entre las entradas observadas y las situaciones de salida. Si bien el control difuso no basa su operación en un aparato matemático riguroso, sí es posible definir los principales conceptos de su metodología y elementos a través de la presentación de cierta nomenclatura matemática, basada en la comparación entre la lógica tradicional y la difusa:

En la teoría clásica o convencional de conjuntos, el conjunto S, está definido por una función f_S, llamada la *función característica de* S. f_S acota los elementos de S a uno (verdadero) o cero (falso), como se muestra en la expresión siguiente:

$$f_S : S \rightarrow \{0,1\}$$

donde,

$$f_S(x) = \begin{cases} 1, si...x \in S \\ 0, si...x \notin S \end{cases}$$

Por lo tanto, cualquier elemento x de S: $f_S(x)=1$, si x es un elemento de S; y $f_S(x)=0$, si x no es un elemento de S. En contraste, en la teoría de los conjuntos

difusos, el conjunto S está definido por una función μ_S, llamada *la función de membresía de* S.

μ_S acota los elementos de S a un valor entre cero y uno:

$f_S : S \rightarrow [0,1]$

Si x es un elemento de S, μ_S es el grado de pertenencia de x a S:

$\mu_S(x) = 1$, indica que x está contenido totalmente en S.

$\mu_S(x) = 0$, indica que x no está en S.

$0 < \mu_S(x) < 1$, indica que x está parcialmente en S.

Funciones con las que es posible visualizar el carácter con el que la lógica difusa toma la correspondencia entre las variables de un sistema y su grado de evaluación. Operaciones que son características y definen la naturaleza de ésta técnica de control.

En el lado práctico, el uso de la lógica difusa para determinar una solución rígida a un problema determinado, generalmente se guía por tres pasos: **Fusificación, Inferencia y Defusificación.**

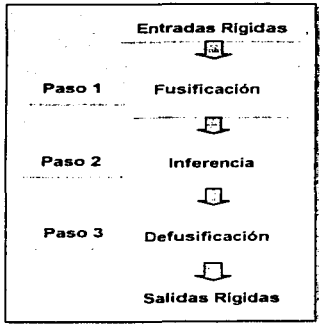


Fig.1.1 Etapas en el proceso con lógica difusa.

Es necesario para el diseño de cualquier proceso, describir el sistema en conjunto (ambas partes: la difusa y la no difusa). Específicamente, es necesario identificar que variables entran al sistema y cuáles salen de éste.

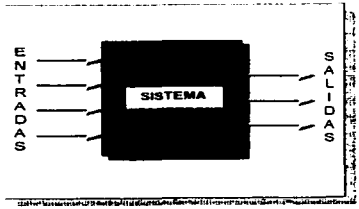


Fig.1.2 Identificación de entradas y salidas.

Para evaluar la respuesta del sistema es necesario determinar el error e existente entre señales proporcionadas por un sistema en general, el error es igual a:

$$e = S_{ref} - S_m$$

donde,

S_{ref} es la señal de referencia.

S_m es la señal medida.

Por otra parte, también es necesario conocer la tendencia del error, es decir, determinar si las compensaciones que efectúa el control están alejando o acercando la respuesta del sistema a la señal de referencia. Tal tendencia está dada por la variación entre la toma sucesiva de dos muestras y se define como Δe.

$$\Delta e = S_{m-1} - S_m$$

donde,

S_m es el valor actual de la señal y

S_{m-1} es la señal medida anterior.

El error e y la variación del error Δe pueden tomar tanto valores negativos como positivos. Si e es negativo, significa que la señal está por arriba de la señal de referencia; si es positivo, está por debajo. En el caso del Δe, si éste es negativo, indica que el valor de la señal actual leído es mayor al de la lectura anterior; en caso contrario, si Δe es positivo, el valor de la lectura actual ha disminuido respecto a la lectura anterior. Para que el sistema haga un adecuado seguimiento a la señal tomada como referencia el proceso de control debe tener en cuenta ambas variables;

es decir, debe conseguir llevarlas a su valor mínimo. La posición del mecanismo es óptimo cuando ambos, el error e y la variación del error Δe, son iguales a cero.

Ya determinados el error e y la variación del error Δe, éstas serán las entradas rígidas al algoritmo difuso de control.

Una vez identificadas las entradas al sistema, es necesario definir la salida de éste, la cual depende del tipo de proceso que se este manejando, ya que los sensores o el equipo acoplado con la programación difusa puede accionarse o ser sensible a diferentes lecturas de Presión, Tiempo, Temperatura, PMW, etc.

1.1. FUSIFICACIÓN.

El primer paso en el proceso difuso involucra una transformación de dominio llamada fusificación. Las entradas rígidas o bien definidas son transformadas en entradas difusas.

La fusificación de las entradas, es el proceso por el cual se calcula su grado de membresía la cual puede pertenecer a uno o a varios de los conjuntos involucrados En este paso se denomina a los rangos de entrada, como **entradas difusas**, con su respectiva membresía, donde cada uno de los valores se encuentra dentro de un conjunto universo denominado **"universo de discurso".**

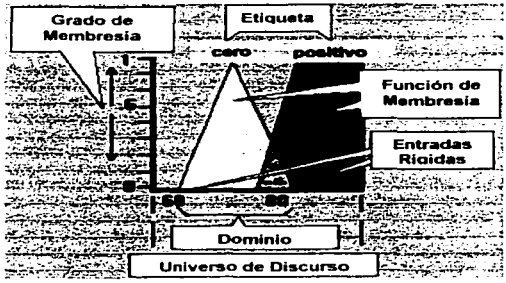


Fig. 1.3 Nomenclatura de conjuntos difusos.

El universo de discurso es el rango de interés de la aplicación en cuestión, en el cual se distribuirán los conjuntos difusos. Cada uno de los valores se encuentra en el universo de discurso teniendo un grado de membresía, con su correspondiente etiqueta difusa.

1. 1. 1. Cálculo del grado de membresía.

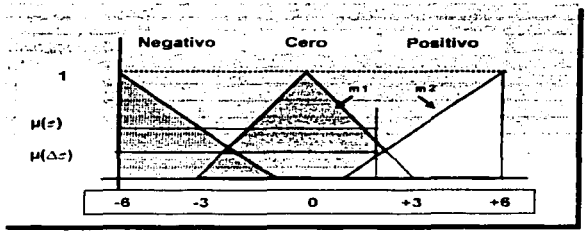


Fig. 1.4. Representación grafica del calculo del grado de membresía

Para obtener el grado de membresía de un valor de entrada, primero se calculan las pendientes de las rectas involucradas con la ecuación de la recta, $Y - Y_0 = m(X - X_0)$ y-y_o. utilizando la misma ecuación se calculan los puntos $\mu(e)$ y $\mu(\Delta e)$ para determinar el grado de membresía, obteniéndose las siguientes ecuaciones:

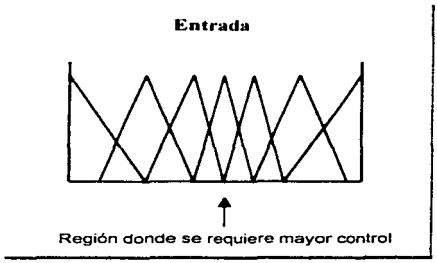
$$\mu(\ell) = m_1(X_1 - X_0) + Y_0$$

$$\mu(\Delta\ell) = Y_1 - m_2(X_1 - X_0)$$

1. 1. 2. Determinación del número y distribución de las funciones de membresía.

Número de funciones de membresía.

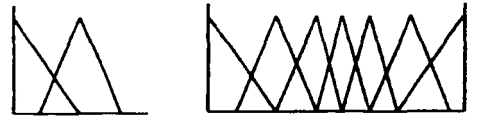
Existen algunos factores para determinar el número y las características de las funciones de membresía a emplear.



El número de funciones de membresía que se emplean usualmente son un número impar, y el número máximo recomendado es 9 (3,5,7,9). Esto no quiere decir que no se pueda utilizar un número par de funciones de membresía, simplemente se recomienda por simetría el utilizar un número impar de conjuntos.

Así mismo como método práctico, se utiliza una mayor densidad de funciones de membresía en el lugar donde se requiere un mayor control.

Si se tienen pocas funciones de membresía para una aplicación dada, la respuesta podría ser muy lenta y la respuesta del control a la salida se vería atrasada para compensar pequeños cambios en la entrada.



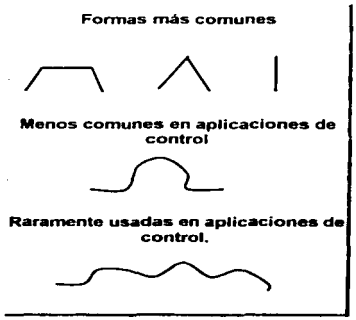
Esto podría causar que el sistema oscile alrededor de un valor deseado. El emplear un gran número de funciones de membresía provocaría la activación de diferentes reglas a pequeños cambios en los valores de entrada, dando por resultado grandes cambios a la salida lo que originaría inestabilidad en el sistema.

En un sistema de control difuso, el grado de traslapamiento entre las funciones de membresía repercute en el desempeño del sistema. Un sistema difuso sin traslapamiento entre sus conjuntos, se reduce a un sistema basado en la lógica boleana.

1. 1. 3. Definición de la forma de las funciones de membresía

Las formas más comunes para las funciones de membresía son trapecios, triángulos y singletons.

Se utilizan por facilidad en su representación y el bajo costo de la implementación del hardware.



No es frecuente utilizar formas complicadas para las funciones de membresía, pero si este fuera el caso, se necesitaría la implementación de hardware acompañado por tablas de consulta en software que representen estas formas complicadas. Estas formas son comúnmente empleadas en problemas estadísticos ó de predicción, rara vez en aplicaciones de control.

1.2. REGLAS DE INFERENCIA.

El segundo paso consiste en la construcción de reglas lingüísticas para determinar el grado de control del sistema. A esto se le conoce como *REGLAS DE INFERENCIA*.

Se utilizan reglas lingüísticas para determinar la acción del control en respuesta a un conjunto de valores de entrada, estas reglas son usualmente del tipo *si- entonces*. La sintaxis de estas reglas es de la siguiente forma:

Donde Y (AND) es uno de los operadores lógicos difusos permitidos, aunque también podríamos utilizar operador lógico (OR).

La primera parte de la regla (***Si***), se denomina antecedente, y contiene una o varias condiciones referidas, así como cada una de las entradas del sistema pertenece a tal o cual conjunto difuso. La segunda parte (***entonces***), denominada consecuente, contiene los nombres de los conjuntos difusos a los que deben pertenecer las salidas del sistema, si se cumple el antecedente correspondiente. La estructura de las reglas está confinada a un predefinido conjuntos de términos lingüísticos y a una estricta sintaxis.

Para la construcción de reglas nos basaremos en la siguiente tabla:

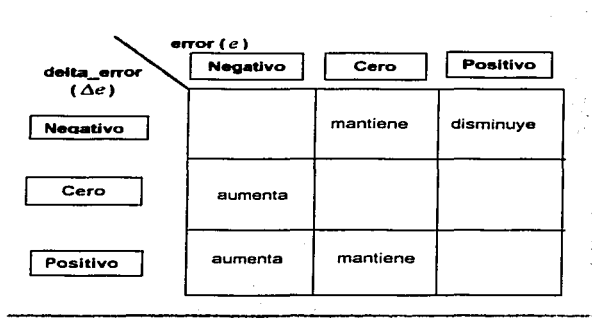


Fig. 1.5. Matriz de memoria Asociativa Difusa (MAD)

La construcción de la tabla se establece bajo las condiciones de trabajo del sistema. Las reglas siguen el comportamiento del sistema y son escritos en términos de las etiquetas lingüísticas de las funciones de membresía.

Para la escritura de las reglas, primero se debe codificar la información que describe el comportamiento del sistema. Las reglas representan la base de conocimientos de nuestra aplicación.

La primera fase en la escritura de reglas involucra la descripción de todas las reglas que son prácticas para describir un sistema.

Específicamente, se tienen que seguir los siguientes pasos:

- Primero describir todas las reglas que son obvias.

- Después describir las reglas que son menos obvias, sin embargo intuitivamente correctas, evitando transiciones abruptas entre celdas adyacentes.

Fase I: Codificación de la regla

- Escritura de las reglas obvias

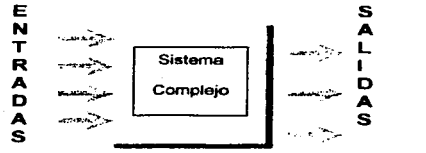
- Escritura de las reglas menos obvias

Fase II: Sintonización de la Regla

Después de que se ha observado y simulado el modelo, se está preparando para la segunda fase de la escritura de las reglas, la sintonización. Las reglas son "sintonizadas" individualmente para que su contribución en la superficie de control refleje con más precisión el comportamiento deseado.

Finalmente las reglas son "sintonizadas" individualmente para que su contribución en la superficie de control refleje con más precisión el comportamiento deseado Sistemas complejos con múltiples entradas y salidas requerirán de un número mayor de matrices para describir todas las posibles combinaciones de entradas y salidas. En este caso se podrá abordar el problema de la siguiente manera:

- Describir las reglas que están bien definidas.
- Describir las reglas que son vagas, sin embargo intuitivamente correctas.



Si cada entrada tiene 5 conjuntos el número de reglas posibles será:
$5^4 \times 3 = 1875$
En la práctica, el número de reglas no serán más de 20 o 40.

El número de reglas eventuales instaladas en un controlador difuso, regularmente es menor que el número total de reglas posibles esto es especialmente cierto en sistemas con muchas entradas, salidas y conjuntos difusos.

1. 2. 1. Sintonización de las reglas

Para modificar el comportamiento del sistema es necesario sintonizar cada una de las reglas individualmente. Para tener una mayor precisión en el comportamiento del sistema a lo esperado, se deben considerar las siguientes características que podrán ser sintonizadas.

- Cortes Alfa
- Pesos de contribución
- Operadores de compensación.

La forma de seleccionar cual de las características anteriores deberá ser sintonizada, lo determinará el comportamiento observado en la salida del sistema.

NOTA: No todos los ambientes de desarrollo soportan estas técnicas.

Si en la salida del sistema muestra pequeñas cantidades de ruido en áreas donde se debería esperar una respuesta plana, en ocasiones se debe al "disparo" de algunas reglas con valor de verdad muy pequeño.

Se puede corregir este error con ayuda del corte alfa por arriba del nivel del ruido.

Un tipo de corte es el llamado corte alfa del conjunto difuso, este tipo de corte se aplica a cada conjunto difuso individualmente.

Cuando utilizar los cortes alfa.

Los cortes alfa son inapropiados para muchas aplicaciones de control en los que se emplean microcontroladores.

Los cortes alfa pueden crear una respuesta de salida escalonada, esto repercutirá en la eficiencia de todo el sistema .

Los cortes alfa son útiles cuando se trabaja con funciones de membresía con límites grandes; los límites grandes provocan que muchas reglas tengan un valor de verdad muy pequeño, a su vez, estos pequeños valores pueden causar ruido en la salida del sistema, así como un incremento en la etapa de inferencias.

La mayoría de las implementaciones con microcontroladores utilizan funciones de membresía triangulares ó trapezoidales las cuales no poseen límites grandes en sus representaciones.

Pesos de contribución.

Cuando se desea que no todas las reglas tengan el mismo impacto en la superficie de control, se emplea la regla de peso. Se pueden encontrar espigas en la superficie de control que podrían ser causadas por una regla que solo maneje condiciones muy especiales, sin embargo, se dispara frecuentemente contribuyendo considerablemente en la superficie de salida. Para poder reducir el efecto de esta regla, se le puede dar un peso menor con respecto a las demás regla, esto reducirá su impacto en la superficie de control.

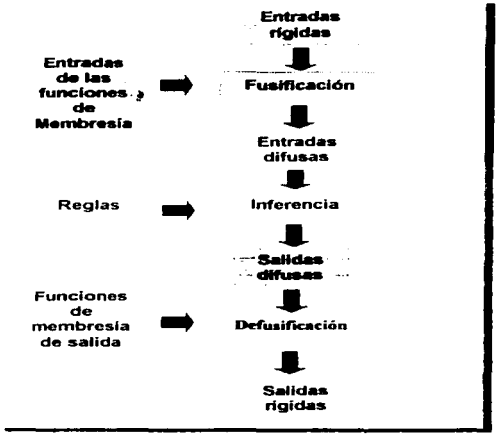
Los pesos de contribución también darán una extraordinaria prioridad a la regla seleccionada.

Operadores de compensación.

Los tipos de operadores de compensación empleados en el proceso de evolución de reglas impactan la forma en que la superficie de control es creada.

Un posible problema con el operador min/max es que si son escritas reglas con muchos antecedentes, el operador "y" mueve el valor de verdad a 0 y el operador "o" mueve el valor de verdad a 1. Si este tipo de comportamiento no es aceptable en nuestro sistema, tal vez un operador promedio o suma sería más apropiado

1. 3. DEFUSIFICACION



Al último paso del proceso de la lógica difusa se le llama *Defusificación*. En este proceso se combinan todas las salidas difusas formadas en la etapa de inferencia, para crear una salida con valor único que podrá ser aplicado a cada sistema de salida.

En este paso se convierten los valores fusificados a valores reales, una de las técnicas más comúnmente usados en el proceso de Defusificación es el llamado método del Centro de Gravedad (COG) ó método del centroide.

En este método, cada función de membresía de salida es truncada a su respectivo valor de salida difuso, se combinan las correspondientes funciones fragmentadas y se calcula el centro de gravedad del conjunto. Tal truncamiento de funciones se denomina *corte lambda*.

Fig. 1.6. Función de membresía truncada a su valor de salida difuso

El corte lambda ($\lambda - cut$) restringe el valor máximo de una región difusa ó función de membresía. Para cada μ_A

$$\mu_A(x) = max[\mu_A(x), \lambda - cut]$$

En teoría, para calcular el centro de gravedad de las funciones umbralizadas se debe considerar el análisis sobre todos los puntos continuos en el dominio de salida,

$$COG = \frac{\int_a^b \mu(x) \cdot x \cdot dx}{\int_a^b \mu(x) \cdot dx}$$

Sin embargo, es posible obtener una buena estimación haciendo el cálculo del COG sobre una muestra de puntos en el dominio de salida,

$$COG = \frac{\sum_{x=a}^{b} \mu(x) \cdot x}{\sum_{x=a}^{b} \mu(x)} \qquad \text{Ecuación 1.1}$$

Con el muestreo de puntos lo suficientemente grande para proveer una buena exactitud sin usar demasiado tiempo de procesamiento.

El método de Defusificación COG también puede aplicarse a las funciones de salida tipo singelton. Las funciones singelton son representadas por un solo punto en el

espacio de salida y por tanto tienen masa cero. La ventaja de adoptar el uso de funciones singelton en lugar de las funciones regulares, es que se requiere menos cálculos en su procesamiento de defusificación, sin embargo, éstas últimas proveen una salida más consistente con sistemas que requieren procesos de control más detallados.

Como ya se dijo, el método de defusificación COG, también se puede aplicar a las funciones de salida singleton. Una función de membresia de salida singleton es representada por un punto individual en el espacio de salida, en otras palabras, es un tipo especial de conjunto difuso que contiene solo un elemento en el conjunto.

Empleando el método de defusificación COG, los valores Singleton de salida son combinados utilizando un valor promedio. La formula del COG para el calculo de los conjuntos singleton se reduce a:

$$Salida\_Rigida = \frac{\sum_i (Salida\_difusai) * (Posicion\_singleton\_en\_eje\_x_i)}{\sum_i (Salida\_difusa_i)}$$

Es importante apreciar que el cálculo de la defusificación utilizando conjuntos singleton es significativamente menor que a otros métodos en los que se necesita mayor número de cálculos matemáticos para obtener el centro de gravedad.

Existen ventajas y desventajas en adoptar los conjuntos difusos regulares o los conjuntos singleton para representar una variable de salida. Durante la defusificación utilizando el COG, los conjuntos singleton requieren menos cálculos, pero el no

utilizar este tipo de conjuntos se puede esperar una salida más cercana a las expectativas del sistema de control.

Método de Defusificación	*Características de la Salida Rígida*
COG(con funciones de membresía de salida singleton)	Requerimiento de memoria(1) Velocidad(59 Algunas fluctuaciones en la salida
COG(sin utilizar funciones de salida singleton)	Requerimiento de memoria(5) Velocidad(1) Salida suave
Máximo Izquierdo	Requerimiento de memoria(2) Velocidad(4) Salida más o menos confiable
Máximo Derecho	Requerimiento de memoria(2) Velocidad(3) Salida muy confiable
Promedio del máximo	Requerimiento de memoria(4) Velocidad(4) Salida confiable
Punto medio del máximo	Requerimiento de memoria(3) Velocidad(4) Salida confiable
Media	Requerimiento de memoria(5) Velocidad(1) Salida suave

Nota: requerimiento de memoria (1) = bajo y (5)= alto
Velocidad de cálculo (1) = lenta y (5) = rápido

Los diferentes métodos de defusificación dan origen a diversas características de un sistema.

1. 3. 1. Observación del comportamiento del modelo.

Inicio de la verificación. Sintonizar si es necesario.

En esta etapa se "ejecutará" el modelo de la siguiente manera:

- Sometiendo varias combinaciones de entrada.
- Observando la actividad asociada de salida.

Intencionalmente se tratara de "romper" el modelo escogiendo una gran variedad de datos de prueba a través del universo de discurso.

Las características del modelo que se necesitan verificar son:

Los valores de salida: Los valores defusificados devueltos por el modelo serán revisados para determinar su precisión.

Superficie de control: La graficación de las entradas y salidas dan como resultado una superficie de control. Un sistema con dos entradas y una salida.

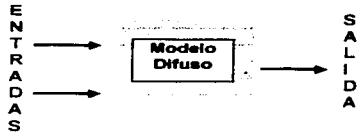
Tiempo de emulación: El comportamiento del modelo a través del tiempo debe ser observado para su sensibilidad y estabilidad.

Basado en nuestras observaciones, se podrá regresar a los anteriores pasos para sintonizar si es necesario, las funciones de membresía, las reglas y/o el método de defusificación.

La aproximación del comportamiento del modelo planteado al deseado, dependerá del tipo de herramienta que se utilice.

Examinar los Resultados de Salida.

Ahora, es necesario colocar a las entradas valores que produzcan un resultado conocido a la salida. Esto nos permitirá verificar que el modelo genere salidas que tengan sentido bajo condiciones conocidas. Para cada combinación de entradas, verificar que:



Revisar:

Salidas con sentido

Las reglas identifican el comportamiento deseado

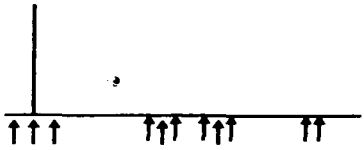
Los conjuntos tengan la forma y el traslape correctos

1) las salidas tengan sentido, si no es el caso, identificar las reglas y las funciones de membresía que causan esta salida errónea.

2) La(s) regla(s) identifica(n) correctamente el comportamiento deseado

3) Los conjuntos para los valores de entrada asociados, tengan la forma y el traslape correctos.

Después de examinar algunos valores de entrada que a su vez nos generan valores bien conocidos a la salida, se tendrá que observar el comportamiento del sistema en todo el universo de discurso, teniendo cuidado en las siguientes regiones de entrada:



Nota: Para cada combinación verificar que la salida del modelo tenga sentido.

- Valores de entrada en los extremos del universo de discurso.
- Valores de entrada en los extremos de cada función de membresía.
- Valores de entrada en los correspondientes a la región de traslape de las funciones de membresía.
- Valores de entrada con algún grado de verdad y con el mayor número de antecedentes.

Se deberá verificar que la forma de la superficie de control esté dentro de los requerimientos. Si existe alguna inconformidad usualmente se debe a uno de estos problemas, en el orden indicado:

1) Reglas definidas incorrectamente.

2) Definidas incorrectamente las funciones de membresía.

3) Utilizar los operadores difusos inapropiados.

La otra anomalía en la superficie de control podría ser causada por el índice de incompatibilidad entre los conjuntos difusos de entrada y de salida.

El índice de compatibilidad es el máximo valor de verdad que se obtiene a la salida debido a un conjunto dado de entradas rígidas.

Algunas de las causas más comunes en la incompatibilidad de los modelos son:

- Los datos "caen" constantemente en los extremos de los conjuntos difusos

- Las funciones de membresía fueron incorrectamente dibujadas y/o traslapadas.
- La inapropiada selección de los operadores difusos.

Algunas soluciones para estos problemas serían:

- Incrementar la cantidad de conjuntos difusos en el área máxima información.
- Examinar y posiblemente modificar el universo de discurso.
- Si el problema fue el utilizar operadores "y" (AND) los cuales toman el valor mínimo de verdad de los antecedentes, seleccionar la otra alternativa de interpretación del operador "y" que es el producto.

1. 3. 2. Sistemas de Optimización para Plataformas Finales.

Análisis y particiones de los Sistemas de Control.

Definición de superficies de entrada y salida

Escritura de reglas

Observación del Comportamiento del modelo. Verificación y sintonía si es necesario.

Sistemas de Optimización para Plataformas Finales

Cuando es ejecutado el modelo difuso en un sistema final, se podrá determinar el tiempo que tardará el sistema en producir una nueva salida difusa a partir de nuevos valores de entrada. También podemos saber cuánta memoria será necesaria para implementar el modelo.

Dos puntos a considerar cuando se ejecuta el modelo a un sistema final son:

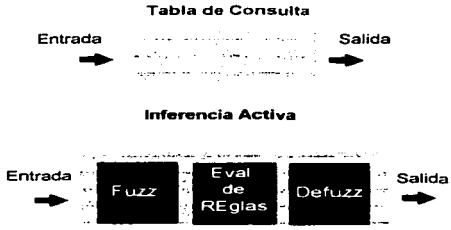
- Los requerimientos de memoria.
- El tiempo de respuesta

Los problemas que posiblemente se pueden encontrar son:

* Insuficiente memoria para nuestro modelo
* Muy lenta la ejecución del modelo

La arquitectura de nuestro procesador es inapropiada para las necesidades del modelo.

1. 3. 3. Implementación Esquemática.



Tal vez lo primero a considerar y que de esto dependerá los requerimientos de memoria y velocidad de ejecución, es el método de implementación.

Dos formas para implementar el modelo difuso a un sistema final son: utilizar una tabla de consulta ó emplear una inferencia activa

Tabla de consulta.

En la implementación de la tabla de consulta, el sistema difuso es simulado en una herramienta de alto nivel, posteriormente, es ejecutado con una gran cantidad de datos. La salida generada por cada valor de la entrada del modelo, es almacenada en una tabla de consulta dentro del sistema final.

Inferencia Activa.

Con la implementación de una inferencia activa de el sistema final obtendremos los procesos de fusificación, evaluación de reglas y defusificación en tiempo real. Un ejemplo de un generador de códigos de estas características sería la herramienta llamada **Kernel.**

Implementación de una Tabla de Consulta.

Tabla de Consulta

Variable 2

	Ent 1	.....		Ent m
Ent. 1	Sal. 1			Sal..m
Ent. m	Sal. m			

La implementación de una tabla de consulta generará un conjunto de celdas donde tendremos todas las posibles combinaciones de valores de entrada. Esto conduce a una mayor velocidad en la respuesta de nuestro sistema, sin embargo, la cantidad de memoria requerida es mayor.

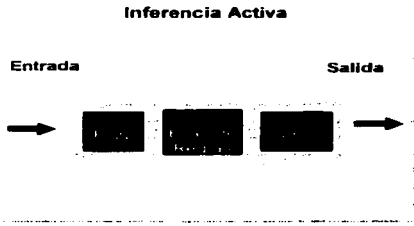
Los requerimientos de memoria se pueden reducir con solo almacenar algunos puntos importantes de la superficie de interpolarlos entre ellos.

Pero el emplear la interpolación, reducirá los detalles en la superficie de control y se decrementará el tiempo de las respuestas de nuestro sistema

Implementación de una Inferencia Activa.

En una implementación de una inferencia activa, las funciones de membresía, las reglas y la máquina de inferencias, están todas juntas en el sistema final y son ejecutados en tiempo real.

También en este tipo de implementaciones existe un compromiso entre los requerimientos de memoria y la velocidad de respuesta del sistema y esto dependerá de las opciones elegidas durante el diseño del sistema, por ejemplo:



- Almacenará las funciones de membresía en forma de tabla de consulta, requerirá mayor cantidad de memoria pero la velocidad del proceso de inferencia será mayor.
- Algunos operadores utilizados requieren de cálculos matemáticos más complejos en su solución que otros.
- En el proceso de defusificación utilizando la forma singleton, requiere de menos procesamiento que al utilizar los conjuntos difusos normales.

Implementación de Tablas de Consulta Contra La Inferencia Activa

Ventajas de las tablas de consulta

⇨ Mayor velocidad en el proceso de inferencia.

Ventajas de la inferencia Activa

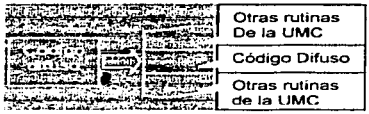
⇨ Sintonizar directamente en el sistema final.
⇨ Requerida por los sistemas difusos adaptados a los reales.
⇨ Práctico para las unidades de inferencia difusa compleja.
⇨ Excelente para los requerimientos bajos de memoria.

Se recomienda utilizar la inferencia activa, al menos que el tiempo requerido para el proceso de inferencia sea contraproducente en el sistema, por lo que la opción sería utilizar las tablas de consulta.

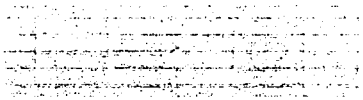
1. 3. 4. Sintonía.

A pesar de la implementación esquemática seleccionada, es deseable sintonizar el sistema difuso en tiempo real. Típicamente, los ambientes de simulación nos ayudarán para la determinación de las reglas, pero la sintonía en tiempo real solo se podrá hacer en el sistema final.

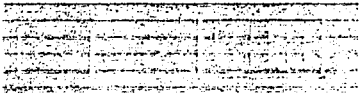
El Proceso de Sintonización



Combinar códigos difusos con otras rutinas de la UMC



Transferir a una memoria no volátil ó EPROM



Colocar el dispositivo programado dentro del sistema.

Una forma de probar el código difuso generado, es incluirlo en otras rutinas con códigos no difusos y transferir el código ejecutable a la unidad microcontroladora (UMC) en la tarjeta de control del sistema en tiempo real.

Si las funciones de membresía llegaran a ser modificadas para la depuración en la ejecución del sistema, debe ser necesario modificar el código. Se deberá borrar el código anterior trasferido y reemplazarlo por el nuevo código hacia el microcontrolador.

1. 3. 5. Consideraciones del Procesador.

Si la ejecución en tiempo real del modelo difuso no es satisfactorio se deberá revisar la elección del procesador. El procesador elegido influirá considerablemente en la resolución, velocidad y sensibilidad del modelo. El tamaño de la palabra del procesador determinará la granulidad de la función de membresía y consecuentemente la definición de la superficie de control.

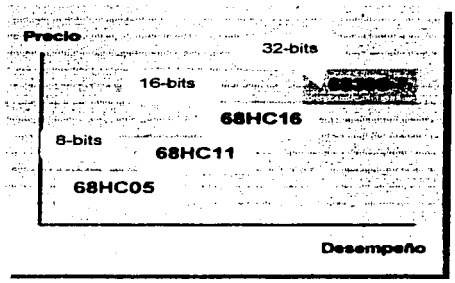
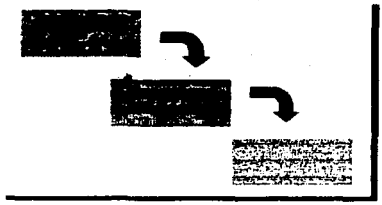


Fig.1.7. Comparación, precio, desempeño y velocidad de algunos procesadores

La capacidad de direccionamiento del procesador influirá en el arreglo y acceso de la estructura de datos. Modelos con una estructura de datos muy compleja requerirán los modelos de direccionamiento complejos. El conjunto de instrucciones y la velocidad del reloj del procesador influirá en la velocidad de la inferencia.

Este proceso puede llegar a ser laborioso y tardado, pero los emuladores utilizados en los microcontroladores han sido desarrollados para hacer esta tarea más eficiente.

1. 3. 6. Emulación.



La etapa de emulación es la forma más cercana al comportamiento del sistema en tiempo real. Los requerimientos generales para la emulación de un sistema son: Una computadora (PC), un programa especial (software) y el sistema final (hardware).

Es posible referirse como programa a las herramientas que son capaces de crear un ambiente que tienen la capacidad de interactuar con el usuario vía PC hacia el sistema final.

El emulador permite interactuar y ejecutar el programa paso a paso, también es posible revisar los registros de estado de microcontrolador, modificar contenidos de memoria y muchas otras características de depuración.

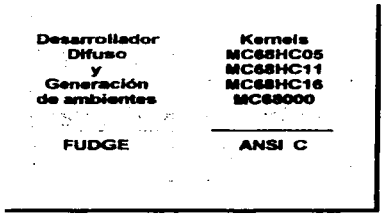
Se ha dicho que la emulación es la forma más cercana al comportamiento en tiempo real del sistema, debido a que el código del programa se ejecuta en memoria **RAM** de nuestro sistema haciendo más rápido el proceso de sintonización. Si fuera necesario corregir el programa, lo primero que se debe hacer es detener al microcontrolador, posteriormente se modifica el programa y es almacenado con la memoria RAM, finalmente se ejecuta éste.

Para poder observar el sistema en tiempo real, solamente se requiere grabar el programa en una memoria volátil como una memoria (EPROM), integrarla a nuestro sistema final y reinicializar todo.

CAPITULO 2

HERRAMIENTAS A UTILIZAR

Como parte de su esfuerzo para hacer de la lógica difusa una tecnología en uso Motorola provée un software libre, de lógica difusa a través del Freeware BBS.



El software consiste de le desarrollador difuso y generación de ambiente FUDGE(Fuzzi development and Generatión enviorment) y KERNELS para la familia de microcontroladores MC68000 y para aplicaciones ANSI C. estas herramientas ofrecen una plataforma de bajo costo para el desarrollo de aplicaciones de lógica difusa.

El FUDGE es usado para definir y probar las reglas y las funciones de membresía en aplicación.

El FUDGE entonces puede codificar la base de conocimientos para ser usadas con el correspondiente KERNEL e implementar una rutina de Lógica Difusa. Finalmente, estas rutinas en Lógica Difusa pueden ser usadas como parte del programa en la aplicación del usuario o como parte de un programa en ANSI C.

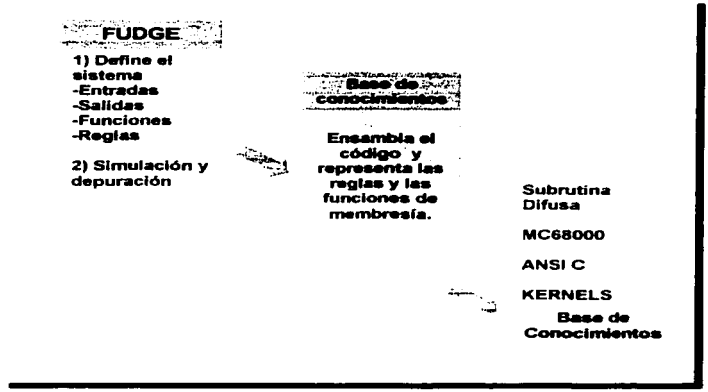
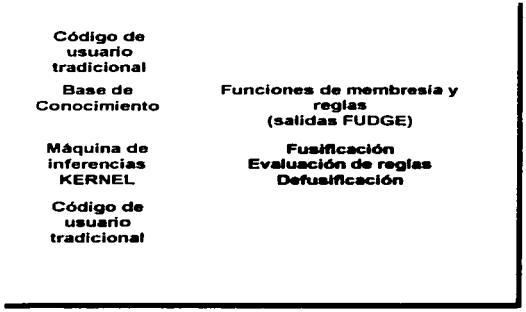


Fig. 2.1 Ruta de la información para obtener un programa de aplicación.



Un KERNEL de Lógica Difusa, también llamada Máquina de Inferencias es una rutina de código desarrollada para ejecutar la fusificación, la evaluación de reglas y la defusificación.

Cuando el KERNEL es concatenado con reglas especificas de aplicación y funciones de membresía el código resultante puede ser usado como una rutina lógica difusa dentro del código de ensamble del usuario.

2.1. FUDGE.

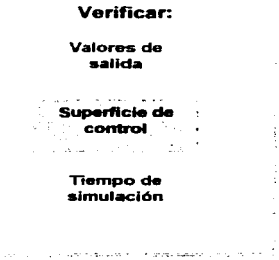
En esta sección se presenta detalladamente el programa para crear y simular el control difuso, el programa consiste de un ambiente de generación y desarrollo difuso, el cual es de gran utilidad gracias a que provée un valioso e interactivo ambiente gráfico para observar el comportamiento del modelo, aislando los problemas potenciales y permitiendo modificar los parámetros del modelo, ofreciendo un bajo costo en el desarrollo de aplicaciones con lógica difusa.

El FUDGE(Fuzzy Development and Generation Environment) es un ambiente de generación y desarrollo difuso que provée una amigable interfaz para el usuario, que le permite modificar reglas, funciones de membresía y además permite desplegar una gráfica de entrada / salida en dos dimensiones.

FUDGE es empleado para definir y examinar las reglas y las funciones de membresía de una aplicación en particular. Las funciones de membresía y las reglas son utilizadas por el programa FUDGE para generar la base de conocimientos en forma de código, que podrá ser utilizado por los microcontroladores de la familia MC68HCXX ó por alguna aplicación en donde se emplee el lenguaje de programación Turbo C, en particular en el desarrollo de la presente tesis utilizaremos como herramienta el lenguaje C.

El programa FUDGE tiene las siguientes características:

- *Hasta 8 variables de entrada*
- *Hasta 4 variables de salida*
- *8 funciones de membresía por cada variable de entrada*
- *8 funciones de membresía por cada variable de salida*
- *Permite hasta 1000 reglas*
- *Cualquier número de condiciones(antecedentes y consecuentes) por regla*
- *Hasta 40 caracteres por nombre (etiquetas de entrada y salida)*
- *Funciones de membresía de entradas triangulares y trapezoidales*
- *Funciones de membresía de salida singleton*



Las características del modelo que se necesitan verificar previamente son:

Los valores de salida: Los valores defusificados devueltos por el modelo serán revisados para determinar su precisión.

Superficie de control: La graficación de las entradas y salidas dan como resultado una superficie de control

Tiempo de emulación: El comportamiento del modelo a través del tiempo debe ser observado para su sensibilidad y estabilidad.

Basado en nuestras observaciones, se podrá regresar a los anteriores pasos para sintonizar si es necesario, las funciones de membresía, las reglas y/o el método de defusificación.

La aproximación del comportamiento del modelo al deseado, dependerá del tipo de herramienta que se utilice.

Ambiente de generación y desarrollo difuso (fudge)

En la figura 2.2 se muestran los comandos utilizados en el programa FUDGE dentro del menú FILE se tiene la opción de abrir un nuevo archivo (New) ó uno ya existente (Open), así como también salvar (Save as) el archivo en uso o simplemente cerrar el archivo sin hacer modificaciones(Close). También se puede crear un archivo de texto (Create Text File), el cual contiene toda la información de lo creado dentro del programa FUDGE.

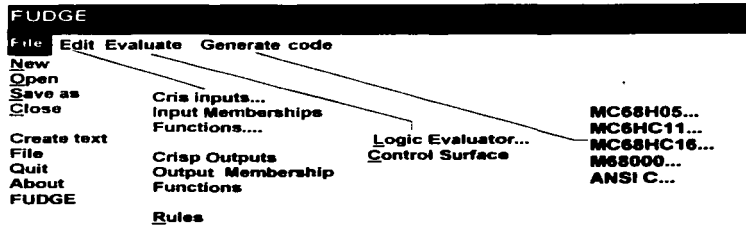
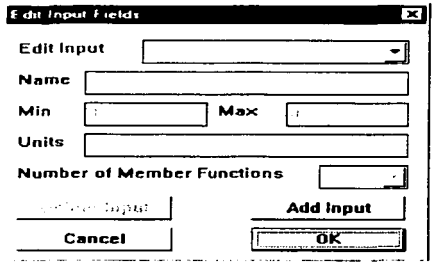


Fig. 2..2. Representación de los comandos y su respectivo menú del programa FUDGE

En el menú que hay dentro de EDIT se editan entradas y salidas del sistema, así como también las reglas creadas para controlar las variables.

Finalmente se crea el código en el cual se va a trabajar Generate Code (ensamblador o ANSI-C)

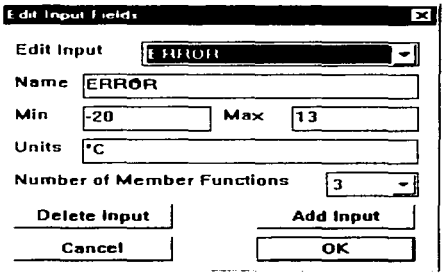
Para definir las entradas rígidas se deberá de elegir la opción (Crisp Inputs) del menú Edit. En esta ventana se especifica el nombre, valor mínimo, valor máximo, las unidades y el número de funciones de membresía asociadas con la entrada rígida. Dentro de la misma opción podremos utilizar el botón (Edit input) para poder seleccionar y editar cada una de las entradas del sistema.

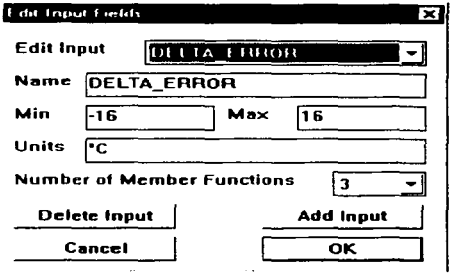


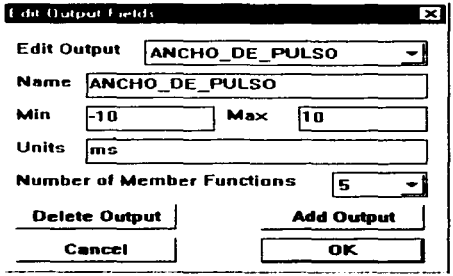
2.3 Ventana del menú Edit Imput, dentro del comando File

En la figura 2.4. Se observan dos variables de entrada y la variable de salida con su respectivo rango, unidad y número de funciones de membresía . Para agregar una entrada se presiona Add Input, posteriormente se le agregan los datos. Al finalizar se presiona el botón de OK. Para observar los datos se presiona nuevamente EDIT y al aparecer el menú se presiona (Cris Inputs). Presionando el

botón de Edit Input se observarán las variables con sus respectivos datos. Para agregar la variable de salida, el proceso se realizará de la misma manera.







2.4. Variables de entrada y salida, en el caso particular del presente trabajo es error, delta_error y ancho de pulso.

Después que se han definido las entradas rígidas se deberá definir las funciones de membresía o conjuntos difusos de entrada (Input Membership Funtions) del menú Edit podemos editar las etiquetas, formas y posiciones de las funciones de membresía. De esta misma forma son definidas las salidas rígidas junto con sus funciones de membresía figuras. 2.5, 2.6. y 2.7.

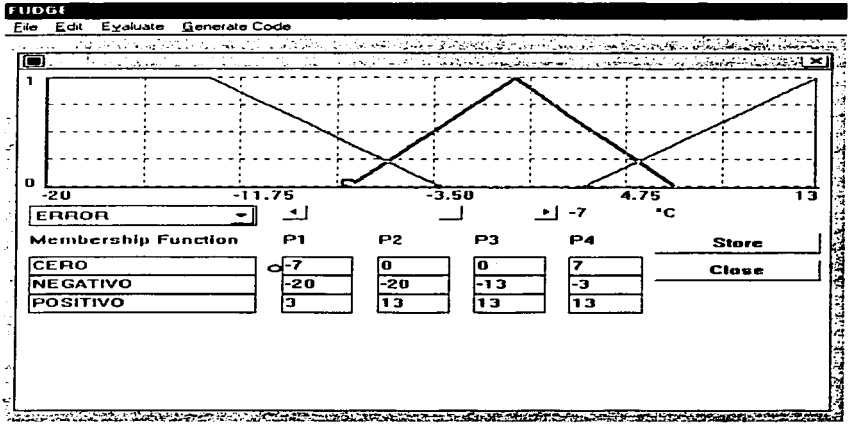


Fig. 2.5. Ventana Input Membership Funtions, con su respectiva variable de entrada (ERROR)

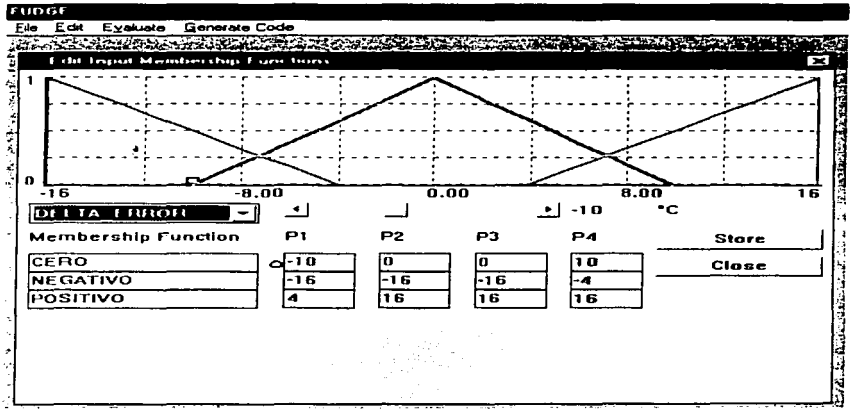


Fig. 2.6. Ventana Input Membership Funtions, Con su respectiva variable de entrada (DELTA ERROR)

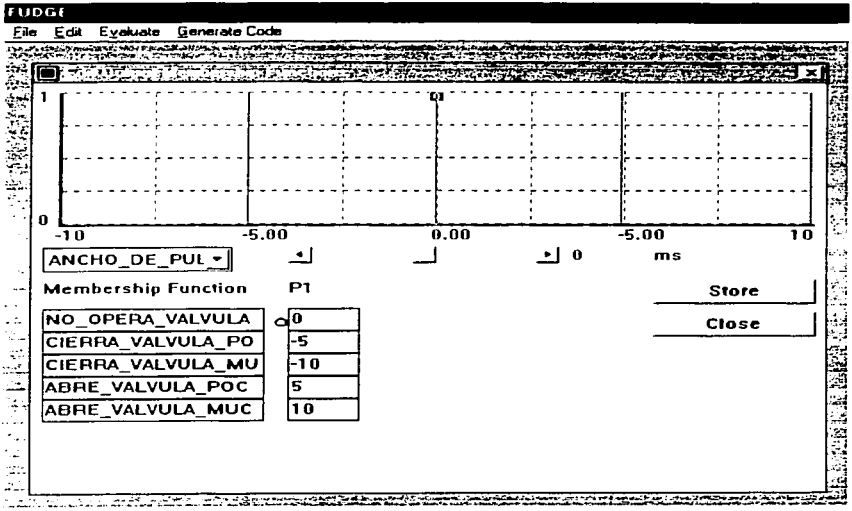


Fig. 2.7. Ventana Output Membership Functions, con su respectiva variable de salida (ANCHO DE PULSO)

Ya definidas las entradas y salidas rígidas con sus respectivas funciones de membresía, estaremos en condiciones de definir las reglas. Para esto, se selecciona la opción (Rules) del menú EDIT. En la ventana que aparece, se podrán observar las reglas ya definidas y se conseguirá con la opción Delete eliminar la regla no deseada, como se muestra en la fig. 2.8.

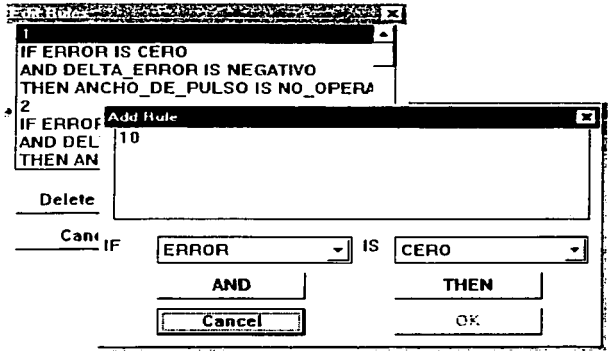


Fig.2.8. Ventana para la edición de reglas de la etapa de inferencia.

Para crear una regla de utiliza la opción Add Rule (Rules). Y estando en el menú Add Rule podemos escribir las reglas a partir de nuestras variables y funciones de membresía de entrada y salida.

La primera parte de la regla son los antecedentes que se forman en una variable de entrada y el nombre o etiqueta de un conjunto difuso, logrando combinar en los antecedentes más variables de entrada y sus funciones de membresía con la ayuda de la opción "AND" dentro de este mismo menú. La otra parte de la estructura de las reglas son los consecuentes, que se editan seleccionando la opción "THEN" dentro de este mismo menú. Los antecedentes se forman de

manera similar que los antecedentes, solo que se necesita una variable de salida y su correspondiente función de membresia en su estructura, fig. 2.9.

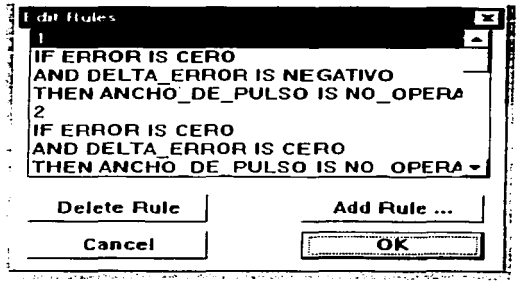


Fig. 2.9. Ventana para la edición de las reglas de la etapa de inferencia

Las opciones del menú Evaluate, se emplean para examinar el comportamiento de la base de conocimientos difusos. Dentro de este menú, la opción (Fuzzy Logic Evaluator) nos permite observar los efectos de la base de conocimientos en el proceso de inferencia difusa, para valores rígidos de entrada muy específicos.

Utilizar la opción (Fuzzy Logic Evaluator) podemos observar el comportamiento del sistema, a partir de un valor rígido seleccionado a la entrada. Variando el valor rígido podremos observar los efectos que se producen, en las funciones de membresia de entrada y salida, así como observar cuales de las reglas son "disparadas". El disparo de una regla significa que el valor de verdad de la regla es diferente de cero, fig. 2.10.

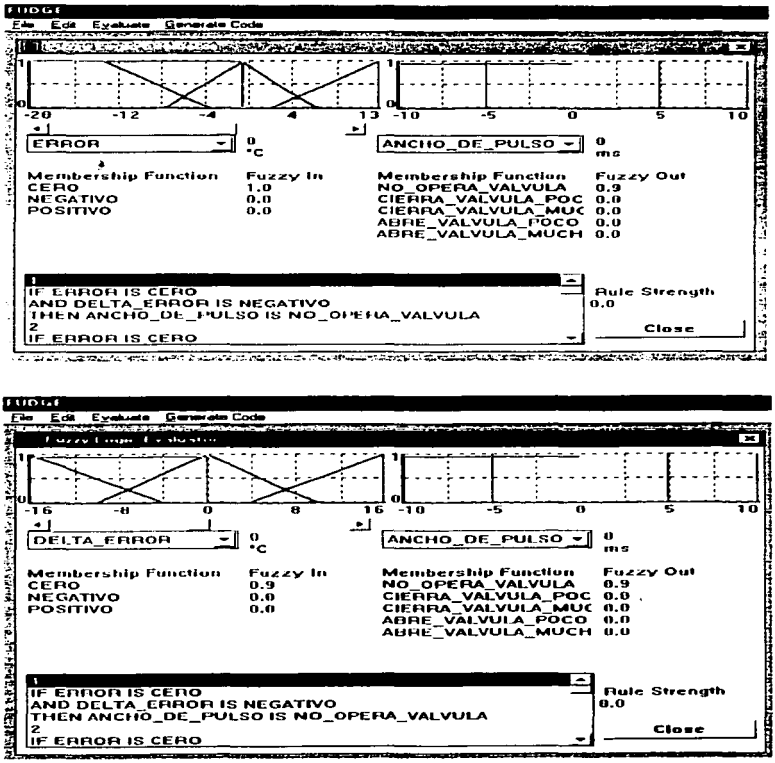


Fig. 2.10. Ventana donde se realiza la evaluación de las reglas de la etapa de inferencia, con las variables de entrada (ERRO Y DELTA ERROR)

Una vez satisfechos con los resultados, el programa FUDGE podrá generar la base de conocimientos requeridos por la máquina de inferencia (KERNEL).

Para guardar los datos introducidos en el programa FUDGE, se salva con el nombre que el usuario le de(8caracteres) con extensión FDG, esto se realizara en el menú de FILE.

Para este ejemplo el archivo lo denominamos:

Tesis_Ca.fdg

También se puede generar un archivo con extensión PRN, que contendrá un resumen de lo hecho en FUDGE, el archivo le denominamos en este caso:

Tesis_Ca.prn

2.2. MÁQUINA DE INFERENCIAS (KERNEL).

El último paso a seguir para la generación de un código difuso, es anexar el apropiado Kernel a la base de conocimientos generado por el programa FUDGE.

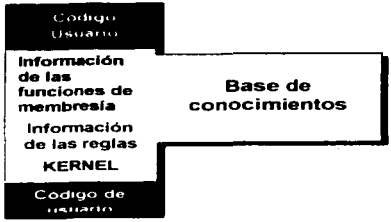
La máquina de inferencia difusa llamada KERNEL es un fragmento de código arreglado para poder realizar las etapas de fusificación, evaluación de reglas y defusificación de un sistema difuso. Cuando la máquina de inferencia es conectada con la base de conocimientos, el código resultante podrá ser utilizado como una rutina de lógica difusa dentro un programa con código ensamblador tradicional ó lenguaje C.

El trabajo manualmente requiere de compilar y verificar los modelos difusos paso a paso. A pesar de que es factible, obviamente es una alternativa que demanda mucho tiempo.

Varias generaciones de KERNELS para Lógica Difusa han sido desarrollados.

Existen KERNELS para uso en los microcontroladores de la familia 68000 de Motorola, así como algunos para aplicaciones en lenguajes de programación de alto nivel como el "C".

Los KERNELS y su asociada base de conocimientos están diseñados para ser integrados en el código de usuario.



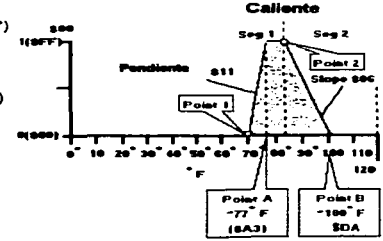
La base de conocimientos contiene las reglas y la información de las funciones de membresía. El KERNEL obtiene las entradas del sistema, aplica la información de la base de conocimientos (reglas y funciones de membresía) y calculan un valor de salida rígido usando las técnicas de inferencia min-max y la defusificación por el método del centro de gravedad (usando funciones de salida singletons).

$$BYTE\_1 = Punto\_1 = \$94 \quad (\sim 70°)$$

$$BYTE\_2 = Pendiente\_1 = \frac{\$FF}{\$A3 - \$94} = \$11 \quad (0.14 por°)$$

$$BYTE\_3 = Punto\_2 = \$AE \quad (\sim 82°)$$

$$BYTE\_4 = Pendiente\_2 = \frac{\$FF}{\$D4 - \$AE} = \$6 \quad (0.6 por°)$$



Representación de las funciones de membresía de entrada.

Las funciones de membresía usadas en los microcontroladores 68HC11 están representados pos 4 bytes cada uno.

Byte 1: Punto 1 rango ($00 a $FF)
Se asume que el punto 1 esta en x = punto 1
y = μ = 0 ($00)

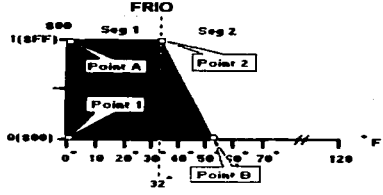
$BYTE\_1 = Punto\_1 = 0$ ($00)

$BYTE\_2 = Pendiente\_1 = 0$ ($00)

$(Clase\_especial)$

$BYTE\_3 = Punto\_2 = 32$ ($40)

$BYTE\_4 = Pendiente\_2 = \frac{\$FF}{\$68 - \$40}$ ($06)



La pendiente 1 siempre se asume como positiva, por lo tanto no tiene signo.

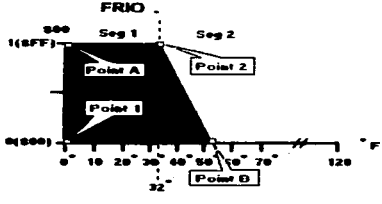
$BYTE\_1 = Punto\_1 = 0$ ($00)

$BYTE\_2 = Pendiente\_1$ ($00)

$(Clase\_especial1)$

$BYTE\_3 = Punto\_2 = 32$ ($40)

$BYTE\_4 = Pendiente\_2 = \frac{\$FF}{\$68 - \$40}$ ($06)

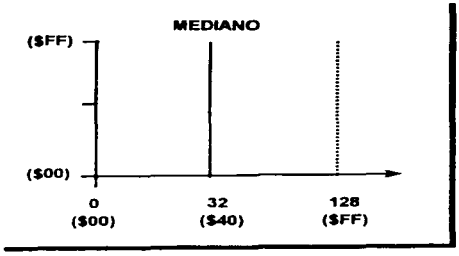


* Se asume que el punto 2 esta en la pendiente 2.
* Se asume que la pendiente 2 es negativa.

$$La\_pendiente\_2\_es\_calculada\_como: \frac{255}{Punto\_B_X - Punto\_2_X}$$

El FUDGE calcula las pendientes.

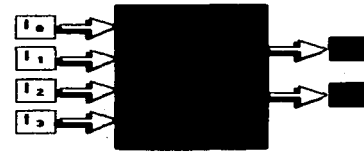
Representación de las funciones de membresía de salida.



Las funciones de membresía de salida están representadas por singletons con 1 byte por función de salida.

Una vez más el FUDGE convertirá las unidades de aplicación a valores hexadecimales requeridos por los Kernels.

El ANSI C Kernel, acepta hasta cuatro entradas con ocho funciones de membresía por entrada, permitiendo hasta dos salidas con hasta 8 singletons por salida. No hay limitación en los antecedentes (típicamente 2-5) o consecuentes (típicamente 2). Todos los antecedentes son conectados por el operador and.



2.3. Lenguaje C.

El lenguaje "C" es un lenguaje de alto nivel de propósito general, es decir, no está orientado a un área especifica, sus características lo han hecho un lenguaje de gran aceptación, sus principales características son:

- Programación estructurada.
- Economía en las expresiones.
- Abundancia en operadores y tipos de datos.
- Codificación en bajo y alto nivel simultáneamente.
- Reemplaza ventajosamente la programación en ensamblador.
- No está orientado a ninguna área en especial.
- Producción de código objeto altamente optimizado.
- Facilidad de aprendizaje.

El lenguaje C nació en los laboratorios Bell de AT&T y ha sido estrechamente asociado con el sistema operativo UNIX, ya que su desarrollo se realizó en este sistema y debido a que tanto UNIX como el propio compilador C y la casi totalidad de los programas y herramientas de UNIX, fueron escritos en C.
En 1972, Dennis Ritchie, modifico el lenguaje B, creando el lenguaje C y rescribiendo el UNIX en dicho lenguaje, La novedad que proporciono el lenguaje C sobre el B fue el diseño de tipos y estructuras de datos.

Los tipos básicos de datos eran char, int, float y double. Posteriormente se añadieron los tipos short, long, unsigned y enumeraciones. Los tipos estructurados básicos son las *estructuras*, *las uniones* y los *arrays*. Estos permiten la definición y declaración de tipos derivados de mayor complejidad.

Las instrucciones de control de flujo de C son las habituales de la programación estructurada: if, for, while,switch-case.

Una de las cosas más importantes de C que se deben de recordar es que es Case sensitive (sensible a las mayúsculas). Es decir que para C no es lo mismo escribir Printf que printf, también es importante indicar que las instrucciones se separan por ";".

Es posible combinar programas en "C" con lenguajes escritos en ensamblador o en otros lenguajes .

Aplicaciones del lenguaje "C".

- Programación de sistemas.
- Bases de datos.
- Sistemas operativos
- Graficación.
- Etc.

Bibliotecas.

La biblioteca estándar de C consiste de grupos de funciones precompiladas que deben encadenarse a cualquier programa que las use.

El programador puede construir sus propias bibliotecas si estas no están incluidas en las estándar.

La secuencia de pasos para la construcción de un programa ejecutable se muestra en la siguiente figura:

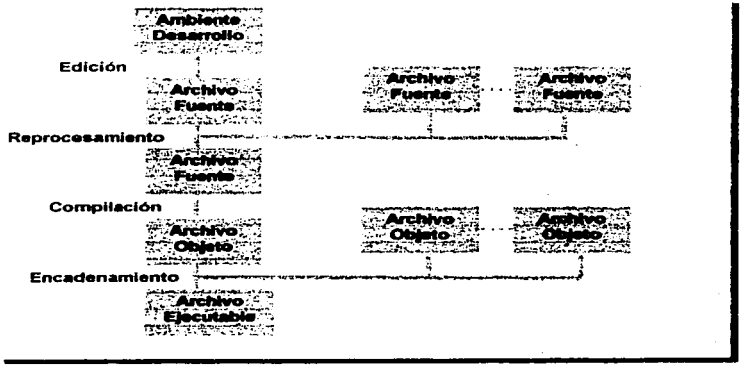


Fig. 2.11. Pasos para la elaboración de un archivo ejecutable

Estructura de un programa en C.

Un programa en C consiste en un grupo de funciones. Todo programa en C contiene una función denominada main que se ejecuta primero, y desde esta función se hace la invocación directa o indirecta a las otras funciones.

Ejemplo:

```
void mensaje 1() {
printf ("      Aplicación de la Lógica Difusa      \n");
}
void mensaje 2() {
printf ("      APLICACION DE LA LOGICA DIFUSA      \n");
}
void mensaje 3() {
printf ("      AL CONTROL DE PRESIÓN Y TEMPERATURA   \n");
}
void mensaje 3() {
printf ("      EN UN SISTEMA  DE AIRE ACONDICIONADO   \n");
}
void mensaje 4() {
printf ("      DE UN EDIFICIO INTELIGENTE   \n");
main (){
mensaje 1();
mensaje 2();
mensaje 3();
mensaje 4();
}
```

identificadores

Utilizamos a los identificadores para diferenciar a las variables, constantes simbólicas o funciones de un programa.
Se componen de letras y dígitos; el primer carácter debe ser una letra. El carácter de subrayado "_" cuenta como una letra.
Las letras mayúsculas y minúsculas son distintas. La práctica tradicional de C es usar letras minúsculas para nombres de variables, y todo en mayúsculas para constantes simbólicas.

Palabras reservadas

Las siguientes palabras están reservadas, es decir, el programador no las puede utilizar como identificadores en su programa.

Palabras reservadas:

auto	double	int	struc
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Elementos del lenguaje

Tamaños y tipos de datos.

Un tipo de datos define una agrupación de valores y puede representarse directamente en la computadora.

Esta representación está en función del "hardware" de la máquina y se refiere al espacio para guardar un valor que corresponde al tipo de dato.

tenemos los siguientes tipos básicos en C:

char Se representa por un solo byte, capaz de contener un carácter del código ASCII.

int Es un entero con signo almacenado en una palabra de la máquina. El tamaño depende de la computadora y del compilador que se usa.

float Punto flotante en presición normal.

double Punto flotante de doble presición.

Además se cuenta con los siguientes modificadores que se aplican a los tipos básicos:

short

long

Establecen diferentes tamaños de acuerdo a la computadora y el compilador de C utilizados. Así tenemos que para el compilador TurboC ver 2.0 *short* es de 16 bits y *long* es de 32 bits.

unsigned

Indica que el tipo de dato no tiene signo, es decir, su valor es positivo. Sigue las reglas de la aritmética 2^n, donde n es el número de bits que lo representa.

signed

Establece que el tipo de dato tiene signo, es decir, su valor puede ser positivo o negativo. El número es guardado en la notación complemento a dos.

La siguiente tabla muestra los tipos y tamaños de datos, esto es algo orientativo, ya que el valor depende del sistema.

TIPO	DATOS ALMACENADOS	N° DE BITS	VALORES POSIBLES (RANGO)	RANGO USADO UNSIGNED
char	caracteres	8	-128 a 128	0 a 255
int	enteros	16	-32.767 a 32.767	0 a 65.535
long	enteros largos	32	-2.147 .483.647 a 2.147 .483.647	0 a 4.294.967.295
float	nums. reales (coma flotante)	32	3,4E-38 a 3,4E38	
double	nums. reales (coma flotante doble)	64	1,7E-307 a 1,7E308	

Constantes y variables

Antes de escribir las instrucciones de un programa, es necesario declarar las variables que se van a utilizar en él, esto significa reservar celdas de memoria para almacenar los datos durante la ejecución de un programa dándoles un nombre y un tipo.

Durante el programa, se hará referencia al dato de una variable mediante un nombre que se le ha asignado a la variable , mediante el tipo de dato asignado a la variable y mediante el tipo de dato asignado a la variable.

Las constantes son aquellos datos que no pueden cambiar a loa largo de la ejecución del programa.

Funciones.

Las funciones dividen tareas grandes de computación en varias más pequeñas y permite la posibilidad de construir sobre lo que otros ya han hecho, en lugar de comenzar de cero.

Una función es un conjunto de estatutos diseñados para resolver alguna tarea. El resultado de esta tarea puede retornarse o no.

Reglas de acceso a variables.

Las reglas de acceso establecen cuándo y donde pueden usarse las variables.

El acceso se determina de dos formas: por la forma de almacenamiento y por sus reglas de alcance.

La forma de almacenamiento de una variable es un atributo que indica en que lugar de la computadora será almacenada y cuando puede usarse.

Las reglas de alcance indican su acceso dependiendo de donde fueron declaradas las variables.

Formas de almacenamiento.

auto: Automáticas

Estas variables son locales al bloque que las declara y solo existen dentro del ámbito del bloque, es decir, las variables se crean al inicio de la ejecución del bloque y se destruyen al final de la ejecución del mismo.

La palabra clave *auto* se toma en forma implícita por el compilador.

static: Estáticas

Estas variables retienen sus valores sus valores a lo largo de la ejecución del programa y se destruye al final de la ejecución delo mismo.

Las variables declaradas como *static* dentro de un bloque (o función) son internas en el sentido de que solo su identificador es conocido dentro del bloque, pero el valor de esta variable se mantiene sin importar cuantas veces se invoque o se abandone el bloque.

Una variable estática declarada fuera de toda función y al comienzo de un archivo, es accesible para cualquiera otro rachivo.

extern: Externas

Este almacenamiento es utilizado para las variables globales. Retienen sus valores a lo largo de toda la ejecución del programa y además son accesibles a cualquier función.

Toda variable declarada de una función se supone externa.

Las variables declarada fuera de una función se supone externa.

Las variables externas estáticas son accesibles dentro del archivo que las declara.

Cualquier otra variable externa es accesible a funciones de otros archivos siempre que en esos archivos se declare como externa.

register Registro

Estas variables tiene forma de almacenamiento automático, pero sus valores se guardan en los registros del procesador con el objeto de hacer más rápida la ejecución.

CAPITULO 3

PLANTEAMIENTO DEL PROBLEMA

La finalidad del sistema de Aire Acondicionado con control difuso es: proporcionar las condiciones ambientales necesarias de temperatura para un espacio de aproximadamente 450 m^2, en el cual los usuarios cuentan con áreas de trabajo divididas en cubículos, por lo que el suministro de aire se distribuirá de igual forma en todo el espacio.

La experiencia en el manejo de Aire Acondicionado señala que una presión de 0.5 in H_2O, es suficiente para proveer el aire requerido al área que se está manejando, por lo que dicha presión se mantendrá estática y los cambios de temperatura en el Aire Acondicionado se obtendrán operando la válvula de agua helada.

Si es accionado el motor del ventilador instalado dentro de la manejadora el consumo de energía será mucho mayor, que si se manipula el actuador de la válvula de agua helada, lo que se pretende es optimizar el ahorro energético del edificio y obviamente no tener oscilaciones considerables entre el valor de temperatura requerido y un valor real.

3. 1. DETERMINACIÓN DE ENTRADAS Y SALIDAS DEL SISTEMA

Como primer punto se identificarán cuáles son las variables de entrada y salida al sistema:

De acuerdo al planteamiento del problema consideramos:

1. **La primera entrada al sistema es la temperatura que esta censando el termostato ubicado dentro del espacio en consideración.**

2. **La segunda entrada al sistema es la temperatura de referencia y es suministrada por el usuario. Esta será la temperatura que debe alcanzar el sistema de aire acondicionado mediante su regulación.**
3. **La salida será el accionamiento del actuador de la válvula de agua helada.**

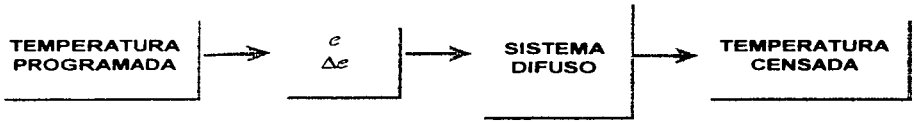


Fig. 3.1. Entradas y salidas del sistema

Para evaluar la respuesta del sistema es necesario determinar el error e existente entre ambas señales, error que está dado por

$$e = T_{ref} - T_m$$

donde,

T_{ref} es la temperatura programada

T_m es la temperatura medida por el termostato

Por otra parte, también es necesario conocer la tendencia del error, es decir, determinar si las compensaciones que efectúa el control están alejando o acercando la respuesta del sistema a la señal de referencia. Tal tendencia está

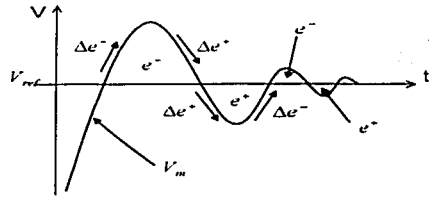
dada por la variación entre la toma sucesiva de dos muestras del termostato y se define como Δe:

$$\Delta e = T_{m-1} - T_m$$

donde,

T_m es el temperatura real medida en el termostato y,

T_{m-1} es el temperatura medida anterior.



Como puede verse, el error e y la variación del error Δe pueden tomar tanto valores negativos como positivos. Si e es negativo, significa que el voltaje del sensor está por arriba del voltaje de referencia; si es positivo, está por debajo. En el caso del Δe, si éste es negativo, indica que el valor del voltaje actual leído en el sensor es mayor al de la lectura anterior; en caso contrario, si Δe es positivo, el valor del voltaje actual ha disminuido respecto a la lectura anterior. Para que el sistema alcance la temperatura requerida, el proceso de control debe tener en cuenta ambas variables; es decir, debe conseguir llevarlas a su valor mínimo. La posición del mecanismo es óptimo cuando ambos, el error e y la variación del error Δe, son iguales a cero.

Ya que para todas las señales globales de entrada al sistema generadas por los sensores es posible implementar el análisis de control anterior, es conveniente tomar como variables de entrada al algoritmo de control el error e y la variación del error Δe. Por tanto, éstas serán las entradas rígidas al algoritmo difuso de control.

Es importante observar que en el presente trabajo solo se realizara la simulación mediante el código de programación en C y el fudge, si se realizara físicamente las pruebas utilizando termostatos, válvulas, motores, en fin todo el equipo para aire acondicionado se tendría que utilizar un microcontrolador y un código ensamblador.

Calculo del e y Δe.

$$e = 16 - 10 = 6°C$$

donde,

T_{ref} = 16°C

T_m = 10°C

$$\Delta e = 18 - 10 = 8°C$$

donde,

T_m = 10°C.

T_{m-1} vamos a suponer con un valor de 18°C.

Los valores de:

e = 6°C. y Δe 8°C, son los valores que se utilizaran para el cálculo del grado de membresía.

3. 2. FUSIFICACIÓN

De acuerdo con el desarrollo del control con lógica difusa, en esta etapa se construyen los modelos difusos para las variables de entrada.

Primero, se asignaran las etiquetas difusas a cada variable de entrada con su correspondiente dominio, indicando así los valores de entrada que le corresponden a cada función de membresía o etiqueta difusa.

Para el error (e) se tiene un rango de (-20, 13) del cual se tomarán los valores correspondientes para los tres conjuntos difusos:

NEGATIVO	CERO	POSITIVO
-20,-3	-7,7	3,13

El rango de Δe será de (-16,16) y los conjuntos se distribuirán de la siguiente manera:

NEGATIVO	CERO	POSITIVO
-16,-4	-10,10	4,16

Tomando en cuenta la distribución gráfica y seleccionando un valor en cada una de las entradas se calculará el grado de membresía para ambas.

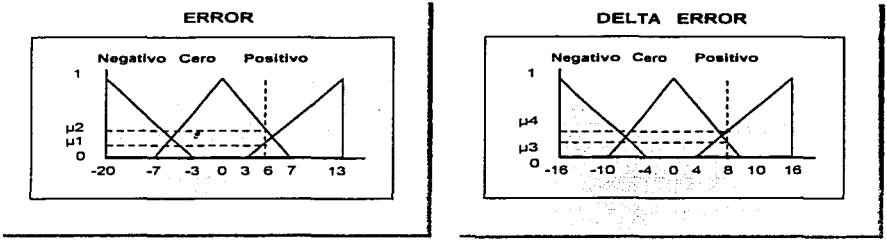


Fig. 3.2.Representación gráfica de las vanables de entrada del sistema en la etapa de fusificación.

Los puntos utilizados para los cálculos del e y Δe son vislumbrados gracias a las graficas localizadas en al figura 3.2.

Ahora calculando el grado de membresía de e para un valor de 6,

Para la pendiente m_1 tenemos los siguientes puntos (0,1) y (7,0) sustituyendo en la ecuación de la recta se tiene:

$$m_1 = \frac{(y - y_0)}{(x - x_0)} = \frac{(0-1)}{(7-0)} = -1/7$$

Para m_2 se tiene el siguiente par de puntos: (3,0) y (13,1).

Por lo tanto:

$$m_2 = \frac{(1-0)}{(13-3)} = \frac{1}{10} = 0.1$$

Para Δe se considera el valor igual a 8.

Para calcular las pendiente m_3, se toman los siguientes puntos: (0,1) y (10,0).

$$m_3 = \frac{(0-1)}{(10-0)} = -\frac{1}{10} = -0.1$$

Para m_4 se tiene el siguiente par de puntos: (4-0) y (16-1).

$$m_4 = \frac{(1-0)}{(16-4)} = \frac{1}{12}$$

Ahora utilizando las pendientes calculadas podemos obtener los puntos $\mu(\ell_1)$, $\mu(\ell_2)$, $\mu(\Delta\ell_1)$ y $\mu(\Delta\ell_2)$.

$$\mu(\ell_1) = m_1(x - x_0) + y_0 = -\frac{1}{7}(6-0) + 1 = 0.14$$

$$\mu(\ell_2) = y - m_2(x - x_0) = 0 - 0.1(3-6) = 0.3$$

$$\mu(\Delta\ell_1) = m_3(x - x_0) + y_0 = -0.1(8-0) + 1 = 0.2$$

$$\mu(\Delta\ell_2) = y - m_4(x - x_0) = 0 - \frac{1}{12}(4-8) = 0.33$$

Gráficamente el grado de membresía es el siguiente:

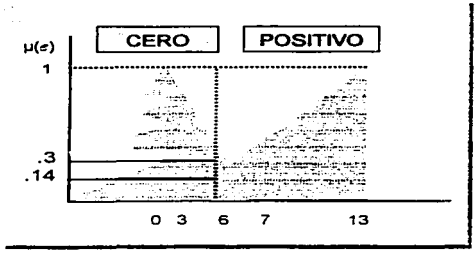


Fig. 3.3 Representación gráfica del grado de membresía para el e

3. 3. INFERENCIA

Ya calculado el grado de membresía para el e y Δe se procede a elaborar la tabla de verdad, la construcción de la tabla se establece bajo las condiciones de trabajo del sistema.

Etiquetas de variables de entrada

e / Δe	NEGATIVO	CERO	POSITIVO
NEGATIVO	ABRIR VÁLVULA MUCHO 1	VALVULA NO OPERA 2	CERRAR VALVULA POCO 3
CERO	ABRIR VALVULA POCO 4	VALVULA NO OPERA 5	CERRAR VALVULA POCO 6
POSITIVO	ABRIR VALVULA POCO 7	VALVULA NO OPERA 8	CERRAR VALVULA MUCHO 9

de regla

Etiquetas de variables de salida

Fig 3.4. Tabla de verdad utilizada para generar las reglas de inferencia

La intersección de estos conjuntos nos determina que reglas van a operarse en el sistema de control, evaluación que se realizara en el siguiente paso.

Las reglas obtenidas son:

R_1. IF e negativo AND Δe negativo THEN abrir válvula mucho.

R_2. IF e cero AND Δe negativo THEN válvula no opera.

R_3. IF e positivo AND Δe negativo THEN cerrar válvula poco.

R_4. IF e negativo AND Δe cero THEN abrir válvula poco.

R_5. IF e cero AND Δe cero THEN válvula no opera.

R_6. IF e positivo AND Δe cero THEN cerrar válvula poco.

R_7. IF e negativo AND Δe positivo THEN abrir válvula poco.

R_8. IF e cero AND Δe positivo THEN válvula no opera.

R_9. IF e positivo AND Δe positivo THEN cerrar válvula mucho.

Las reglas sombreadas es la intersección de los conjuntos, por lo que para:

$e = 6$ y $\Delta e = 8$

Las reglas activadas serán:

R_5. IF e cero (0.14) AND Δe cero (0.2) THEN válvula no opera.

R_6. IF e positivo (0.3) AND Δe cero (0.2) THEN cerrar válvula poco.

R_8. IF e cero (0.14) AND Δe positivo (0.33) THEN válvula no opera.

R_9. IF e positivo (0.3) AND Δe positivo (0.33) THEN cerrar válvula mucho.

Simplificando:

R_5. (0.14) (0.2) = 0.14
R_6. (0.3) 0.2) = 0.2
R_8. (0.14) (0.33) = 0.14
R_9. (0.3) (0.33) = 0.3

Valor de la regla

Para la salida tomamos el valor máximo de las reglas activadas es decir la válvula va a cerrar mucho (0.3) a cerrar válvula poco (0.2). Gráficamente se observa:

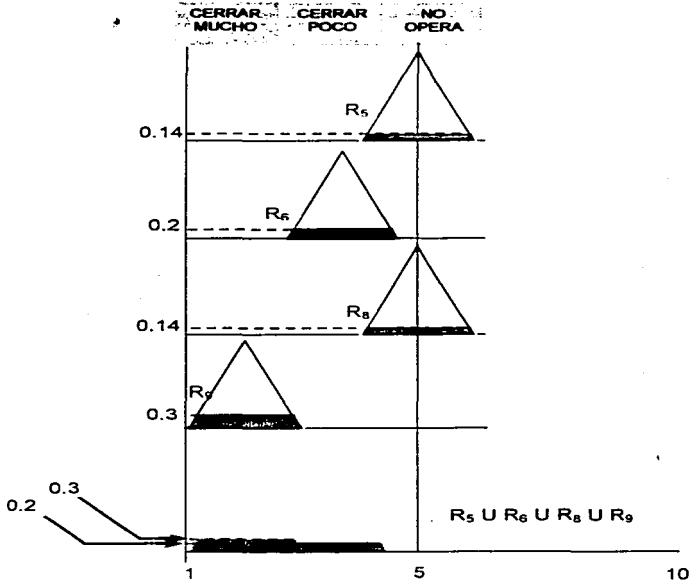


Fig.3.5. Representación gráfica de la etapa de inferencia

3. 4. DEFUSIFICACION.

En esta etapa se encontrarán los valores reales en los cuales va oscilar el actuador de la válvula de agua helada.
Como referencia hablamos de un actuador que trabaja con ancho de pulso.
Ahora utilizando diferentes valores dentro del rango de conjuntos activados y empleando la formula:

$$COG = \frac{\sum_{i=1}^{n} \mu(x) \cdot x}{\sum_{i=1}^{n} \mu(x)}$$

Se calculará por el método del centroide el valor real de oscilación de la válvula de agua helada.

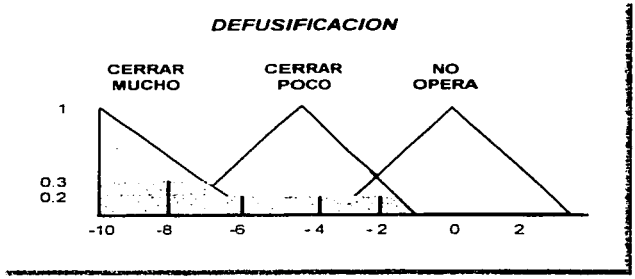


Fig.3.5. Representación gráfica de la etapa de defusificación.

$$COG = \frac{(-10)(0.3)+(-8)(0.3)+(-6)(0.2)+(-4)(0.2)+(-2)(0.2)}{(0.3)+(0.3)+(0.2)+(0.2)+(0.2)} = -6.5mS$$

El valor calculado indica el valor del tiempo en mS ya que el actuador usado para programar el movimiento de giro a la válvula controladora del paso del agua helada es un servo-motor regulado con una señal modulada en ancho de pulso (PWM-Pulse Width Modulation).

El diseño básico de un servo-motor modulado por ancho de pulso es un motor eléctrico acoplado aun circuito electrónico que hace las veces de un regulador y convertidor de señal de entrada a una señal eléctrica de corriente directa.

El circuito electrónico genera a su salida una señal regulada en amplitud y polaridad que controla la posición, sentido de giro y velocidad del motor de c.d.

Por su parte, la señal de entrada requerida en el circuito será una señal digital a frecuencia constante y variando solo el ancho de pulso, en este caso una variación de 1 a 10mS en la amplitud del pulso es suficiente para abarcar 360° de giro en el motor de c.d. De tal modo que un ancho de pulso de 1 mS girará la flecha del motor (y por lo tanto, la válvula de paso) en el sentido de las manecillas del reloj a una posición de 0°, consecuentemente, una señal con un ancho de pulso de 10 mS girará la flecha del motor hasta 360° en el sentido contrario a las manecillas del reloj. señales con un ancho de pulso entre este rango generarán el movimiento de giro proporcional a su valor.

Así, una señal PWM de 5mS colocará al servo a la mitad de la carrera, es decir a 180°.

Estas características de movimiento son útiles para mover dispositivos que giran gradualmente hasta 360°, como es el caso de las válvulas y llaves de paso.

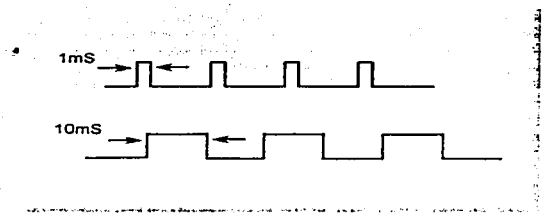
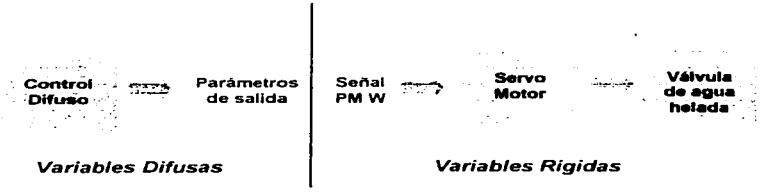


Fig. 3.6. Señales de ancho de pulso para el movimiento del servo-motor.

La salida del sistema de control difuso es una señal modulada de ancho de pulso adecuada para conseguir la apertura o cierre de la válvula de paso de acuerdo a los consecuentes definidos en los conjuntos de salida difusos (abre mucho, cierra poco, cierra mucho, etc).



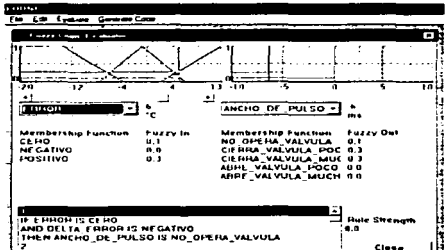
CAPITULO 4

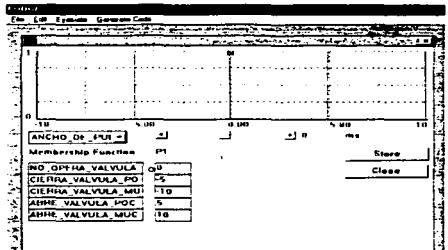
PRUEBAS Y RESULTADOS.

En el capitulo anterior fueron seleccionados diferentes valores dentro del rango de conjuntos de salida activados y empleando la formula para calcular el COG, se obtuvo la siguiente ecuación:

$$COG = \frac{(-10)(0.3)+(-8)(0.3)+(-6)(0.2)+(-4)(0.2)+(-2)(0.2)}{(0.3)+(0.3)+(0.2)+(0.2)+(0.2)} = -6.5mS$$

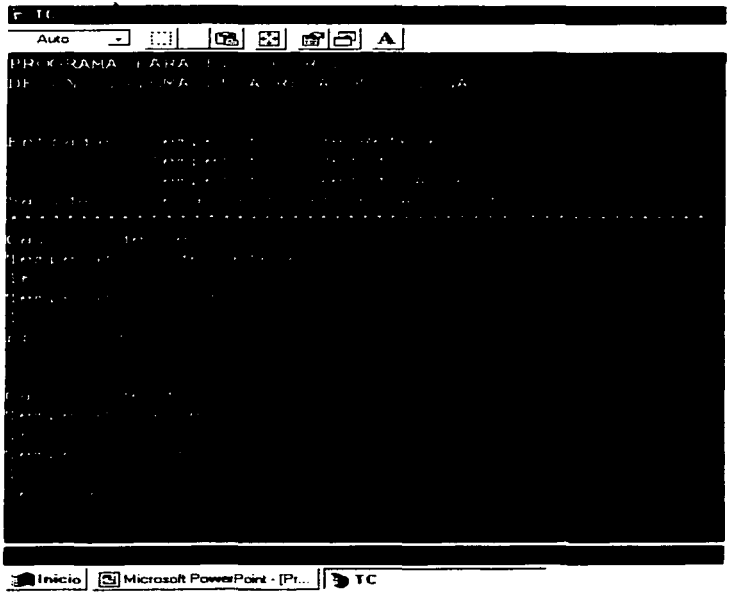
Se observa que el valor final es de –6.5 mS. Ahora se realizara una corrida en el simulador FUDGE con los datos de entrada: $c = 6$ y $\Delta c = 8$, si los datos introducidos en FUDGE son correctos se obtendrá un valor aproximado al obtenido en la ecuación anterior.

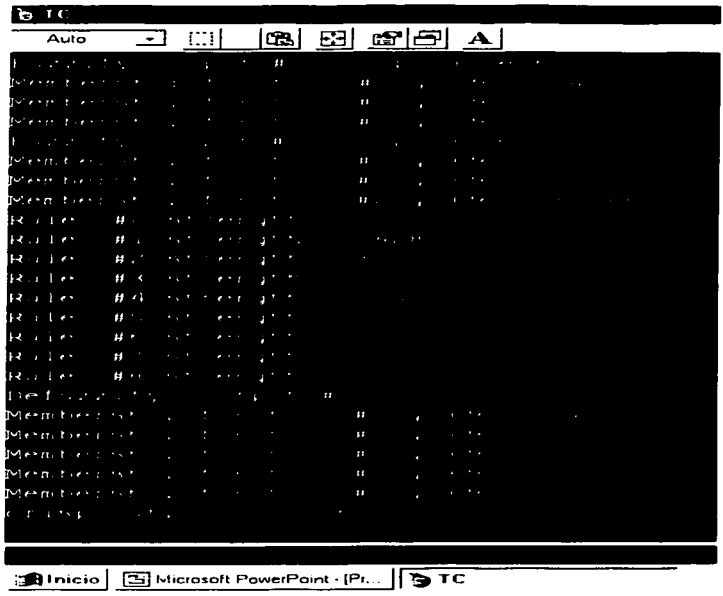




Se observa que las reglas han sido activadas y el porcentaje de apertura de la válvula indica: -6 mS.

Utilizando los valores iniciales de T_m, T_{ref} y T_{m-1} del presente trabajo, se correrá el código obtenido en programación en C.





Como puede observarse el código de C primero utiliza los valores: "temperatura de referencia, temperatura medida y temperatura medida anterior" para calcular el error y el delta error y después calcula el valor de apertura de la válvula, el valor obtenido en C, es aún mas cercano al valor obtenido en la ecuación de COG que el obtenido en FUDGE, la diferencia radica en el tipo de conjuntos de salida utilizados en el COG Y FUDGE, ya que en el primer caso, los cálculos es realizaron con conjuntos

triangulares y trapezoidales y en el segundo caso fueron utilizados singletons, y como se menciono en el capitulo de Lógica Difusa, los conjuntos triangulares nos proporcionan mayor exactitud que los singletons para el cálculo de respuesta en el sistema.

Sin embargo la aproximación es muy buena, por lo que se recomienda implementar la simulación realizada en un sistema real.

El valor obtenido nos indica que bajo las condiciones iniciales de error y delta error, el motor va a girar en sentido contrario a las manecillas del reloj, oscilando la posición de la válvula entre "cerrar mucho y cerrar poco" hasta que se alcancen las condiciones requeridas y así, nuevamente cuando las condiciones en la zona controlada por el aire acondicionado sean diferentes y se requiera otro tipo de descarga, el error y el delta error oscilaran hasta alcanzar el valor requerido y proporcionar la temperatura ideal según las nuevas condiciones, para comodidad del usuario.

Es importante hacer hincapié que el motor que se esta utilizando se mueve en un rango de 1 a 10 mS, según el valor de la amplitud va a ser el movimiento de giro del motor, sin embargo a la salida se utilizan valores negativos (-10 a 10), por que si sólo fueran valores positivos el motor solo tendría un sentido de giro, entonces si contamos con valores positivos y negativos se moverá en ambos sentidos cerrando y abriendo la válvula de agua helada de esta manera es posible alcanzar la temperatura deseada.

BIBLIOGRAFIA

1. Heriberto Ordóñez Mondragón, CINVESTAV-IPN. Departamento de Ingeniería, Sección de Bioelectrónica. *Aplicación de la lógica difusa a una planta de fermentación*
2. Ruben Santiago Godoy. CINVESTAV-IPN. Departamento de Ingeniería, Sección de Bioelectrónica. *Apuntes de lógica difusa.*
3. Rojas S. Antonio S; Carrillo M. Rodolfo; Herrera C. Omar David; *Memoria del cálculo del Módulo de Adquisición de Datos del sistema de Diagnóstico por Vibraciones Mecánicas Portátil.* Instituto Nacional de Investigaciones Nucleares. Informe técnico P.AU-9704. Noviembre 1997.
4. Rojas S. Antonio S; Carrillo M. Rodolfo; Herrera C. Omar David; *Procedimiento de Desarrollo de Software para Pruebas de Funcionamiento del Módulo de Adquisición de datos del Sistema de Diagnóstico.* Instituto Nacional de Investigaciones Nucleares. Informe técnico P.AU-9704. Noviembre 1997.
5. Motorola Corporation Inc. *Fast and LS TTL Data*, 1994.
6. Intel Corporation, Component Data Catalog, 1993
7. Motorola Corporation Inc., *Fuzzy Logic Educatión Program 2.0.* Center for Emerging Computer Technologies, Motorola, Inc; USA 1994.
8. Ogata Katsushito, *Teoria Del Control Moderno*, Ed. Prentice Hall, 1992.
9. METASYS, *Operator Workstation Tomo I.* Manual del usuario (Fan 634) Jhonson Controls, 1999.

10. METASYS, *Operator Workstation Tomo II*. Manual del usuario (Fan 634) Jhonson Controls, 1999.
11. David Elias, *Lenguaje de Programación C*, CINVESTAV-IPN
12. www.elrincondelc.com Curso De C.

APENDICE A
Código de programación en C

```
/*
        Application name:       FUzzy Development and Generation Environment (FUDGE)
Version V1.02
        File name:              Fuzzy.c
        Written by:             Alex DeCastro & Jason Spielman
        Copyright Motorola 1994
        Aplication by:          Carolina Gutiérrez Bárcenas
        SCALE           1
*/

#include <stdio.h>
#include "Fuzzy.h"

int     num_inputs = 2;
int     num_outputs = 1;
int     num_rules = 9;

int     num_input_mfs[2] = { 3, 3 };

struct  In      Inputs[] =
{
        { -20.000000, 13.000000 },
        { -16.000000, 16.000000 }
};

float   inmem_points[2][7][4] =
{
        {
                { -7.000000, 0.000000, 0.000000, 7.000000 },
                { -20.000000, -20.000000, -13.000000, -3.000000 },
                { 3.000000, 13.000000, 13.000000, 13.000000 }
        },
        {
                { -10.000000, 0.000000, 0.000000, 10.000000 },
                { -16.000000, -16.000000, -16.000000, -4.000000 },
                { 4.000000, 16.000000, 16.000000, 16.000000 }
        }
};
int     num_output_mfs[1] = { 5 };

struct  Out     Outputs[] =
{
```

```
};

float   outmem_points[1][7][4] =
{
        {
                { 0.000000 },
                { -5.000000 },
                { -10.000000 },
                { 5.000000 },
                { 10.000000 }
        }
};
float   crisp_outputs[1] = { 0};

int     num_rule_ants[9] = { 2, 2, 2, 2, 2, 2, 2, 2, 2};
int     num_rule_cons[9] = { 1, 1, 1, 1, 1, 1, 1, 1, 1};

struct  Rule    Rules[9] =
{
        { { 0x00, 0x09 }, { 0x80 } },
        { { 0x00, 0x01 }, { 0x80 } },
        { { 0x10, 0x11 }, { 0x90 } },
        { { 0x08, 0x09 }, { 0xa0 } },
        { { 0x08, 0x01 }, { 0x98 } },
        { { 0x08, 0x11 }, { 0x98 } },
        { { 0x10, 0x09 }, { 0x88 } },
        { { 0x10, 0x01 }, { 0x88 } },
        { { 0x10, 0x09 }, { 0x88 } }
};

void main ()
{
float temp_ref,temp_med,temp_med_1;
float delta_error;
float error;

clrscr();
printf("PROGRAMA PARA EL CONTROL DIFUSO\n");
printf("DE UN SISTEMA DE AIRE ACONDICIONADO\n\n\n");
printf("Entradas: Temperatura de Referencia (øC)\n");
printf("          Temperatura Medida (øC)\n");
printf("          Temperatura Medida Anterior (øC)\n");
printf("Salida:   Señal Modulada en Ancho de Pulso (PWM)\n");
printf("*****************************************************\n");
printf("Calculo del error\n");
printf("Temperatura de referencia? \n");
scanf ("%f", &temp_ref);
```

```
printf ("Temperatura Medida? \n");
scanf ("%f", &temp_med);
error =temp_ref -temp_med;
printf ("Error = %f\n\n\n", error);
//getche ();

printf("Calculo de delta_error\n");

printf("Temperatura medida anterior? \n");
scanf ("%f", &temp_med_1);

printf ("Temperatura medida? \n");
scanf ("%f", &temp_med);
delta_error = temp_med_1 -temp_med;

printf ("Delta error = %f",delta_error);
getche ();

fuzzy_step(&error, &delta_error);

getche();
}

void fuzzy_step(float *crisp_inputs, float *crisp_outputs)
{
        int     in_index,rule_index,out_index;
        float   in_val;
        for (in_index = 0;in_index < num_inputs;in_index++)
        {
                fuzzify_input(in_index,crisp_inputs[in_index]);
        }
        for (rule_index = 0;rule_index < num_rules;rule_index++)
        {
                eval_rule(rule_index);
        }
        for (out_index = 0;out_index < num_outputs;out_index++)
        {
                crisp_outputs[out_index] = defuzzify_output(out_index, crisp_inputs);
//              /*if (TRACE)*\
                printf("crisp_output[%d] = %f\n", out_index, crisp_outputs[out_index]);
        }
}
void fuzzify_input(int in_index,float in_val)
{
        int i;
//      if (TRACE)
```

```
 printf("Fuzzify: input #%d crisp value %f\n", in_index, in_val);
        for (i = 0;i < num_input_mfs[in_index];i++)
        {
                fuzzy_inputs[in_index][i] = get_membership_value(in_index,i,in_val);
//              if (TRACE)
 printf("Membership function #%d grade %f\n", i, fuzzy_inputs[in_index][i]);
        }
}
float get_membership_value(int in_index,int mf_index,float in_val)
{
        if (in_val < inmem_points[in_index][mf_index][0]) return 0;
        if (in_val > inmem_points[in_index][mf_index][3]) return 0;
        if (in_val <= inmem_points[in_index][mf_index][1])
        {
                if (inmem_points[in_index][mf_index][0] ==
inmem_points[in_index][mf_index][1])
                        return 1;
                else
                        return ((in_val - inmem_points[in_index][mf_index][0]) /
                                        (inmem_points[in_index][mf_index][1] -
inmem_points[in_index][mf_index][0]));
        }
        if (in_val >= inmem_points[in_index][mf_index][2])
        {
                if (inmem_points[in_index][mf_index][2] ==
inmem_points[in_index][mf_index][3])
                        return 1;
                else
                        return ((inmem_points[in_index][mf_index][3] - in_val) /
                                        (inmem_points[in_index][mf_index][3] -
inmem_points[in_index][mf_index][2]));
        }
        return 1;
}
void eval_rule(int rule_index)
{
        int             in_index,out_index,mf_index,ant_index,con_index;
        int     val;
        float   rule_strength = 1;
        for     (ant_index = 0;ant_index < num_rule_ants[rule_index];ant_index++)
        {
                val = Rules[rule_index].antecedent[ant_index];
                in_index = (val & 0x07);
                mf_index = ((val & 0x38) >> 3);
                rule_strength = MIN(rule_strength,fuzzy_inputs[in_index][mf_index]);
        }
        rule_strengths[rule_index] = rule_strength;
```

```
//      if (TRACE)
 printf("Rule  #%d strength %f\n", rule_index, rule_strength);
        for (con_index = 0;con_index < num_rule_cons[rule_index];con_index++)
        {
                val = Rules[rule_index].consequent[con_index];
                out_index = (val & 0x03);
                mf_index = ((val & 0x38) >> 3);
                fuzzy_outputs[out_index][mf_index] =
MAX(fuzzy_outputs[out_index][mf_index],
                        rule_strengths[rule_index]);
        }
}
float defuzzify_output(int out_index,float *inputs)
{
        float       summ = 0;
        float       product = 0;
        float       temp1,temp2;
        int         mf_index,in_index;
//      if (TRACE)
 printf("Defuzzify: output #%d\n", out_index);
        for (mf_index = 0;mf_index < num_output_mfs[out_index];mf_index++)
        {
                temp1 = fuzzy_outputs[out_index][mf_index];
                temp2 = outmem_points[out_index][mf_index][0];
                summ = summ + temp1;
                product = product + (temp1 * temp2);
//              if (TRACE)
printf("Membership function #%d grade %f\n", mf_index,
fuzzy_outputs[out_index][mf_index]);
                fuzzy_outputs[out_index][mf_index] = 0;
        }
        if (summ > 0)
        {
                crisp_outputs[out_index] = product / summ;
                return crisp_outputs[out_index];
        }
        else
        {
//              if (NO_RULES){
                printf("No rules fire for:\n");
                        for (in_index = 0;in_index < num_inputs;in_index++)
                          printf("Input #%d=%f ", in_index,inputs[in_index]);
                                        printf("\n");
//                                      }
                return crisp_outputs[out_index];
        }
}
```