UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS
INTELIGENCIA ARTIFICIAL

CONTROL DIFUSO DE VIBRACIONES EN EDIFICIOS

TESIS

QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
HUGO ISRAEL ALCARAZ HERRERA

DIRECTOR DE LA TESIS:
DR. LUIS AGUSTÍN ÁLVAREZ ICAZA LONGORIA
INSTITUTO DE INGENIERÍA

CIUDAD UNIVERSITARIA, CD. MX. **OCTUBRE 2016**

Agradecimientos

Al *CONACyT*, ya que sin su invaluable apoyo, hubiese sido muy complicado poder terminar este ciclo, ¡muchas gracias!

A la *Universidad Nacional Autónoma de México* (UNAM) por cobijarme una vez más como un hijo, ¡siempre estaré agradecido contigo, Alma Materl!

Al *Dr. Luis Agustín Álvarez Icaza Longoria*, ya que gracias a su constante apoyo, paciencia y conocimiento, esta tesis pudo llevarse a cabo, ¡muchas gracias, Doc!

A la *Dra. Katya Rodríguez Vázquez* porque gracias a su clase de Cómputo Evolutivo y a su apoyo, esta tesis tiene mucho más que aportar, ¡gracias, Doctora!

A mis sinodales *Dr. Fernando Arámbula Cosío*, *Dr. Sergio Marcellin Jacques* y al *Dr. Héctor Benítez Pérez* por el tiempo dedicado en la corrección y comentarios de esta tesis, ¡gracias por su apoyo!

Dedicatoria

A mi querida familia, a mis amigos y a todas esas personas que estuvieron, están y estarán en mi camino.

¡Gracias a todos y a cada uno de ustedes!

Índice

Capítulo 1. Introducción

Capítulo 2. Modelado

Capítulo 3. Sistema difuso

Capítulo 4. Simulaciones numéricas

Capítulo 5. Conclusiones

Referencias y Apéndices

Capítulo 1
Introducción.

Estado del arte, antecedentes y motivación

Uno de los fenómenos naturales más devastadores para el hombre es el movimiento brusco y pasajero de la corteza terrestre, mejor conocido como temblor o sismo. Este fenómeno afecta principalmente a las edificaciones debido a que el movimiento provoca deformaciones en las estructuras y dependiendo de la intensidad del movimiento, los materiales con los que están construidos pueden salirse de la denominada "región elástica del material", es decir, el movimiento puede ser tan brusco que el material ya no puede regresar a su forma original provocando una deformación permanente y a su vez esto provoca el colapso de la edificación.

Para tratar de resolver este problema, hasta ahora existen dos posibles soluciones. La primer solución consiste en construir estructuras más robustas, es decir usar materiales más resistentes y en mayor cantidad para que el movimiento no afecte de forma considerable a la edificación. Esta solución es muy buena debido a que muy pocas veces falla, sin embargo el costo es muy alto. La segunda solución consiste en implementar un sistema de control que compense las fuerzas inducidas por el sismo, esto con la finalidad de evitar el colapso de la edificación. La confiabilidad de un sistema de control es alta y el costo resulta ser mucho menor comparado con la primer solución.

Considerando lo expuesto en el párrafo anterior, una solución fiable consiste en instalar un amortiguador magnetoreológico en la base de un edificio para atenuar las vibraciones producidas durante un sismo(Alvarez y García 2011). La solución antes descrita está desarrollada bajo el paradigma del control clásico, es decir, se ha modelado matemáticamente cada elemento del sistema (el edificio y el amortiguador), se ha diseñado un observador que obtiene los desplazamientos y velocidades de cada piso del edificio y se ha diseñado un controlador que actúa sobre el amortiguador y recibe como entradas las velocidades y desplazamientos de los edificios. Para corroborar la efectividad de este sistema de control, se presentan simulaciones numéricas

usando el registro sísmico de Santiago, en Chile en 1985. Otra solución, un poco similar a la descrita anteriormente, consiste en usar también un amortiguador magnetoreológico para amortiguar las vibraciones en los edificios a través de un controlador diseñado con la técnica de control semi activo (Dyke, Spencer, Sain y Carlson 1996). Dicho controlador tiene como entrada las aceleraciones de la tierra y no es necesario acoplar el modelo del amortiguador para su funcionamiento. Incluso, hay propuestas que utilizan el mismo actuador, sin embargo, el diseño del controlador está inspirado en el controlador "quasi-bang-bang" (Mousaad 2013). Este controlador arroja salidas parecidas a un controlador difuso, sin embargo, estas salidas únicamente toman los valores predefinidos por las reglas del controlador y sus entradas son los desplazamientos y aceleraciones de cada piso del edificio.

Existen otros trabajos cuyo propósito es el mismo, sin embargo, la forma de atenuar las vibraciones cambia, por ejemplo, se puede usar una base aislada (Chang y Spencer Jr 2010). Esta forma de atenuación consiste en tener un elemento que provoca el desacoplamiento entre la cimentación y el edificio. Este elemento está conectado a su vez a un conjunto de válvulas hidráulicas que harán las veces de actuadores gobernados por la técnica de control activo. La retroalimentación de este sistema de control es la aceleración de los pisos del edificio.

Por otro lado, es posible tratar este problema con un enfoque completamente distinto: Inteligencia Artificial, cuyas técnicas pueden resultar muy efectivas para tratar de resolver este tipo de problemas. De entre las diversas técnicas que constituye la Inteligencia Artificial, la Lógica Difusa es muy usada para el control de procesos industriales, en sistemas de decisión o identificación de imágenes aéreas (Duarte 2000). Se diseñó un sistema difuso capaz de clasificar fotografías aéreas considerando los niveles de luminancia por pixel. De esta forma, cierta zona de la fotografía corresponde a una determinada categoría que puede ser:

- Zona de río.
- Zona de construcción Humana.
- Zona dedicada a la agricultura.
- Zona boscosa.

Hay trabajo en el ámbito de las energías renovables, por ejemplo en Santos y Miranda (2012) se diseña un sistema difuso para un aerogenerador cuyo objetivo es aprovechar de la mejor forma el viento para generar la mayor cantidad de energía eléctrica. El diseño de este sistema difuso tiene un nivel de complejidad alto debido a que este sistema eólico es altamente no lineal y

le afectan diversos componentes físicos como la dirección y sentido del viento así como la generación de perturbaciones que afectan a la energía captada del viento cada vez que una pala pasa cerca de la torre.

Los sistemas difusos también se usan en electrodomésticos y sistemas de aire acondicionado, por mencionar algunos ejemplos.

Un sistema de control basado en lógica difusa, puede llamarse también "sistema difuso". Existen muchos sistemas difusos que han sido diseñados para sustituir algunas labores humanas debido a que resultan ser peligrosas o muy repetitivas. El diseño de un sistema difuso basado en la experiencia de un operador de trenes para controlar el frenado de los mismos (Yasunobu, Miyamoto y Ihara 2002) es un ejemplo concreto de cómo un sistema difuso puede implementarse en una situación real. El funcionamiento de este sistema difuso consiste en reducir la aceleración del tren en función de la velocidad que lleva y de la distancia que le falta por recorrer para llegar a la siguiente estación.

Así como hay sistemas difusos para un propósito específico (como el mencionado anteriormente) hay sistemas difusos cuyo propósito es más demostrativo o con la finalidad de simular el comportamiento de los elementos a controlar. Un sistema difuso para controlar la velocidad de un motor de corriente directa (Coronel y Hernández 2004) desarrollado en Matlab es un ejemplo de cómo es posible simular el control de un dispositivo a través de un software. Este sistema difuso usa las variaciones en el par de carga para poder controlar las revoluciones del motor de corriente directa.

Ya sea con otras técnicas de Inteligencia Artificial o con otras ramas de la Computación, los sistemas difusos pueden interactuar de tal forma que pueden formar parte de un sistema más complejo para llevar a cabo una tarea concreta. Tal el caso de un sistema difuso que, con Visión Artificial, puede controlar el comportamiento de un robot móvil (Alcaraz 2012) para buscar un patrón contenido en un objeto y recolectarlo. Este sistema fue desarrollado en LabVIEW y tiene como entrada, la distancia del robot móvil con respecto al objeto que va a recolectar. Esta distancia se calcula a través de un mapeo entre pixeles y centímetros. Este mapeo se consigue mediante la adquisición de imágenes por medio de una cámara web en donde se muestra la posición del robot móvil y del objeto a recolectar. Como salida, tiene el voltaje que se aplica a dos motores que constituyen su sistema de locomoción y un motor más que rige el sistema de sujeción. De esta forma, conforme el robot móvil se acerca al objeto a recolectar, el voltaje suministrado al sistema de locomoción disminuye y cuando el robot móvil está

a una distancia dentro del rango de operación del sistema de sujeción, el objeto es recolectado.

Como hasta ahora se ha visto, los sistemas difusos pueden implementarse en sistemas susceptibles a la aplicación de la teoría de Control, brindando rapidez y simpleza en la construcción y diseño del sistema, flexibilidad en el manejo de datos de entrada imprecisos o ambiguos así como también en la generación de una salida (o salidas, según sea el caso) precisa a través de la utilización de una base de conocimientos que ayuda a tomar decisiones para el procesado de los datos y obtener así el comportamiento deseado.

Teniendo lo anterior presentes, con este trabajo, se abren nuevos caminos de exploración en la utilización de esta técnica de Inteligencia Artificial en áreas donde hay escaso o nulo trabajo en la aplicación de este tipo de sistemas.

Objetivo de la tesis

El objetivo de la presente tesis es desarrollar un Sistema de Control usando los conceptos de Lógica Difusa para atenuar las vibraciones que se pueden presentar en edificios a consecuencia de sismos.

Aportaciones

Si se usan técnicas clásicas de control, el desarrollo resulta ser largo y complejo debido al modelado matemático que implica. La presente tesis desarrolla una alternativa de control confiable basada en Lógica Difusa para atenuar vibraciones debido a que existe poco trabajo en la resolución de este tipo de problemas con técnicas de Inteligencia Artificial.

El software desarrollado está construido para poder usarse con cualquier actuador y cualquier planta modificando el módulo correspondiente, ofreciendo una gran flexibilidad para cualquiera que quiera crear y probar su propio Sistema de Control basado en Lógica Difusa.

Una aportación muy particular de la presente tesis está relacionada con la conceptualización del sistema de control: el amortiguador usado en este sistema presenta el fenómeno de histéresis y no es lineal. Se busca limitar el movimiento del edificio de cinco pisos de tal forma que los materiales con los que fue construido, no salgan de su región plástica y elástica con sólo un amortiguador.

Organización de la tesis

Como hasta ahora se ha visto, en el capítulo uno se presenta el contexto y el alcance de la presente tesis, es decir, se describe el estado del arte, los antecedentes, el objetivo y las aportaciones de la misma. En el capítulo dos se presentan los modelos matemáticos del edificio de cinco pisos usado como plataforma de pruebas y del amortiguador magnetoreológico que hace las veces de actuador. Tomando en cuenta lo anterior, es posible considerar un modelo combinado basado en la estructura del edificio y el comportamiento del amortiguador con la finalidad de tener una implementación más simple.

En el capítulo tres se describe el diseño y planteamiento del sistema difuso que controlará el Amortiguador. Se consideran tres sistemas difusos. El primero tiene una entrada (velocidad del primer piso) y una salida (voltaje para el amortiguador). El segundo, también tiene una entrada (desplazamiento del primer piso) y una salida (voltaje para el amortiguador) y finalmente un sistema de dos entradas (velocidad y desplazamiento del primer piso) y una salida (voltaje para el amortiguador).

El capítulo cuatro consiste en la descripción de las simulaciones que se han llevado a cabo para probar la funcionalidad de cada sistema difuso planteado en el capítulo tres, así como un análisis comparativo de efectividad entre los tres sistemas desarrollados. El capítulo cinco presenta las conclusiones generales de la presente tesis así como el trabajo futuro a desarrollar. Finalmente, se presentan las referencias consultadas así como los apéndices en los que se explica más a detalle los conceptos de la Lógica difusa y se presenta el código fuente del amortiguador, del edificio y del controlador difuso.

Capítulo 2
Modelado.

Tomando en cuenta la naturaleza del problema a tratar en la presente tesis, no es posible usar modelos físicos reales para llevar a cabo las pruebas necesarias al sistema difuso debido a que experimentar con edificios reales resultaría muy costoso e inclusive peligroso.

Una de las alternativas más confiables para poder llevar a cabo estas pruebas al sistema difuso consiste en usar los modelos matemáticos que representan el comportamiento físico de la planta y del actuador, es decir, del edificio y del amortiguador, respectivamente.

Modelo del amortiguador magnetoreológico

Existen materiales conocidos como "materiales inteligentes", los cuales tienen la particularidad de que algunas de sus propiedades pueden ser controladas y cambiadas a través de algún estímulo externo. Algunos ejemplos de este tipo de materiales son:

- Materiales piezoeléctricos
- Materiales con efecto térmico de memoria
- Materiales con efecto magnético de memoria
- Materiales magnetoreológicos
- Polímeros sensitivos al Ph
- Materiales halocrómicos

Los materiales magnetoreológicos están constituidos por partículas magnetizables divididas finamente que están suspendidas en un líquido portador lo suficientemente elástico para permitir la orientación de las partículas. Estos fluidos responden a la aplicación de un campo magnético con un cambio en su comportamiento reológico, es decir, qué tanto esfuerzo se necesita para poder deformar el material para que éste fluya.

La respuesta producida en estos fluidos resulta de la polarización en las partículas suspendidas mediante la aplicación de un campo magnético externo. La interacción entre los dipolos resultantes produce que las partículas suspendidas formen estructuras en forma de columna y sean paralelas al campo magnético aplicado. Este tipo de estructuras que se forman, restringe el movimiento del fluido originando un incremento de viscosidad en la suspensión. La energía mecánica necesaria para producir estas estructuras se incrementa conforme al aumento del campo magnético aplicado, lo que produce un esfuerzo dependiente del campo. Cuando no se aplica un campo magnético, este tipo de fluidos muestran un comportamiento Newtoniano, es decir, permanecen con una viscosidad constante.

Los fluidos magnetoreológicos son usados principalmente en amortiguadores para aplicación en automóviles y en amortiguadores en construcción civil, como pueden ser puentes, edificios, etc. Sin embargo, debido a factores tales como temperatura, interacciones entre las partículas, concentración de las partículas, sedimentación, forma, dimensión, viscosidad del líquido portador (Jiménez y Álvarez 2004), estos dispositivos poseen histéresis y una alta no linealidad, lo que complica el modelado matemático de los mismos. En la Figura 2.1 se muestra un diagrama que contiene los componentes de un amortiguador magnetoreológico.

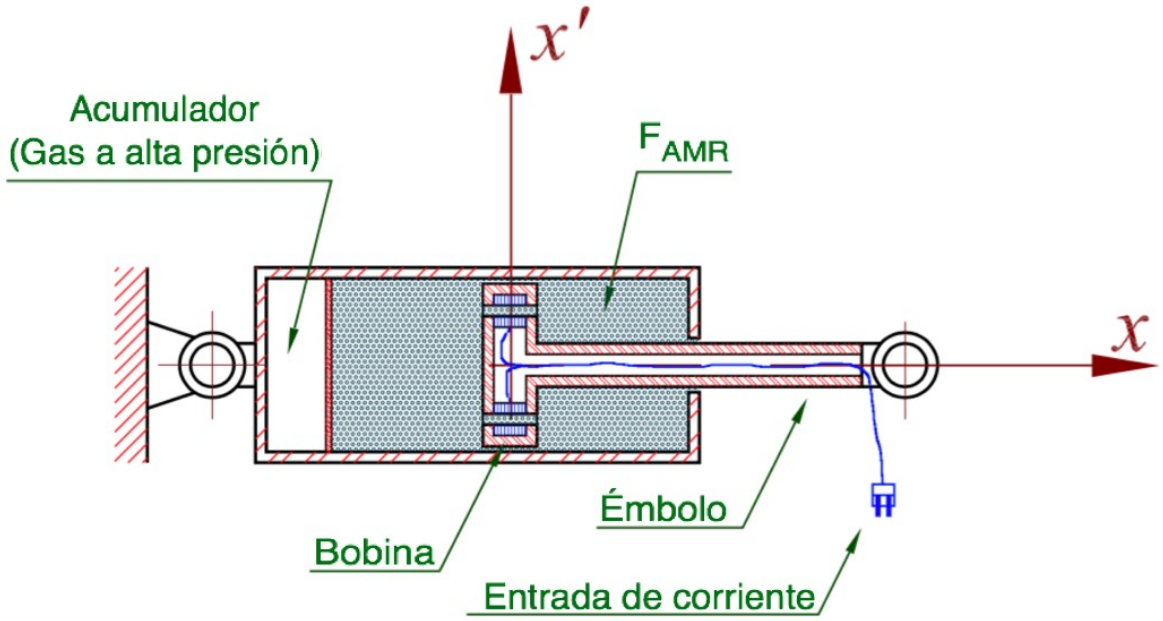

Figura 2.1 Diagrama de un amortiguador magnetoreológico

Existen algunos modelos matemáticos basados en la dinámica de la fricción del fluido que intentan describir el comportamiento de estos amortiguadores; muchos de ellos muy precisos, no obstante, muy complejos para poder usarse en el paradigma de control. Sin embargo, existe una versión modificada del modelo de fricción de LuGre (Jiménez y Álvarez 2004), nombrado así en honor a las universidades de donde proceden sus autores: Karl Johan Aström, de la Universidad de Lund, Suecia y Carlos Canudas de Wit, de la Universidad de Grenoble, Francia. Este modelo reproduce con una aceptable precisión la respuesta del amortiguador magnetoreológico. Este modelo es más simple comparado con los antes mencionados por lo que resulta una opción ideal para usarse en el paradigma de control. A continuación se presenta el modelo modificado de LuGre:

$$F_{AMR} = \sigma_0 z\upsilon + \sigma_1 \dot{z} + \sigma_2 \dot{x}$$

$$\dot{z} = \dot{x} - \sigma_0 a_0 \left(1 + a_1 \upsilon\right)\left|\dot{x}\right| z$$

Donde F_{AMR} es la fuerza que ejerce el amortiguador. La variable z representa un estado interno que es el promedio de la deformación transitoria que se genera en el fluido del amortiguador cuando el émbolo comienza a moverse en una dirección o en otra. Es importante mencionar que el estado z no es medible directamente, sin embargo, es posible estimarlo a través de mediciones de aceleración o fuerza. La variable $\dot{z}$ representa la velocidad relativa que se presenta en la deformación en el fluido. La variable $\dot{x}$, medida en [m/s], representa la velocidad relativa entre cada extremo del amortiguador. La variable υ, medida en [v] es el voltaje aplicado al amortiguador. Las variables σ_0[N/mV], σ_1[Ns/m] y σ_2[Ns/m] son parámetros constantes de las propiedades físicas del fluido magnetoreológico. Las variables a_0[V/N] y a_1[V^{-1}] son constantes propias del efecto Stribeck, observado en el fenómeno de fricción a bajas velocidades, es decir, la fricción va disminuyendo a medida que aumenta la velocidad.

Considerando la dinámica del modelo anteriormente descrito y la arquitectura de la implementación del mismo (de la que posteriormente se tratará), es necesario calcular el valor del estado interno en el tiempo *k+1* para tener continuidad en el valor del estado interno. Este valor se obtiene por medio de la siguiente ecuación:

$$z(k+1)T = z(kT) + T\left[\dot{x}(kT) - \sigma_0 a_0 \left(1 + a_1 \upsilon\right)\left|\dot{x}(kT)\right| z(kT)\right]$$

Donde *T* es el tiempo de muestreo de la señal del fenómeno a controlar.

Una vez definido el modelo matemático que representa el comportamiento del amortiguador, es necesario definir las constantes involucradas en el modelo matemático que son indispensables para la implementación del modelo computacional que representará al modelo del amortiguador en las pruebas de comportamiento y eficacia del sistema difuso.

La tabla 2.1 muestra los valores nominales asignados a cada constante del modelo. Estos valores han sido tomados de(Álvarez y Jiménez 2004).

Tabla 2.1 Parámetros del amortiguador magnetoreológico

Constante	Valor
σ_0	105,900 [N/mV]
σ_1	5,700 [Ns/m]
σ_2	2,300 [Ns/m]
a_0	0.0030 [V/N]
a_1	-0.1444 [V^{-1}]

Es importante agregar que el valor del tiempo de muestreo *T* para la anterior ecuación es 0.005 [s].

Para poder implementar el modelo computacional del amortiguador se ha determinado el uso del paradigma de programación orientado a objetos debido a que este paradigma, entre otras ventajas ofrece:

- Reusabilidad: Las clases se pueden usar en otras partes del código o en otros proyectos.

- Mantenibilidad: Facilidad para modificar algún componente del proyecto con el fin de eliminar errores o mejorar el rendimiento.

- Modularidad: Capacidad de dividir el proyecto en partes más pequeñas y que sean lo más independientes posible unas con respecto de otras.

La Figura 2.2 muestra dos clases que interactúan entre sí para representar el comportamiento del amortiguador magnetoreológico.

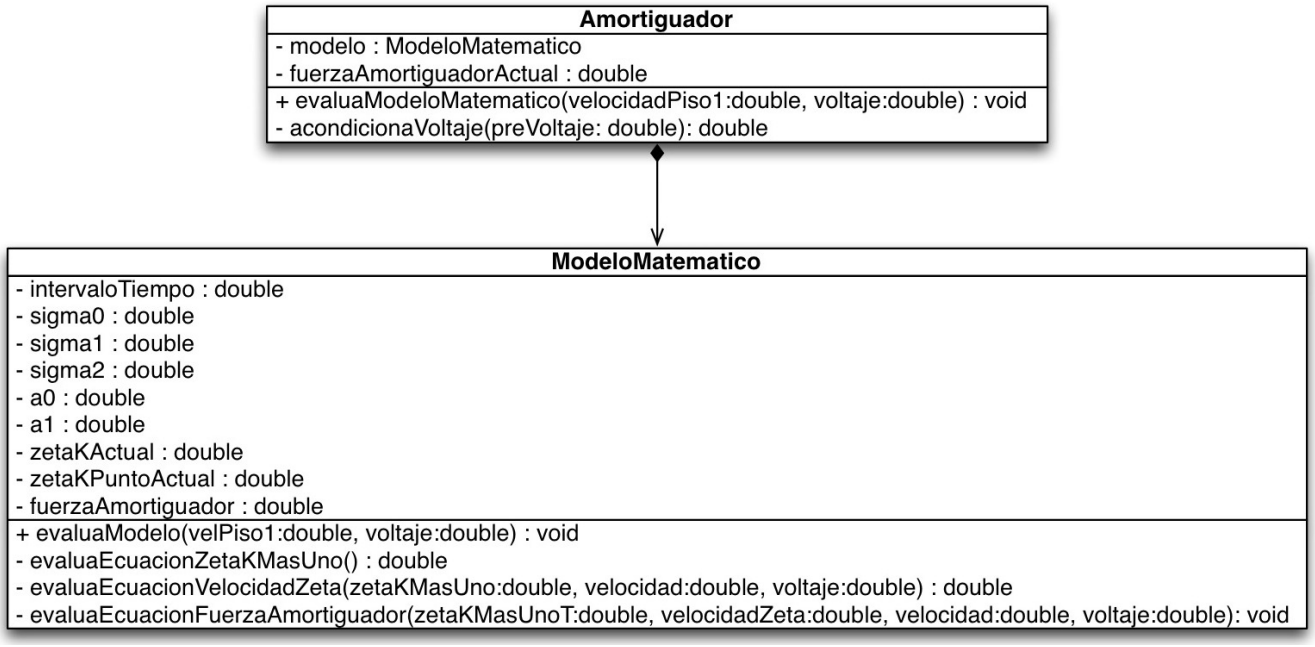

Figura 2.2 Diagrama de clases del amortiguador magnetoreológico

Este diagrama tiene una clase llamada "Amortiguador" que tiene como atributos un objeto de la clase "ModeloMatematico" y un valor que representa la fuerza que ejerce el amortiguador. Tiene un método que sirve para recibir un valor de velocidad (la velocidad del primer piso del edificio) y otro de voltaje; el último método sirve para adecuar el valor del voltaje recibido.

La otra clase, llamada "ModeloMatematico" tiene como atributos las constantes del modelo matemático y tres valores que representan el valor del estado z, la velocidad de z en el tiempo actual y el valor de la fuerza del amortiguador calculada. El primer método de esta clase recibe como parámetros una velocidad y un voltaje. Cada método siguiente representa una ecuación del modelo matemático respectivamente. En general, el nombre de cada método describe su propia función.

Modelo del edificio

Como ya se ha mencionado en el capítulo anterior, los sismos tienen terribles consecuencias, particularmente en las edificaciones civiles debido a que este tipo de movimiento produce esfuerzos en los componentes estructurales que a su vez pueden producir fallas e inclusive el colapso de la edificación. Es por esto, que es importante conocer el comportamiento que las edificaciones civiles tienen bajo la acción de un sismo.

El modelo matemático seleccionado describe el comportamiento elástico a cortante del edificio (Morales y Álvarez-Icaza 2013). Este modelo considera al edificio como una estructura cuyo número de pisos es equivalente a hablar del mismo número de grados de libertad. La masa de cada piso está concentrada en el centro del techo de cada uno de ellos. Las columnas son flexibles a la deformación lateral y son rígidas a la dirección vertical.

La estructura está sobre el suelo firme, el movimiento que presenta la estructura es únicamente sobre un eje coordenado y no se presentan efectos de torsión. En la Figura 2.3 se muestra el diagrama de un edificio de *n* pisos considerando el modelo descrito anteriormente.

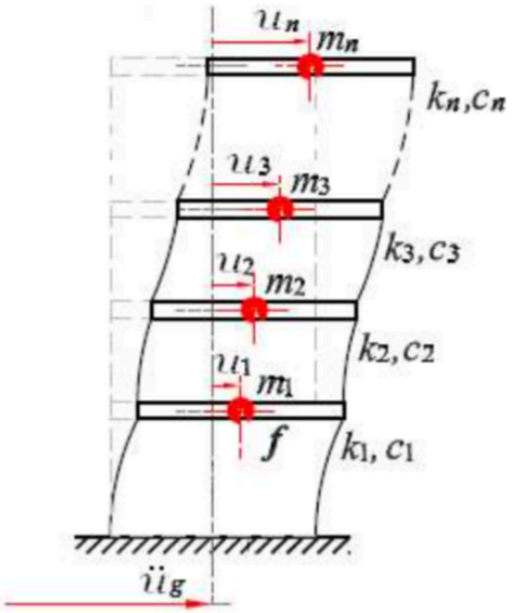

Figura 2.3 Diagrama de un edificio de *n* pisos o de *n* grados de libertad

El modelo matemático se presenta a continuación:

$$M\ddot{x} + Kx + C\dot{x} = -M\ddot{x}_g$$

Donde *M* es la matriz de masa. Contiene en su diagonal los valores de masa de cada uno de los pisos m_i. La matriz *K* es la matriz de rigidez que contiene los valores de rigidez por cada piso, y la matriz *C* es la matriz de amortiguamiento que contiene los valores de amortiguamiento por cada piso. Los vectores $\ddot{x}$, $\dot{x}$, x representan la aceleración, la velocidad y el desplazamiento de cada grado de libertad, respectivamente. El vector $\ddot{x}_g$ representa la aceleración de la tierra que se aplica a todos los pisos.

Respectivamente, las matrices de masa [kg], rigidez [N/m] y amortiguamiento [Ns/m], están definidas de la siguiente forma:

$$M = \begin{bmatrix} m_1 & 0 & 0 & \dots & 0 \\ 0 & m_2 & 0 & \dots & 0 \\ 0 & 0 & m_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & m_n \end{bmatrix} > 0$$

$$K = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & \dots & 0 \\ -k_2 & k_2 + k_3 & -k_3 & \dots & 0 \\ 0 & -k_3 & k_3 + k_4 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & k_n \end{bmatrix} \geq 0$$

$$C = \begin{bmatrix} c_1 + c_2 & -c_2 & 0 & \dots & 0 \\ -c_2 & c_2 + c_3 & -c_3 & \dots & 0 \\ 0 & -c_3 & c_3 + c_4 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & c_n \end{bmatrix} > 0$$

Es posible considerar que este modelo matemático cumple con un comportamiento lineal, diafragma de piso rígido, parámetros concentrados y marcos planos.

Una forma confiable y precisa de cuantificar la eficacia del sistema difuso es medir el desplazamiento y velocidad de los pisos del edificio cuando éste se ve afectado por la acción del sismo y del amortiguador. A partir del modelo matemático antes descrito, es posible obtener estos dos valores. Después de aplicar el álgebra matricial correspondiente, el modelo puede plantearse de la siguiente forma:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} - \begin{bmatrix} 0 \\ \ddot{x}_g \end{bmatrix}$$

Una vez replanteado el modelo, es posible obtener el vector de velocidades y desplazamientos realizando las operaciones necesarias y considerando que estos dos valores deben tener continuidad para conocer el comportamiento del edificio en el momento de la excitación externa, es necesario calcularlos para el tiempo *k+1* empleando la llamada descretización de Euler:

$$\dot{x}(k+1)T = \dot{x}(kT) + T\left[-M^{-1}Kx(kT) - M^{-1}C\dot{x}(kT) - \ddot{x}_g(kT)\right]$$

$$x(k+1)T = x(kT) + T\left[\dot{x}(kT)\right]$$

Donde *T* es el tiempo de muestro de la señal del fenómeno a controlar.

Definido el modelo matemático del edificio, el siguiente paso es definir el modelo computacional que lo representará. Para ello, es necesario definir las matrices de masa, rigidez y amortiguamiento que serán usadas. Es importante mencionar que el edificio que se ha modelado consta de cinco pisos, por lo que las matrices tienen una dimensión de 5 x 5. Los valores nominales de éstas, fueron tomados de (Morales y Álvarez-Icaza 2013).

$$M = \begin{bmatrix} 9.5 & 0 & 0 & 0 & 0 \\ 0 & 9.5 & 0 & 0 & 0 \\ 0 & 0 & 9.5 & 0 & 0 \\ 0 & 0 & 0 & 9.5 & 0 \\ 0 & 0 & 0 & 0 & 9.5 \end{bmatrix} [kg]$$

$$K = \begin{bmatrix} 26315 & -10748 & 0 & 0 & 0 \\ -11796 & 24327 & -12530 & 0 & 0 \\ 0 & -12800 & 25215 & -12416 & 0 \\ 0 & 0 & -12863 & 25946 & -13083 \\ 0 & 0 & 0 & -12427 & 12427 \end{bmatrix} [N/m]$$

$$C = \begin{bmatrix} 18.5325 & -6.0971 & 0 & 0 & 0 \\ -24.5452 & 33.6949 & -9.1498 & 0 & 0 \\ 0 & -11.8545 & 16.9426 & 5.0881 & 0 \\ 0 & 0 & -5.472 & 6.1502 & -0.6783 \\ 0 & 0 & 0 & -2.5259 & 2.5259 \end{bmatrix} [Ns/m]$$

Análogamente al modelo del amortiguador, el valor del tiempo de muestreo *T* para las anteriores ecuaciones es $0.005/\Delta$.

Como se mencionó anteriormente, el paradigma de programación orientado a objetos se ha seleccionado para la implementación de los modelos computacionales. La Figura 2.4 muestra dos clases que interactúan entre sí para representar el comportamiento del edificio.

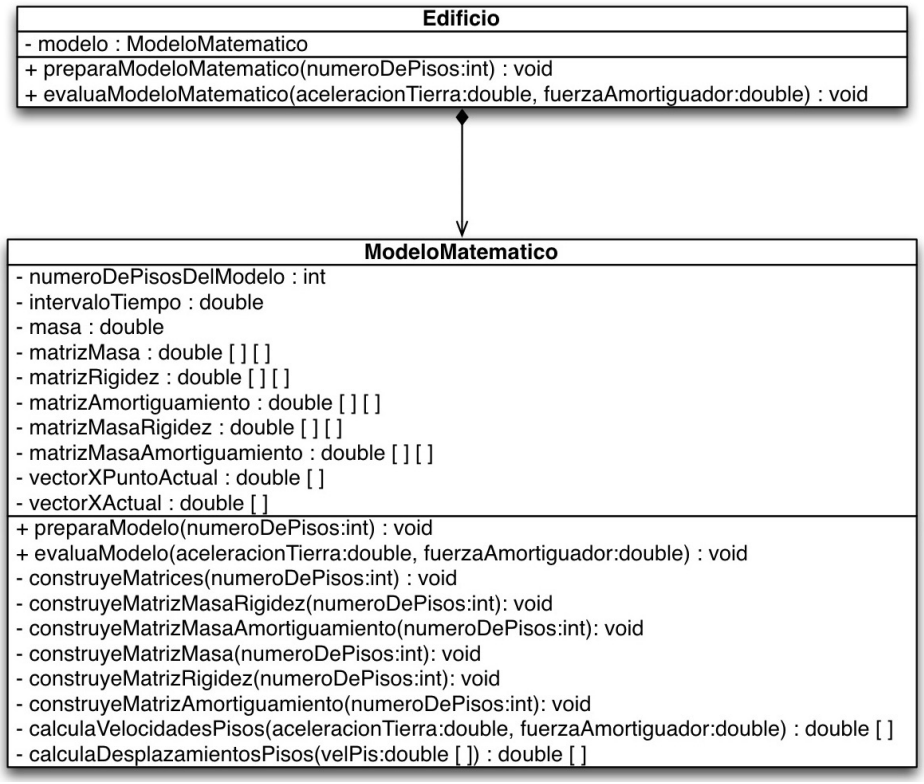

Figura 2.4 Diagrama de clases del edificio de cinco pisos

Este diagrama tiene dos clases. La clase "Edificio" tiene un atributo que es un objeto de la clase "ModeloMatematico". El primer método recibe la cantidad de pisos de edificio y se lo envía al modelo matemático. El segundo método recibe el valor de la aceleración de la tierra y la fuerza del amortiguador para enviarlos al modelo matemático.

La clase "ModeloMatematico" tiene como atributos, las matrices de masa, rigidez y amortiguamiento así como también las matrices resultantes de multiplicar la matriz de masa por la matriz de rigidez y la matriz de masa por la de amortiguamiento. Tiene también como atributos el vector de velocidades de los pisos así como también el vector de desplazamientos de los mismos. El primer método define el tamaño de las matrices y rellena las matrices que tienen valores constantes. El siguiente método recibe el valor de la aceleración de la tierra y la fuerza de amortiguador para calcular la velocidad y desplazamiento de cada piso.

Los métodos siguientes son auxiliares de los métodos descritos en el párrafo anterior, es decir, ayudan a construir las matrices de masa, amortiguamiento y rigidez que son necesarias para calcular posteriormente las velocidades y desplazamientos de cada piso al aplicarse la aceleración de la tierra y la fuerza del amortiguador. En general, el nombre de cada método auxiliar describe su propia función.

Capítulo 3
Sistema difuso.

Hasta este punto, los elementos con los que va a interactuar el sistema difuso están definidos. Desde una perspectiva más general, la planta y el actuador o el edificio y el amortiguador, respectivamente, constituyen la plataforma de pruebas del sistema difuso.

A grandes rasgos, el sistema difuso busca atenuar las vibraciones en un edificio producidas por un sismo, a través de la acción del amortiguador magnetoreológico. Un sistema difuso funciona mediante la llamada "Lógica difusa"(ver Apéndice A).

Mapeo entrada - salida

Para poder alcanzar el objetivo del sistema difuso, el primer paso, consiste en definir las variables de entrada y variables de salidas del mismo.

Tomando en cuenta lo planteado en el capítulo 2, es posible cuantificar la respuesta del edificio con respecto al sismo mediante los desplazamientos y velocidades de cada piso que lo conforma. Con base en lo anterior, es plausible tomar como entradas del controlador difuso estas dos mediciones ya que éstas, son las variables de estado del modelo del edificio.

Se sabe que el amortiguador magnetoreológico entra en funcionamiento a través de una señal de voltaje, por lo que se puede tomar como salida del controlador difuso esta señal. Considerando lo anterior, se puede plantear el diagrama que se muestra en la Figura 3.1.

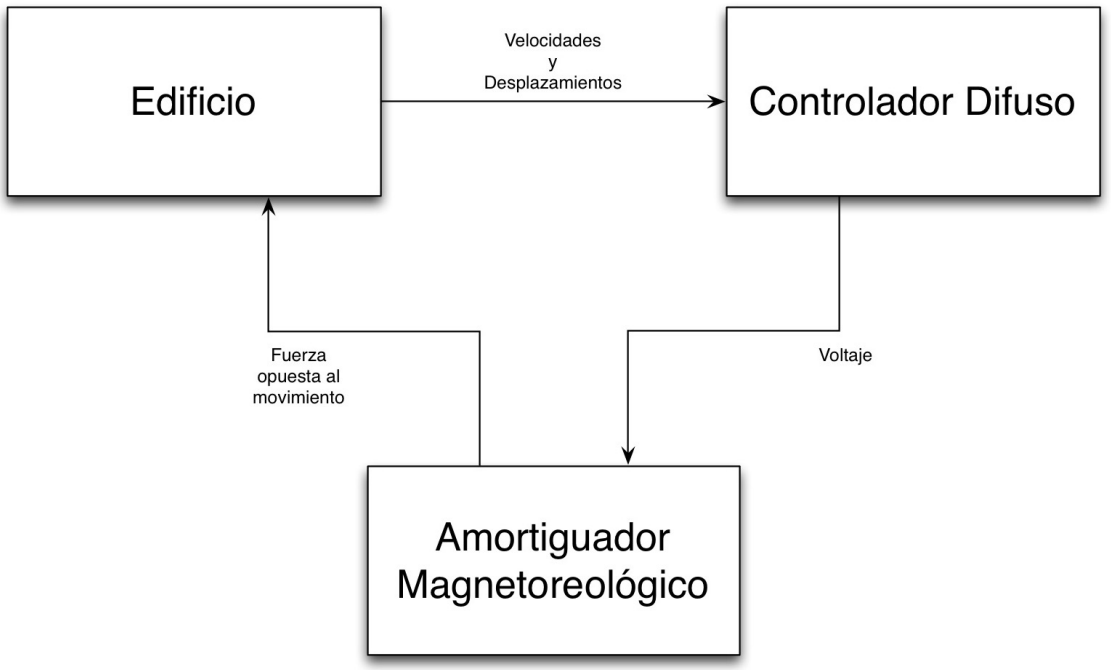

Figura 3.1 Diagrama del bloques del sistema difuso

El diagrama muestra que las entradas del sistema difuso son las velocidades y los desplazamientos del edificio obtenidos a través de mediciones de acelerómetros; éste las procesa y arroja como salida una señal de voltaje que recibe el amortiguador para que se active y genere una fuerza opuesta al movimiento del edificio, provocando una atenuación en las velocidades y desplazamientos de éste, generando un lazo cerrado. Desde la perspectiva del paradigma de control, el anterior diagrama puede replantearse como se muestra en la Figura 3.2.

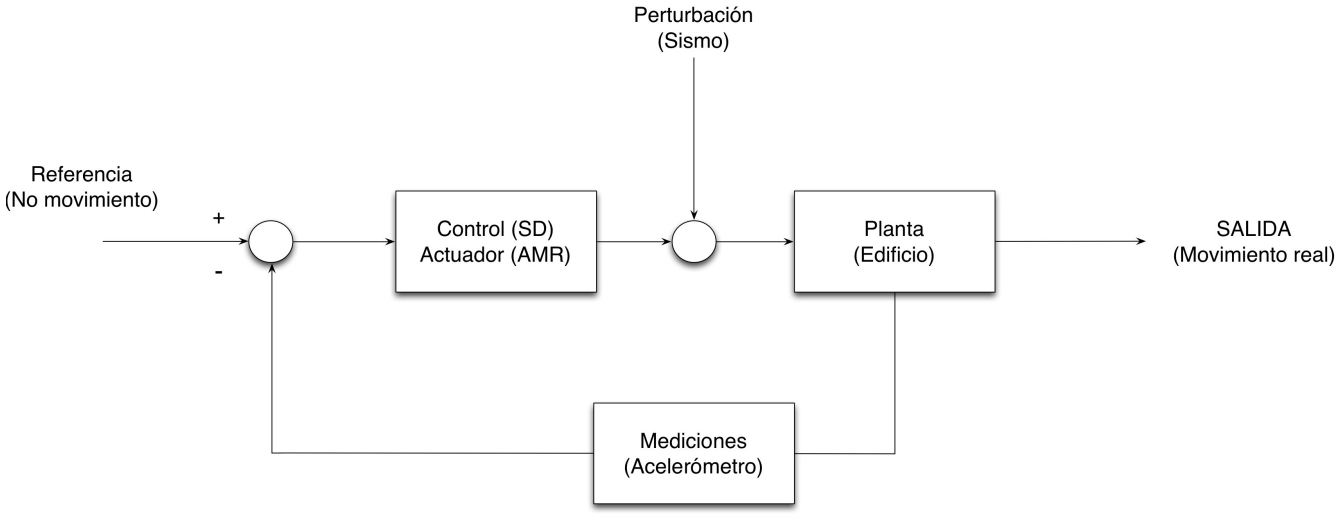

Figura 3.2 Diagrama de control de lazo cerrado del sistema difuso

Este diagrama muestra de forma más clara el objetivo del sistema difuso y el flujo de información para lograrlo. Se busca que la salida sea lo más parecida a la referencia, es decir, que el movimiento real del edificio sea el menor posible. Esto, mediante la acción del sistema difuso, que recibe como entradas las mediciones del acelerómetro y que es capaz de controlar al amortiguador a través de una señal de voltaje.

Como puede observarse, el controlador difuso es de tipo MISO (Multiple Input Single Output) ya que a partir de velocidades y desplazamientos, se obtiene voltaje.

Definición del universo del discurso

Después de definir las variables de entrada y salida del sistema difuso, es necesario asociarlas a una variable lingüística y definir un universo del discurso para cada una. Esto se lleva a cabo analizando el comportamiento de cada variable lingüística de entrada para definir el rango de operación y a su vez, los conjuntos difusos asociados. La Figura 3.3 describe cómo es el movimiento del edificio cuando está sujeto al movimiento sísmico así como la ubicación del amortiguador magnetoreológico.

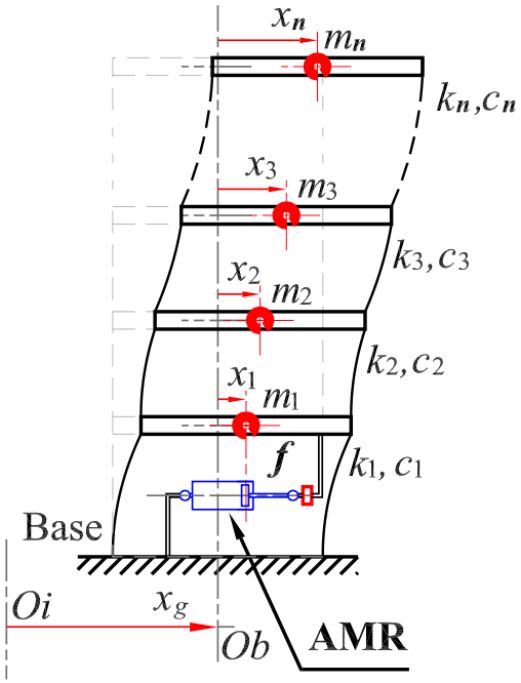

Figura 3.3 Diagrama que describe el movimiento del edificio al aplicarse un movimiento sísmico

Como puede apreciarse en el diagrama, el edificio puede oscilar sobre el eje de referencia dependiendo a la dirección de la fuerza sísmica, por lo tanto, las variables lingüísticas de entrada "Velocidad· y "Desplazamiento" pueden tomar valores negativos o positivos con respecto al eje de referencia. Una vez definido esto, se puede considerar que si el valor de las variables lingüísticas es 0, el edificio está en reposo, si el valor de las variables es mayor a 0, se considera un desplazamiento o velocidad positivos y si el valor de las variables es menor a 0, se considera un desplazamiento o velocidad negativos.

El universo del discurso de las variables de entrada está conformado por cinco conjuntos difusos simétricos con respecto a 0. Esta simetría se debe a que se desea restituir el estado de reposo del edificio lo más pronto posible tomando como referencia el comportamiento del edificio descrito en el párrafo anterior, es decir, se deben considerar desplazamientos y velocidades positivas y negativas para el fin descrito anteriormente, según corresponda.

En las Figuras 3.4 y 3.5 se muestran los universos del discurso de cada variable lingüística de entrada.

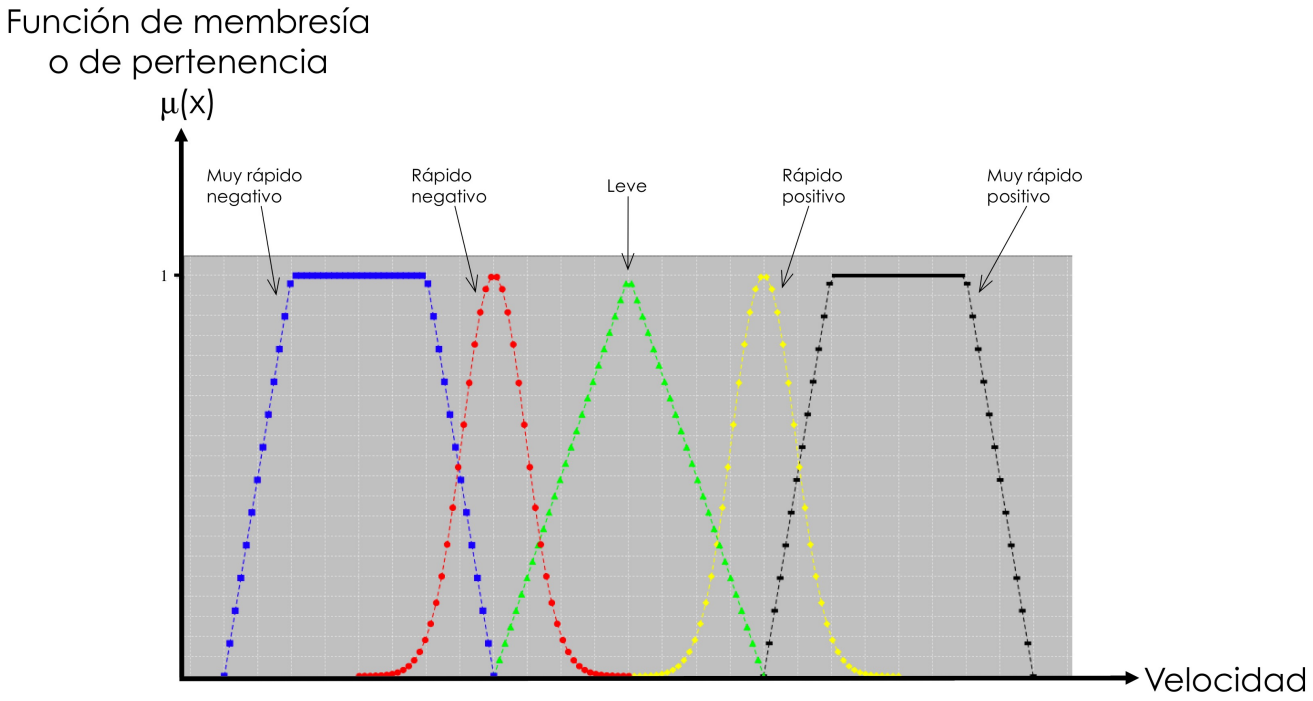

Figura 3.4 Universo del discurso de la variable de entrada "Velocidad"

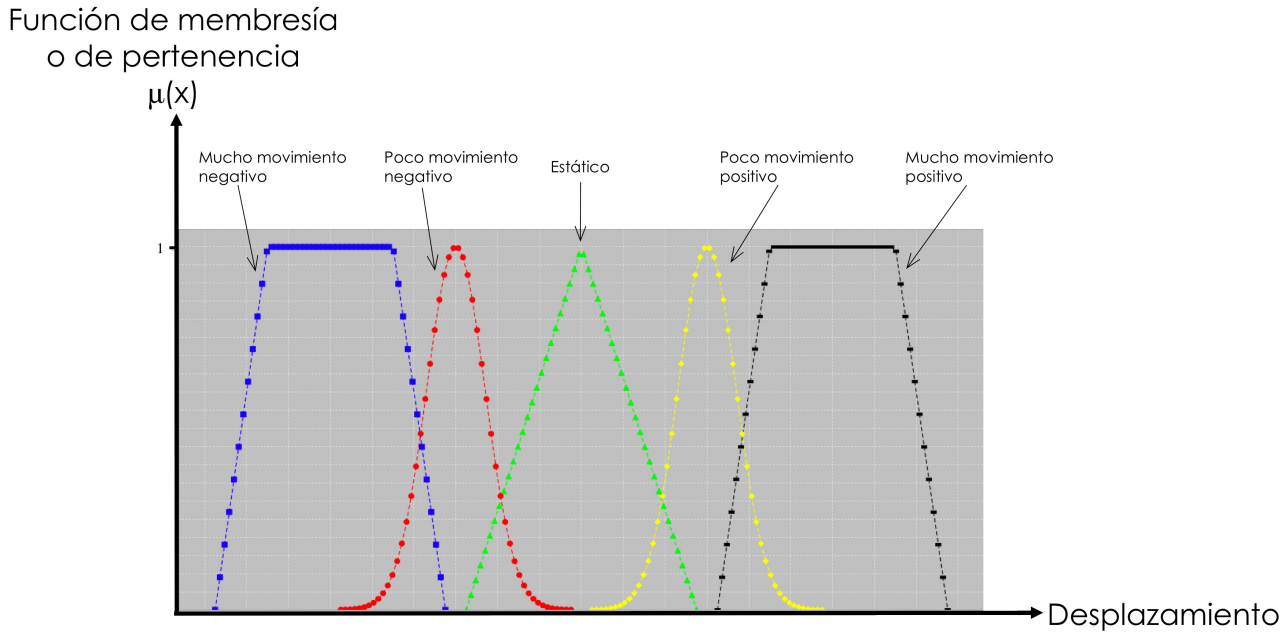

Figura 3.5 Universo del discurso de la variable de entrada "Desplazamiento"

De forma similar a la variables lingüísticas de entrada, el universo del discurso de la variable de salida se define de la misma forma que las variables de entrada, la única diferencia es que se aumenta un conjunto difuso del tipo singletone con el valor lingüístico "nulo". Esto se debe a que es necesario considerar el valor de voltaje 0 como valor mínimo ya que el amortiguador opera únicamente con voltaje positivo. En la Figura 3.6 se muestra el universo del discurso para la variable lingüística de salida.

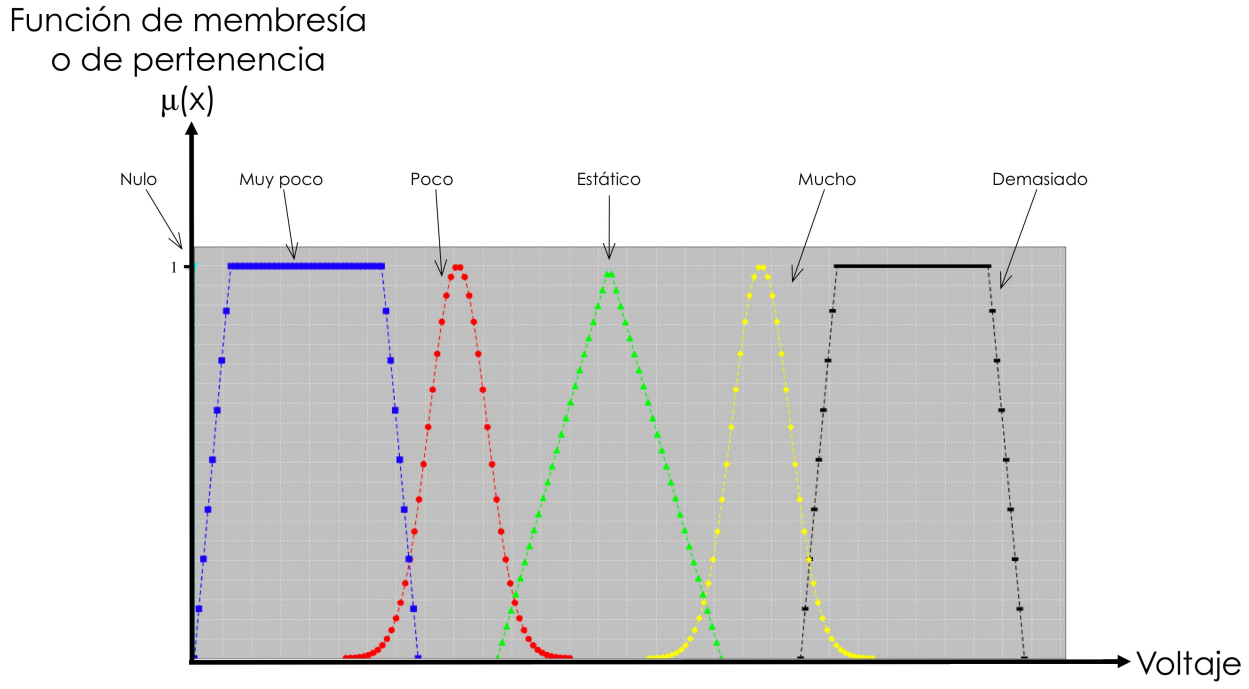

Figura 3.6 Universo del discurso de la variable de salida "Voltaje"

Como puede observarse, tanto las variables lingüísticas de entrada como la variable lingüística de salida tienen las mismas funciones de pertenencia en su correspondiente universo del discurso. Esto se debe a que se busca tener una relación 1 a 1 de los conjuntos difusos de entrada con respecto a los conjuntos difusos de salida para simplificar el proceso del difusor y del desdifusor.

Base de conocimiento y desdifusión

Para la elaboración de las reglas que conformarán la base de conocimiento que gobernará al sistema difuso, se ha seleccionado el formato Mandami debido a que la estructura que define a las reglas resulta ser más simple para su implementación.

Considerando que el amortiguador magnetorológico es capaz de ejercer fuerzas en un solo sentido con respecto al eje de referencia, algunas reglas de la base de conocimiento tienen como consecuente el valor lingüístico "Nulo". La Tabla 3.1 muestra el formato de algunas reglas las cuales constituyen un ejemplo de una base de conocimiento:

Tabla 3.1 Ejemplo de una base de conocimiento para el controlador difuso

Regla	Antecedentes	Consecuentes
1	Si velocidad es "Muy rápido negativo" y desplazamiento es "Mucho movimiento negativo"	Entonces voltaje es "Nulo"
2	Si velocidad es "Rápido negativo" y desplazamiento es "Poco movimiento negativo"	Entonces voltaje es "Nulo"
3	Si velocidad es "Leve" y desplazamiento es "Estático"	Entonces voltaje es "Moderado"
4	Si velocidad es "Rápido positivo" y desplazamiento es "Poco movimiento positivo"	Entonces voltaje es "Mucho"
5	Si velocidad es "Muy rápido positivo" y desplazamiento es "Mucho movimiento positivo"	Entonces voltaje es "Demasiado" y "Mucho"

Es importante aclarar que en los consecuentes de la base de conocimiento anterior, el orden de aparición de los conjuntos difusos asociados a los consecuentes es determinante, esto quiere decir que "Demasiado y Mucho" no es equivalente a "Mucho y Demasiado" debido a que el primer conjunto tiene un valor de pertenencia calculado mayor con respecto al segundo.

Como se sabe, el conjunto de reglas de la base de conocimiento se construye a partir del conocimiento o experiencia del experto. En este caso, la base de conocimiento se construyó empíricamente, es decir, se buscó que el valor o valores del antecedente de la regla sean neutralizados con el o los valores del consecuente de la misma, de tal forma que si existe un desplazamiento y velocidad considerable en el edificio, el amortiguador genere la fuerza suficiente para contrarrestar o atenuar lo más posible el desplazamiento y velocidad del edificio. Análogamente, si el o los consecuentes tienen valores que impliquen un movimiento poco significativo en el edificio, el amortiguador ejerce una fuerza mínima o nula.

Sin embargo, la base de conocimiento puede optimizarse a través de alguna técnica de inteligencia artificial. Los Algoritmos Genéticos resultan ser una opción confiable debido a que esencialmente se utilizan para optimizar procesos o funciones.

En cuanto al proceso que lleva a cabo el desdifusor, es decir, convertir el o los conjuntos difusos resultantes en un valor real o "crisp", se ha seleccionado el método del centroide debido a su exactitud para calcular el valor de salida y a que su complejidad es relativamente baja, por lo que computacionalmente es el menos costoso.

Implementación del controlador difuso

Una vez definidas las variables de entrada y salida, y el formato de la base de conocimiento así como las reglas que la conforman, es posible asumir que la arquitectura del controlador difuso ha sido diseñada. La figura 3.7 muestra esta arquitectura.

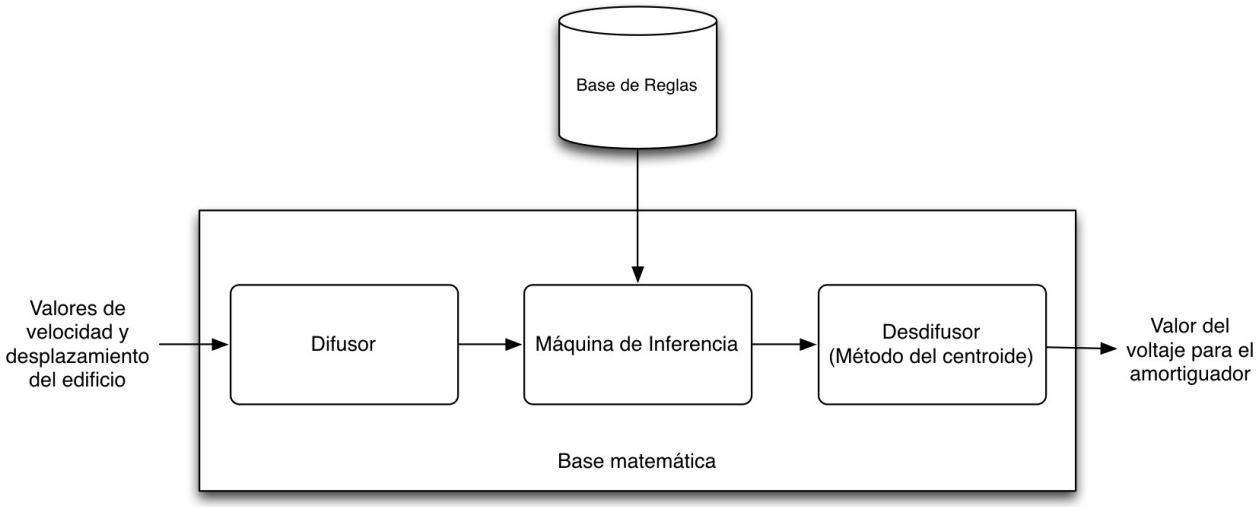

Figura 3.7 Arquitectura del controlador difuso diseñado

Hasta este punto, el diseño del controlador difuso está completo, sin embargo, es necesario abstraerlo al paradigma de programación orientado a objetos.

La figura 3.8 muestra cuatro clases. La clase "ControladorDifuso" tiene como atributos los valores de entrada, es decir, la velocidad y el desplazamiento, tiene el valor de salida, o sea, el voltaje. Tiene también como atributos un objeto de la clase "Difusor", un objeto de la clase "MaquinaDeInferencia" y un objeto de la clase "Desdifusor".

El primer método se encarga de configurar cada componente del controlador difuso, es decir, inicializa el universo del discurso correspondiente a cada variable de entrada en el difusor, incializa la máquina de inferencia construyendo la base de conocimiento y define el universo del discurso de la variable de salida en el desdifusor.

El segundo método, de acuerdo a lo expuesto en la primer sección de este capítulo, hace uso de los métodos internos de cada objeto que representa un componente de la arquitectura del controlador difuso para obtener el voltaje a partir de la velocidad y el desplazamiento. El último método limpia las variables de entrada y salida para evitar tener datos basura guardados.

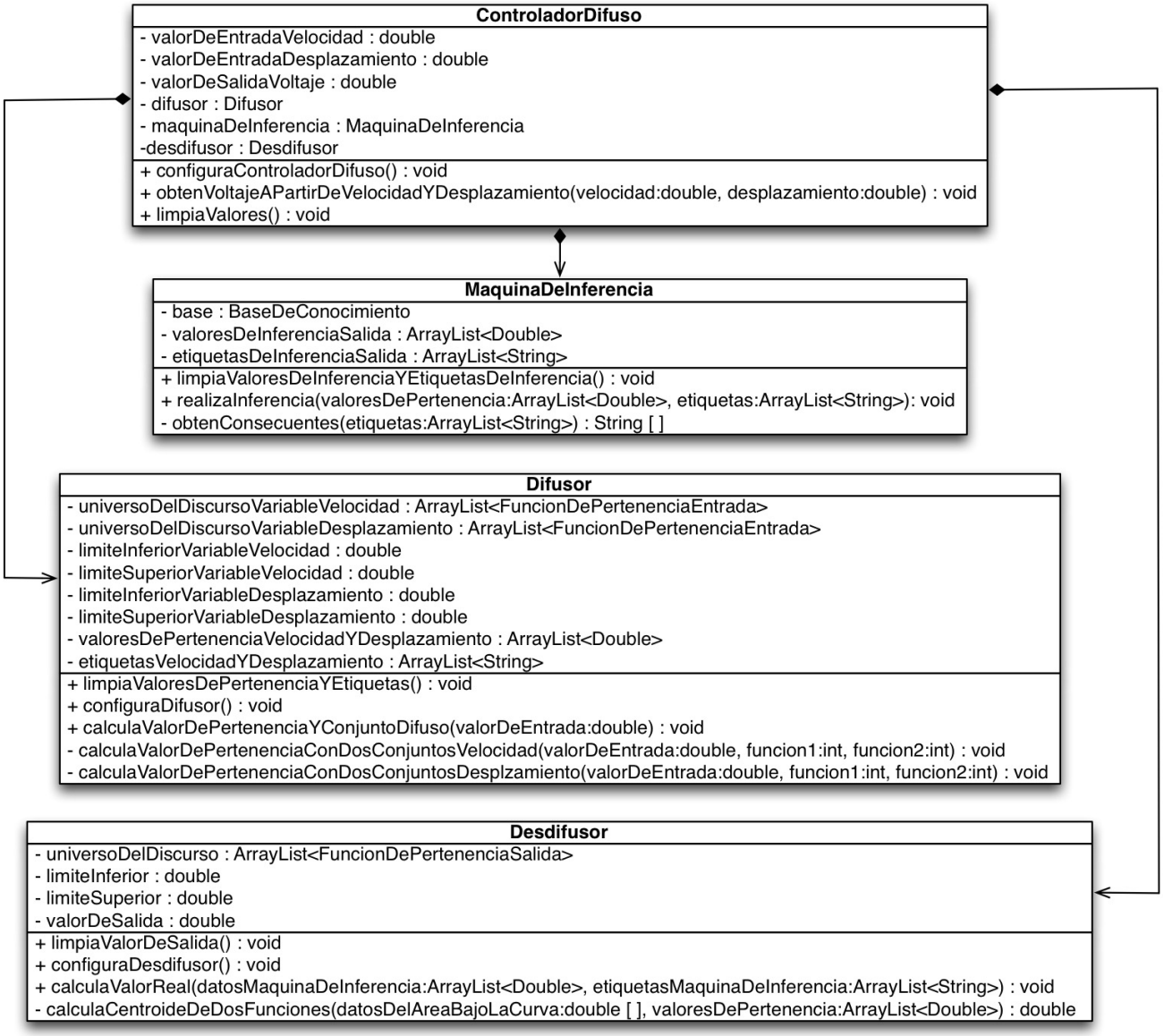

Figura 3.8 Diagrama de clases del controlador difuso

Capítulo 4
Simulaciones numéricas.

Una vez que se han definido los elementos que conforman al sistema difuso, el siguiente paso es probarlo. Para ello, hay que llevar a cabo las simulaciones pertinentes para determinar qué tan eficaz es la atenuación del movimiento de un edificio provocado por un sismo. Una simulación consiste en aplicar una señal de excitación externa que representa la aceleración del suelo al modelo matemático del edificio. El modelo recibe la aceleración del suelo, la procesa y arroja la velocidad y desplazamiento de cada piso. Los valores de salida del modelo son la entrada del controlador difuso que a su vez, los procesa y arroja un valor que representa el voltaje que debe aplicarse al amortiguador para que éste ejerza una fuerza que trate de mitigar el movimiento en el edificio. La respuesta de cada piso del edificio durante el periodo de excitación externa se muestra en una gráfica.

Para llevar a cabo estas simulaciones, se seleccionó como excitación externa del edificio el registro de aceleraciones del terremoto de Loma Prieta, ocurrido el martes 17 de octubre de 1989 en el área de la bahía de San Francisco, en California. Marcó 6.9 en la escala sismológica de magnitud de momento. Esta magnitud es logarítmica y se basa en la medición de la energía total que se libera durante el sismo; es muy parecida a la escala de Richter. Se han implementado tres sistemas difusos con la finalidad de determinar qué entrada o conjunto de entradas definen un comportamiento más próximo al deseado. La siguiente tabla muestra la relación de los sistemas difusos implementados con respecto a sus entradas y salidas.

Tabla 4.1 Controladores difusos con sus variables de entrada y salida

Tipo	Número de entradas	Variables de entrada	Número de salidas	Variable de salida
A	1	velocidad	1	voltaje
B	1	desplazamiento	1	voltaje
C	2	velocidad, desplazamiento	1	voltaje

Es importante destacar que para estas simulaciones, solamente se considera un amortiguador que está instalado en la parte inferior del edificio y que a su vez, está conectado con el primer piso, por lo que únicamente se miden las velocidades y desplazamientos de éste. Dichas mediciones fungen como entrada del controlador difuso, según corresponda. Además, resulta más económico usar un solo amortiguador. El uso de un número mayor de amortiguadores no se exploró y es trabajo futuro.

Respuesta libre del edificio

Antes de llevar a cabo las simulaciones con los controladores difusos, es necesario tener una referencia para poder cuantificar qué tan eficaz es la acción de cada controlador difuso. Esta referencia es el comportamiento o respuesta del edificio al aplicársele el registro sísmico de Loma Prieta. A continuación se muestran las respuestas con respecto a la velocidad y al desplazamiento de cada piso que conforma al edificio.

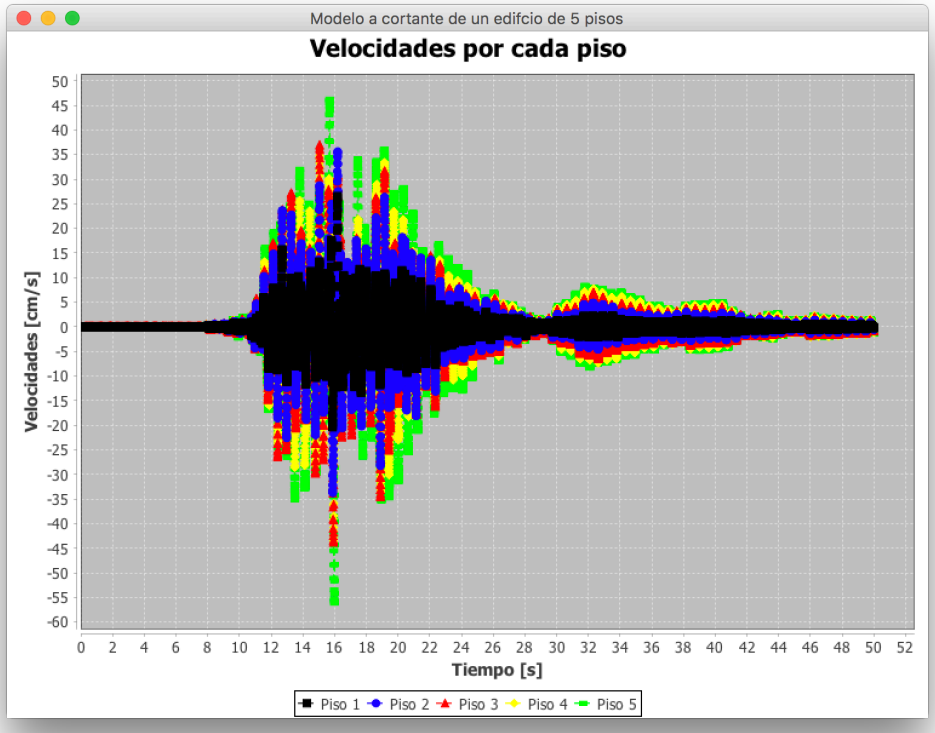

Figura 4.1 Respuesta con respecto a la velocidad del edificio de 5 pisos

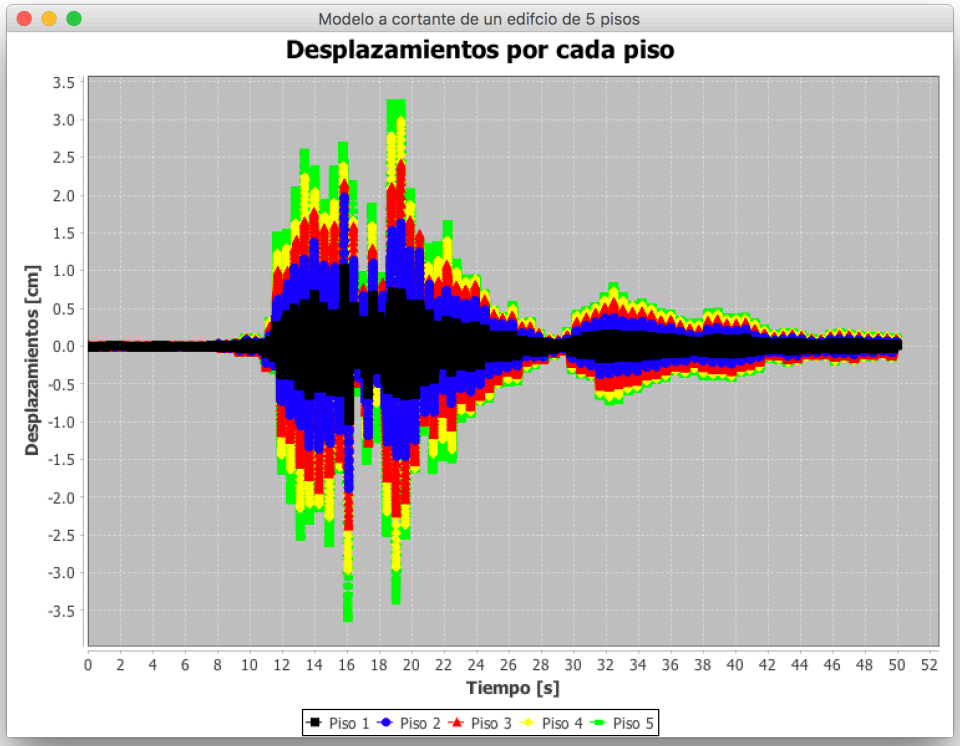

Figura 4.2 Respuesta con respecto al desplazamiento del edificio de 5 pisos

Como se mencionó en el capítulo 2 de la presente tesis, el tiempo de muestro es de 0.005 segundos, es decir, se tomaron 200 muestras por segundo y el periodo de excitación para esta simulación es de 50 segundos, por lo que el número de muestras total es de 10,000.

Como puede observarse en las anteriores figuras, el piso 5 es el más afectado por la aceleración del suelo. Considerando esto, el valor más alto registrado de velocidad positiva es 46.292 [cm/s] y el valor más bajo registrado de velocidad negativa es -56.178 [cm/s]. En cuanto a los desplazamientos, el valor positivo más alto registrado es 3.238 [cm] y el valor negativo más bajo registrado es -3.619 [cm].

Considerando estos valores, el valor máximo de respuesta del edificio se puede calcular de la siguiente forma:

$$ValorMáximoDeRespuesta = Max(ValorMaxAmplitudPositiva, |ValorMaxAmplitudNegativa|)$$

Dada la expresión anterior, es posible calcular los valores máximos de respuesta del edificio con respecto a la velocidad y al desplazamiento:

$$ValorMáximoDeRespuesta_{Velocidad} = Max(46.292, |-56.178|)$$
$$ValorMáximoDeRespuesta_{Velocidad} = 56.178 \text{ [cm/s]}$$

$$ValorMáximoDeRespuesta_{Desplazamiento} = Max(3.238, |-3.619|)$$
$$ValorMáximoDeRespuesta_{Desplazamiento} = 3.619 \text{ [cm]}$$

Controlador difuso tipo A

Este controlador difuso tiene una entrada que es la velocidad del primer piso del edificio y como salida el voltaje que hay que aplicar al amortiguador.

Dado que este controlador difuso tiene una entrada y una salida, la base de conocimiento es distinta con respecto a la planteada en el capítulo 3. A continuación se muestra la base de conocimiento que se usó para este controlador difuso.

Tabla 4.2 Base de conocimiento para el controlador difuso tipo A

Regla	Antecedentes	Consecuentes
1	Si velocidad es "Muy rápido negativo"	Entonces voltaje es "Nulo"
2	Si velocidad es "Rápido negativo"	Entonces voltaje es "Nulo"
3	Si velocidad es "Leve"	Entonces voltaje es "Moderado"
4	Si velocidad es "Rápido positivo"	Entonces voltaje es "Mucho"
5	Si velocidad es "Muy rápido positivo"	Entonces voltaje es "Demasiado"
6	Si velocidad es "Muy rápido negativo" y "Rápido negativo"	Entonces voltaje es "Nulo"
7	Si velocidad es "Rápido negativo" y "Muy rápido negativo"	Entonces voltaje es "Nulo"
8	Si velocidad es "Rápido negativo" y "Leve"	Entonces voltaje es "Nulo"
9	Si velocidad es "Leve" y "Rápido negativo"	Entonces voltaje es "Muy poco" y "Poco"

Regla	Antecedentes	Consecuentes
10	Si velocidad es “Leve” y “Rápido positivo”	Entonces voltaje es “Moderado” y “Mucho”
11	Si velocidad es “Rápido positivo” y “Leve”	Entonces voltaje es “Mucho” y “Moderado”
12	Si velocidad es “Rápido positivo” y “Muy rápido positivo”	Entonces voltaje es “Mucho” y “Demasiado”
13	Si velocidad es “Muy rápido positivo” y “Rápido positivo”	Entonces voltaje es “Demasiado” y “Mucho”

A continuación se muestran los universos del discurso de la variable de entrada y de la variable de salida.

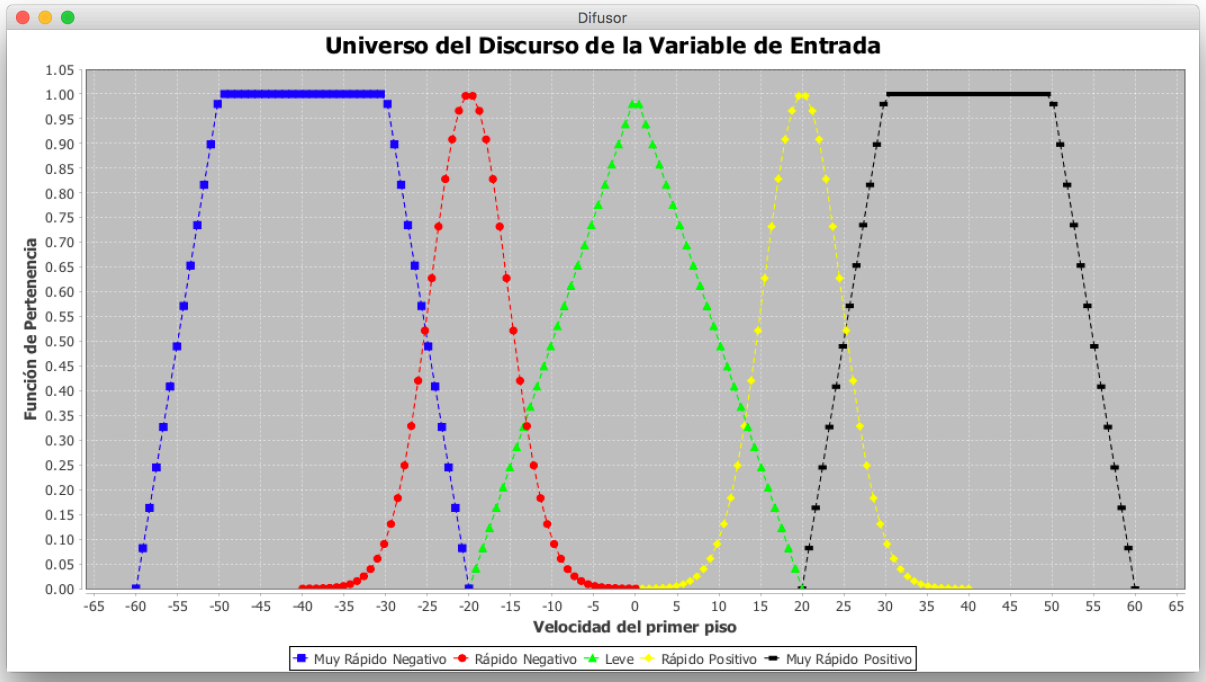

Figura 4.3 Universo del discurso de la variable de entrada “velocidad”

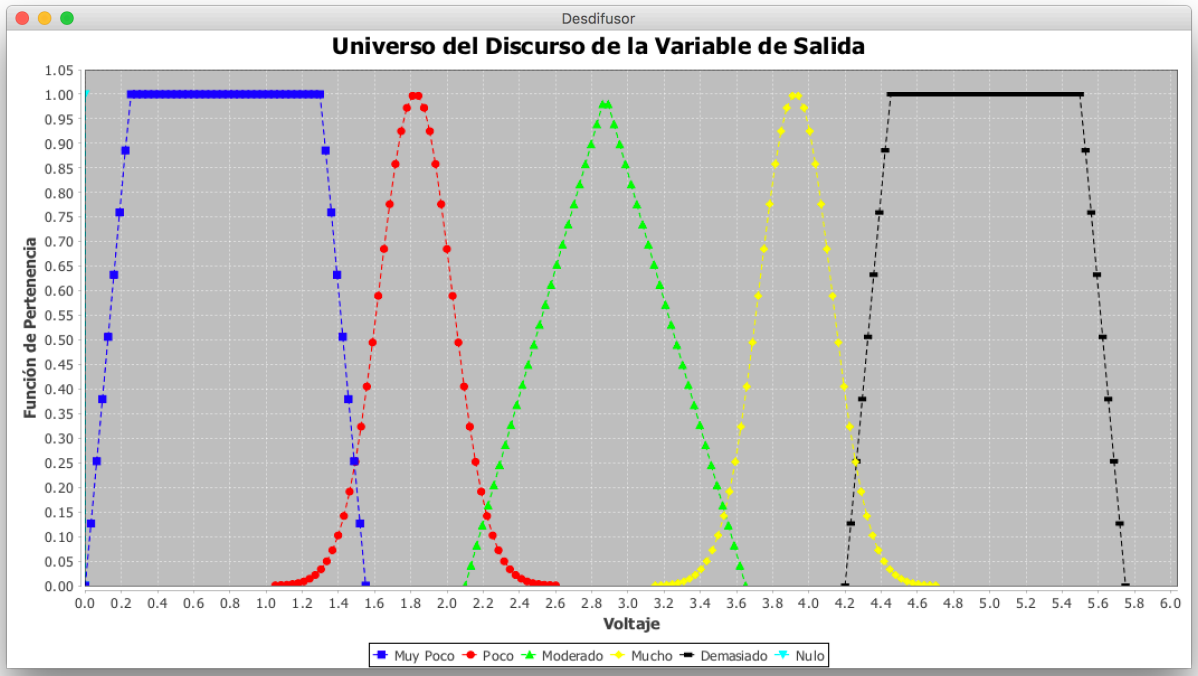

Figura 4.4 Universo del discurso de la variable de salida "voltaje"

El rango de la variable de entrada se ha definido considerando la respuesta libre del edificio con respecto a la velocidad [cm/s]. Este rango es de [-60.0, 60.0] [cm/s]. El rango de la variable de salida se ha definido con el rango de operación del voltaje [v] con el que trabaja el amortiguador. Este rango es de [0.0, 5.0] [v].

Las Figuras 4.5 y 4.6 muestran la respuesta con respecto a la velocidad y al desplazamiento del edificio con la intervención del controlador difuso tipo A.

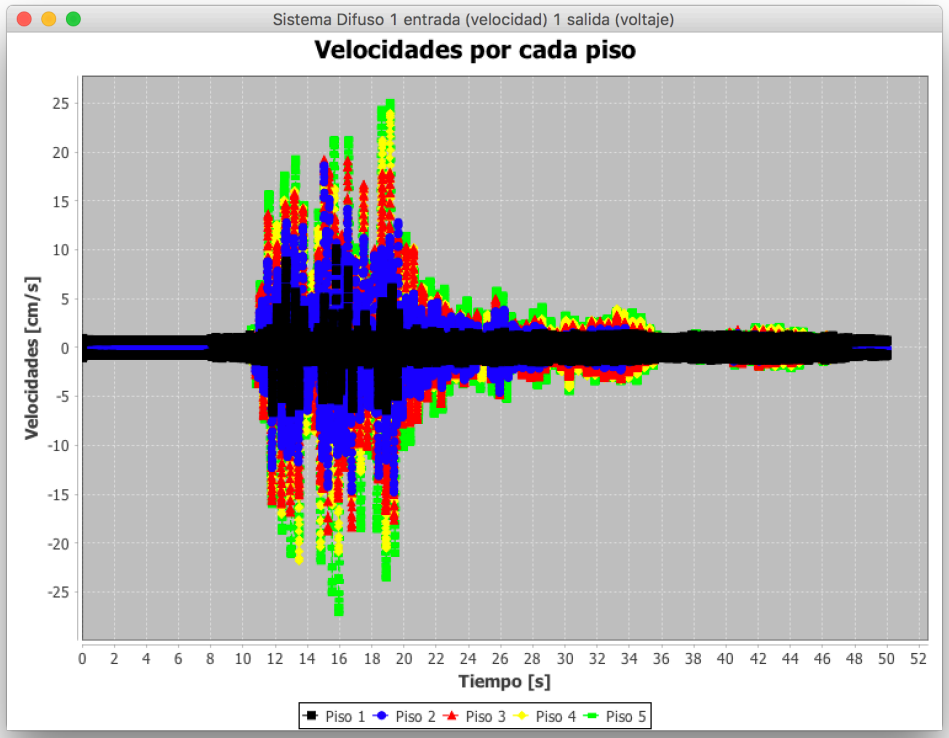

Figura 4.5 Respuesta con respecto a la velocidad del edificio usando el controlador difuso tipo A

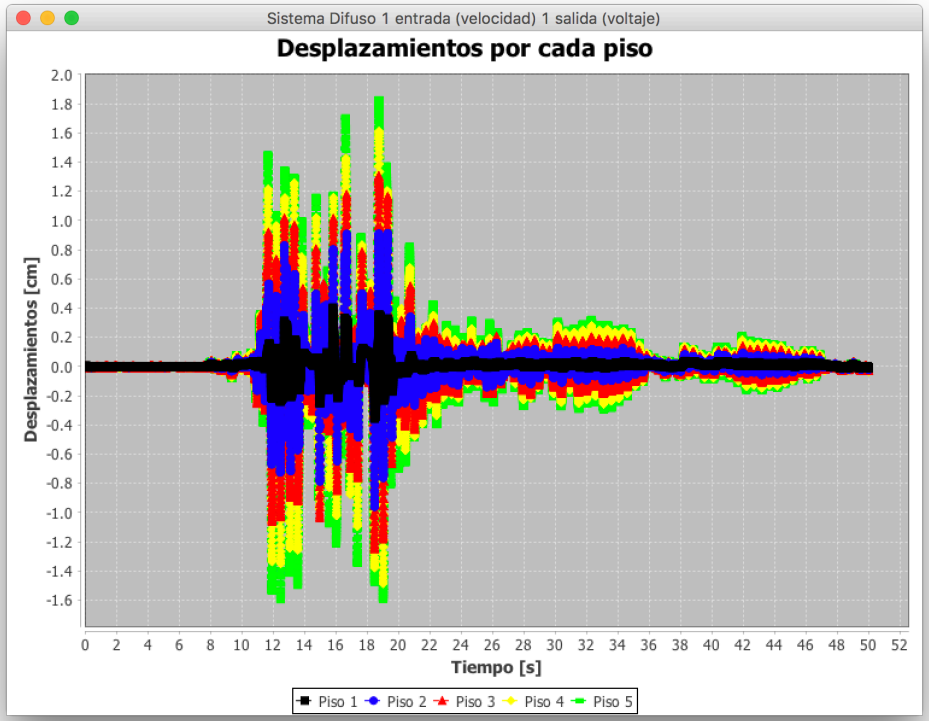

Figura 4.6 Respuesta con respecto al desplazamiento del edificio usando el controlador difuso tipo A

El valor más alto registrado de velocidad positiva es 25.186 [cm/s] y el valor más bajo que se registró de velocidad negativa es -27.206 [cm/s]. En cuanto a los desplazamientos, el valor positivo más alto registrado es 1.834 [cm] y el valor negativo más bajo es -1.606 [cm].

Usando la expresión para obtener el valor máximo de respuesta, se calculan los valores máximos de respuesta con respecto a la velocidad y al desplazamiento del edificio con la intervención del controlador difuso tipo A:

$$ValorMáximoDeRespuesta_{Velocidad} = Max(25.186, |-27.206|)$$
$$ValorMáximoDeRespuesta_{Velocidad} = 27.206 \text{ [cm/s]}$$

$$ValorMáximoDeRespuesta_{Desplazamiento} = Max(1.834, |-1.606|)$$
$$ValorMáximoDeRespuesta_{Desplazamiento} = 1.834 \text{ [cm]}$$

Controlador difuso tipo B

Este controlador difuso tiene una entrada que es el desplazamiento del primer piso del edificio y como salida el voltaje que hay que aplicar al amortiguador.

Al igual que el controlador difuso tipo A, este controlador tiene una entrada y una salida por lo que la base de conocimiento que lo gobierna es muy similar a la del controlador descrito anteriormente. A continuación se muestra la base de conocimiento que se usó para este controlador difuso.

Tabla 4.2 Base de conocimiento para el controlador difuso tipo B

Regla	Antecedentes	Consecuentes
1	Si desplazamiento es "Mucho movimiento negativo"	Entonces voltaje es "Nulo"
2	Si desplazamiento es "Poco movimiento negativo"	Entonces voltaje es "Nulo"
3	Si desplazamiento es "Estático"	Entonces voltaje es "Moderado"

Regla	Antecedentes	Consecuentes
4	Si desplazamiento es “Poco movimiento positivo”	Entonces voltaje es “Mucho”
5	Si desplazamiento es “Mucho movimiento positivo”	Entonces voltaje es “Demasiado”
6	Si desplazamiento es “Mucho movimiento negativo” y “Poco movimiento negativo”	Entonces voltaje es “Nulo”
7	Si desplazamiento es “Poco movimiento negativo” y “Mucho movimiento negativo”	Entonces voltaje es “Nulo”
8	Si desplazamiento es “Poco movimiento negativo” y “Estático”	Entonces voltaje es “Nulo”
9	Si desplazamiento es “Estático” y “Poco movimiento negativo”	Entonces voltaje es “Muy poco” y “Poco”
10	Si desplazamiento es “Estático” y “Poco movimiento positivo”	Entonces voltaje es “Moderado” y “Mucho”
11	Si desplazamiento es “Poco movimiento positivo” y “Estático”	Entonces voltaje es “Mucho” y “Moderado”
12	Si desplazamiento es “Poco movimiento positivo” y “Mucho movimiento positivo”	Entonces voltaje es “Mucho” y “Demasiado”
13	Si desplazamiento es “Mucho movimiento positivo” y “Poco movimiento positivo”	Entonces voltaje es “Demasiado” y “Mucho”

A continuación se muestran los universos del discurso de la variable de entrada y de la variable de salida.

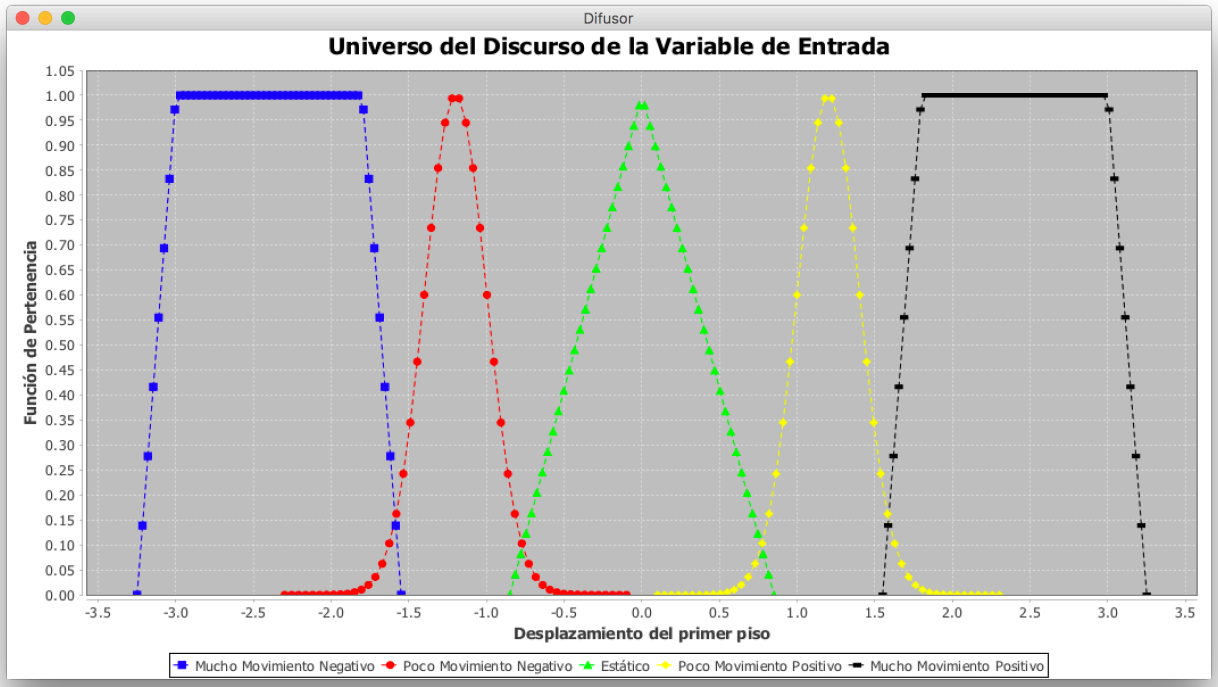

Figura 4.7 Universo del discurso de la variable de entrada "desplazamiento"

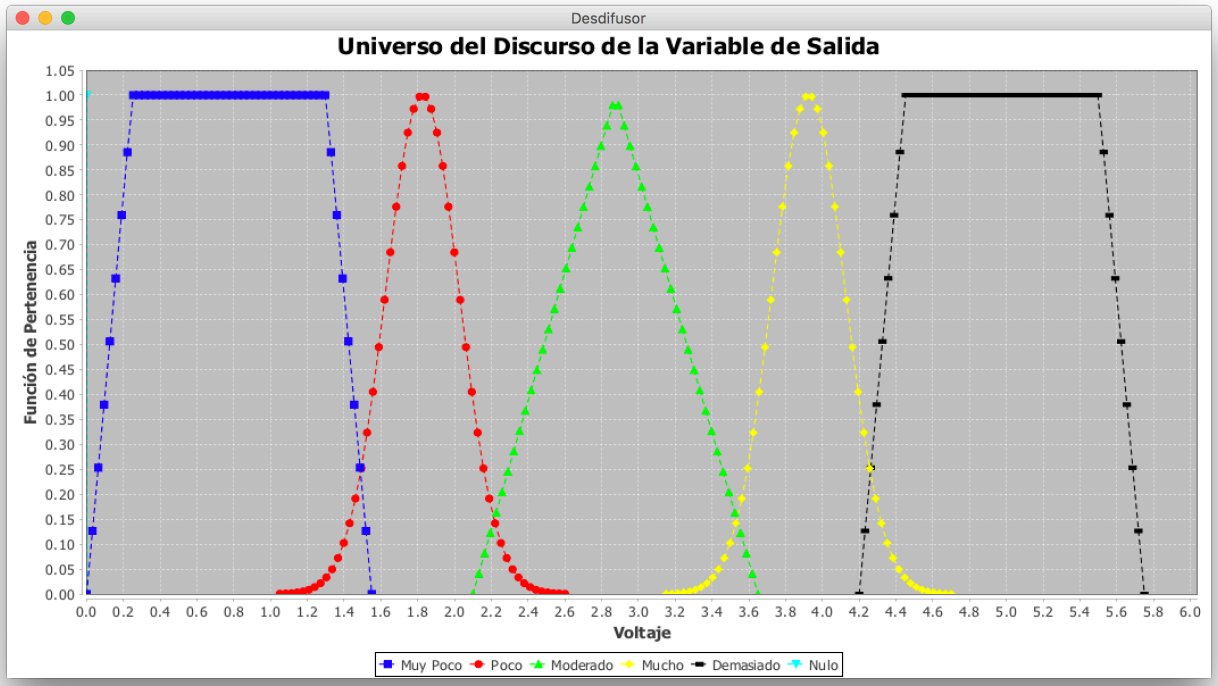

Figura 4.8 Universo del discurso de la variable de salida "voltaje"

El rango de la variable de entrada se ha definido considerando la respuesta libre del edificio con respecto al desplazamiento [cm]. Este rango es de [-3.25, 3.25] [cm]. El rango de la variable de salida se ha definido con el rango de operación del voltaje [v] del amortiguador. Este rango es de [0.0, 5.0] [v].

Las Figuras 4.9 y 4.10 muestran la respuesta con respecto a la velocidad y al desplazamiento del edificio con la intervención del controlador difuso tipo B.

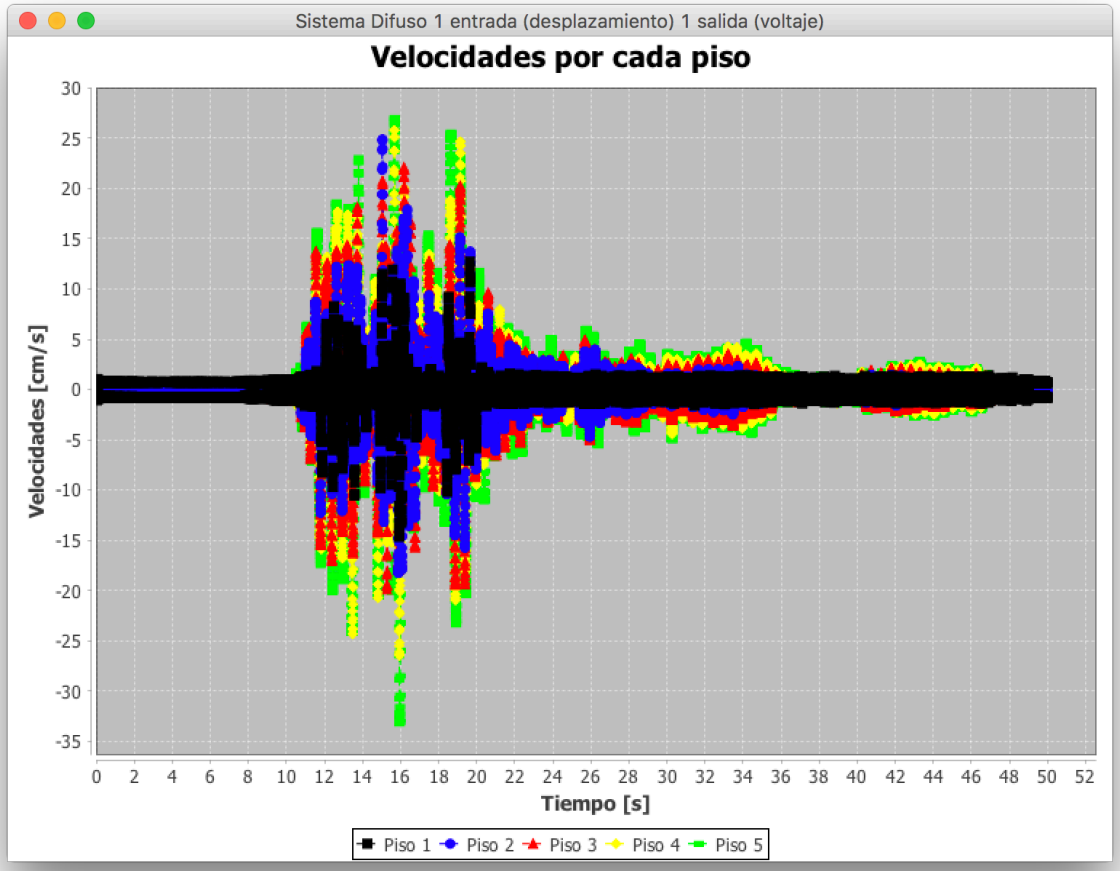

Figura 4.9 Respuesta con respecto a la velocidad del edificio usando el controlador difuso tipo B

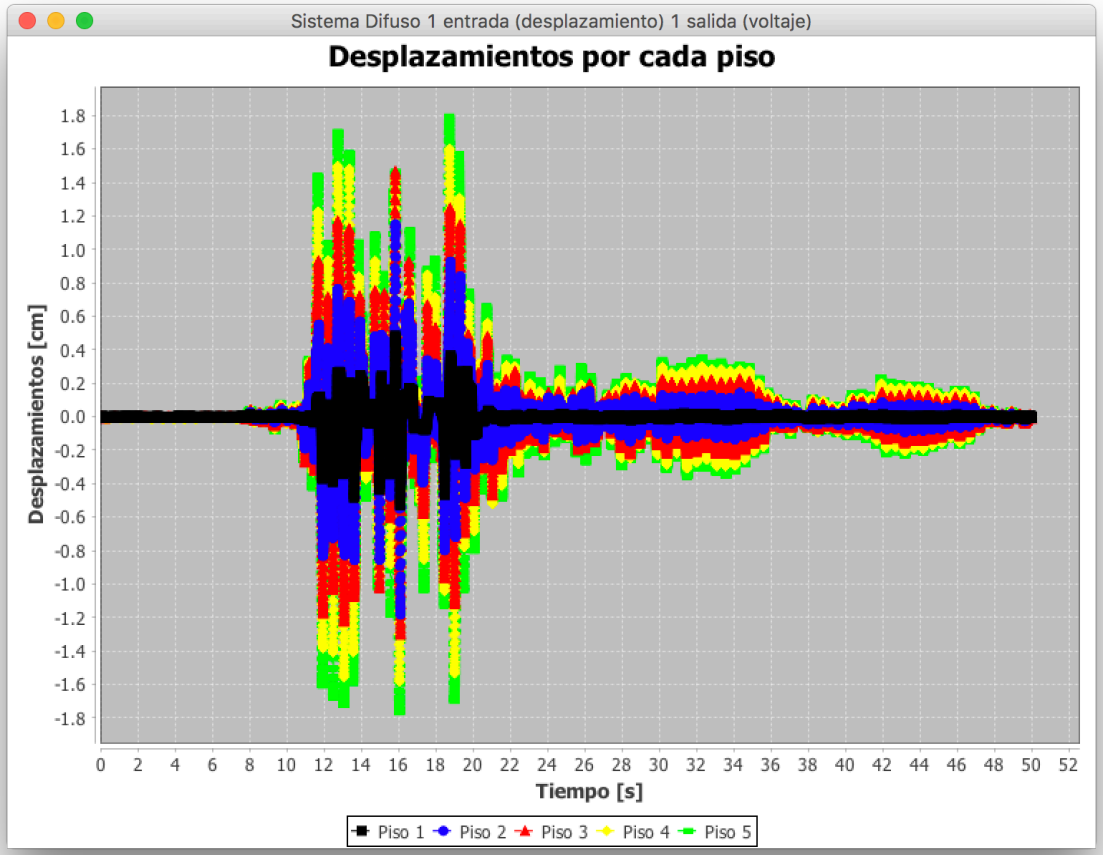

Figura 4.10 Respuesta con respecto al desplazamiento del edificio usando el controlador difuso tipo B

El valor más alto registrado de velocidad positiva es 27.028 [cm/s] y el valor más bajo que se registró de velocidad negativa es -33.227 [cm/s]. En cuanto a los desplazamientos, el valor positivo más alto registrado es 1.795[cm] y el valor negativo más bajo es -1.768 [cm]. Usando la expresión para obtener el valor máximo de respuesta, se calculan los valores máximos de respuesta con respecto a la velocidad y al desplazamiento del edificio con la intervención del controlador difuso tipo B:

$$ValorMáximoDeRespuesta_{Velocidad} = Max(27.028, |-33.227|)$$
$$ValorMáximoDeRespuesta_{Velocidad} = 33.227 \text{ [cm/s]}$$

$$ValorMáximoDeRespuesta_{Desplazamiento} = Max(1.795, |-1.768|)$$
$$ValorMáximoDeRespuesta_{Desplazamiento} = 1.795 \text{ [cm]}$$

Controlador difuso tipo C

Este controlador difuso tiene dos entradas: la velocidad y el desplazamiento del primer piso. Como salida, al igual que los dos anteriores controladores, es el voltaje que debe aplicarse al amortiguador para que éste opere. La base de conocimiento que gobierna a este controlador se muestra a continuación.

Tabla 4.3 Base de conocimiento para el controlador difuso tipo C

Regla	Antecedentes	Consecuentes
1	Si velocidad es “Muy rápido negativo” y desplazamiento es “Mucho movimiento negativo”	Entonces voltaje es “Nulo”
2	Si velocidad es “Muy rápido negativo” y desplazamiento es “Poco movimiento negativo”	Entonces voltaje es “Nulo”
3	Si velocidad es “Muy rápido negativo” y desplazamiento es “Estático”	Entonces voltaje es “Nulo”
4	Si velocidad es “Muy rápido negativo” y desplazamiento es “Poco movimiento positivo”	Entonces voltaje es “Muy poco”
5	Si velocidad es “Muy rápido negativo” y desplazamiento es “Mucho movimiento positivo”	Entonces voltaje es “Muy poco” y “Poco”
6	Si velocidad es “Rápido negativo” y desplazamiento es “Mucho movimiento negativo”	Entonces voltaje es “Nulo”
7	Si velocidad es “Rápido negativo” y desplazamiento es “Poco movimiento negativo”	Entonces voltaje es “Nulo”
8	Si velocidad es “Rápido negativo” y desplazamiento es “Estático”	Entonces voltaje es “Nulo”
9	Si velocidad es “Rápido negativo” y desplazamiento es “Poco movimiento positivo”	Entonces voltaje es “Poco”
10	Si velocidad es “Rápido negativo” y desplazamiento es “Mucho movimiento positivo”	Entonces voltaje es “Poco” y “Muy poco”
11	Si velocidad es “Leve” y desplazamiento es “Mucho movimiento negativo”	Entonces voltaje es “Muy poco” y “Poco”
12	Si velocidad es “Leve” y desplazamiento es “Poco movimiento negativo”	Entonces voltaje es “Poco” y “Muy poco”

Regla	Antecedentes	Consecuentes
13	Si velocidad es “Leve” y desplazamiento es “Estático”	Entonces voltaje es “Moderado”
14	Si velocidad es “Leve” y desplazamiento es “Poco movimiento positivo”	Entonces voltaje es “Moderado” y “Poco”
15	Si velocidad es “Leve” y desplazamiento es “Mucho movimiento positivo”	Entonces voltaje es “Moderado” y “Mucho”
16	Si velocidad es “Rápido positivo” y desplazamiento es “Mucho movimiento negativo”	Entonces voltaje es “Poco” y “Muy poco”
17	Si velocidad es “Rápido positivo” y desplazamiento es “Poco movimiento negativo”	Entonces voltaje es “Poco”
18	Si velocidad es “Rápido positivo” y desplazamiento es “Estático”	Entonces voltaje es “Poco” y “Moderado”
19	Si velocidad es “Rápido positivo” y desplazamiento es “Poco movimiento positivo”	Entonces voltaje es “Mucho”
20	Si velocidad es “Rápido positivo” y desplazamiento es “Mucho movimiento positivo”	Entonces voltaje es “Mucho” y “Moderado”
21	Si velocidad es “Muy Rápido positivo” y desplazamiento es “Mucho movimiento negativo”	Entonces voltaje es “Moderado”
22	Si velocidad es “Muy Rápido positivo” y desplazamiento es “Poco movimiento negativo”	Entonces voltaje es “Moderado” y “Mucho”
23	Si velocidad es “Muy Rápido positivo” y desplazamiento es “Leve”	Entonces voltaje es “Demasiado”
24	Si velocidad es “Muy Rápido positivo” y desplazamiento es “Poco movimiento positivo”	Entonces voltaje es “Mucho” y “Demasiado”
25	Si velocidad es “Muy Rápido positivo” y desplazamiento es “Mucho movimiento positivo”	Entonces voltaje es “Demasiado” y “Mucho”

A continuación se muestran los universos del discurso que están presentes en este controlador difuso.

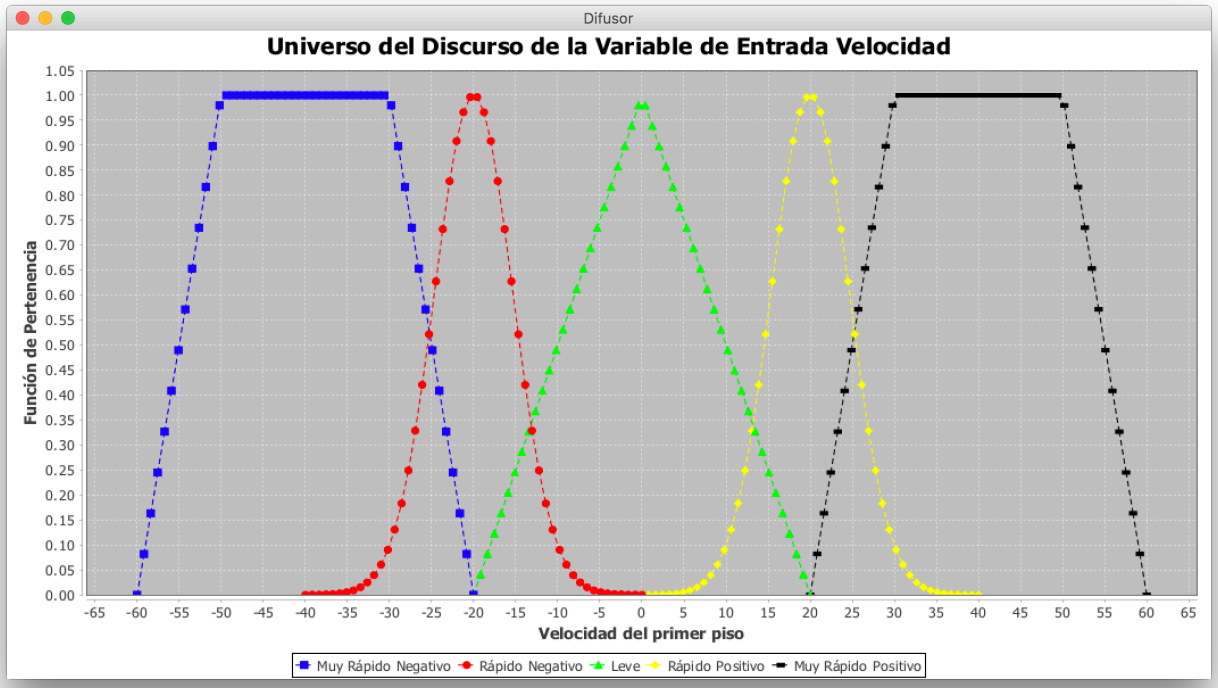

Figura 4.11 Universo del discurso de la variable de entrada "velocidad"

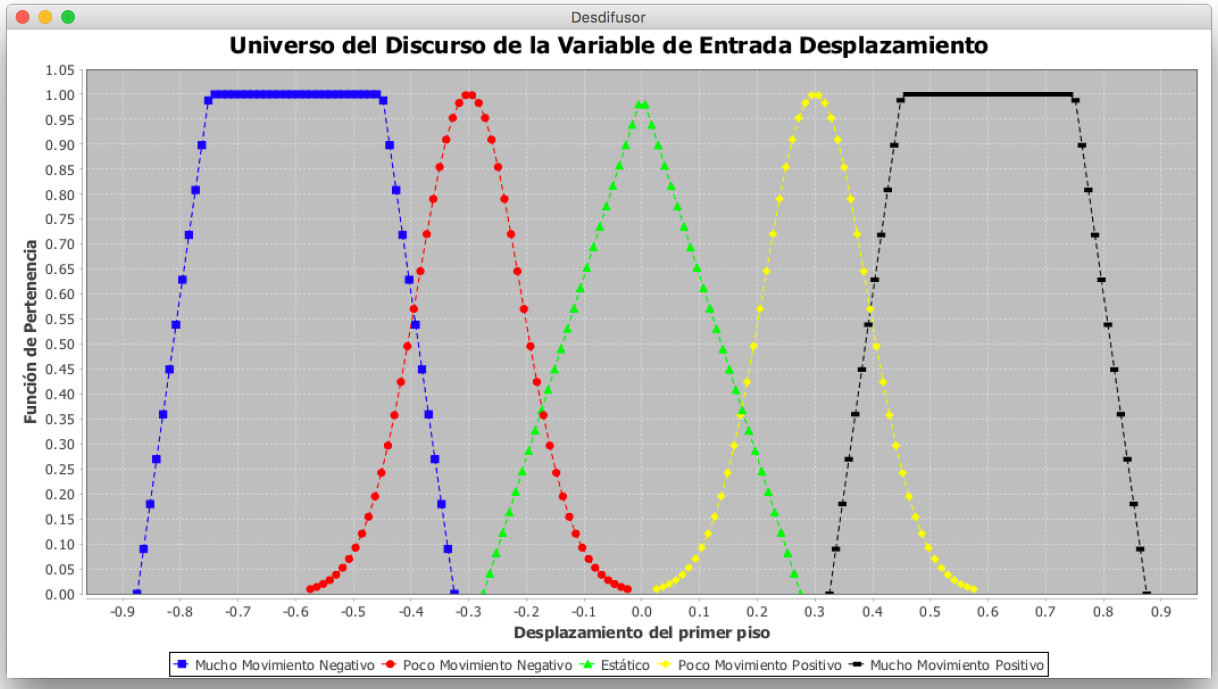

Figura 4.12 Universo del discurso de la variable de entrada "desplazamiento"

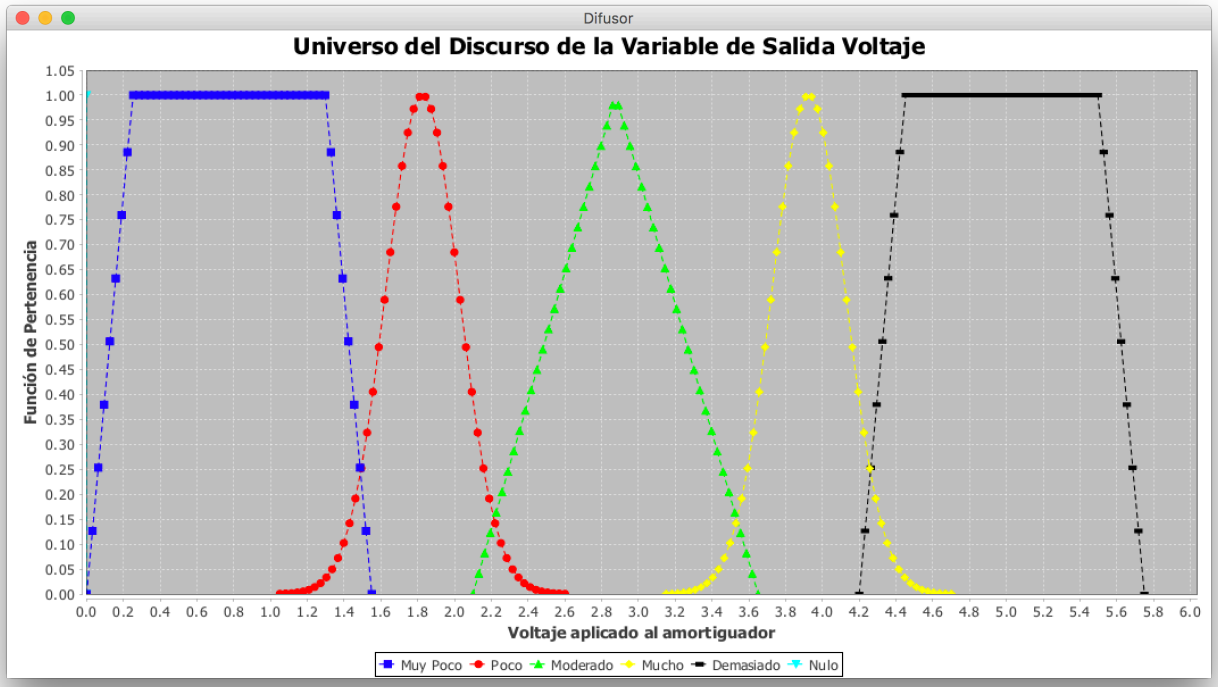

Figura 4.13 Universo del discurso de la variable de salida "voltaje"

El rango de la variable de entrada "velocidad" es el mismo que se usó en los controladores difusos anteriores: [-60.0, 60.0] [cm/s].

Considerando que existe una alta epístasis entre las variables de entrada debido a que la velocidad del edificio es directamente proporcional al desplazamiento del mismo, provocado por la excitación externa y considerando también que la naturaleza oscilatoria de ésta, provoca que en determinados momentos el sentido y magnitud de las variables de entrada se contrapongan (Muy Rápido Positivo y Mucho Movimiento Negativo, por ejemplo), el rango de la variable de entrada "desplazamiento" se reduce considerablemente a [-0.875, 0.875] [cm].

A continuación se muestran las respuestas con respecto a la velocidad y al desplazamiento del edificio con la intervención del controlador difuso tipo C.

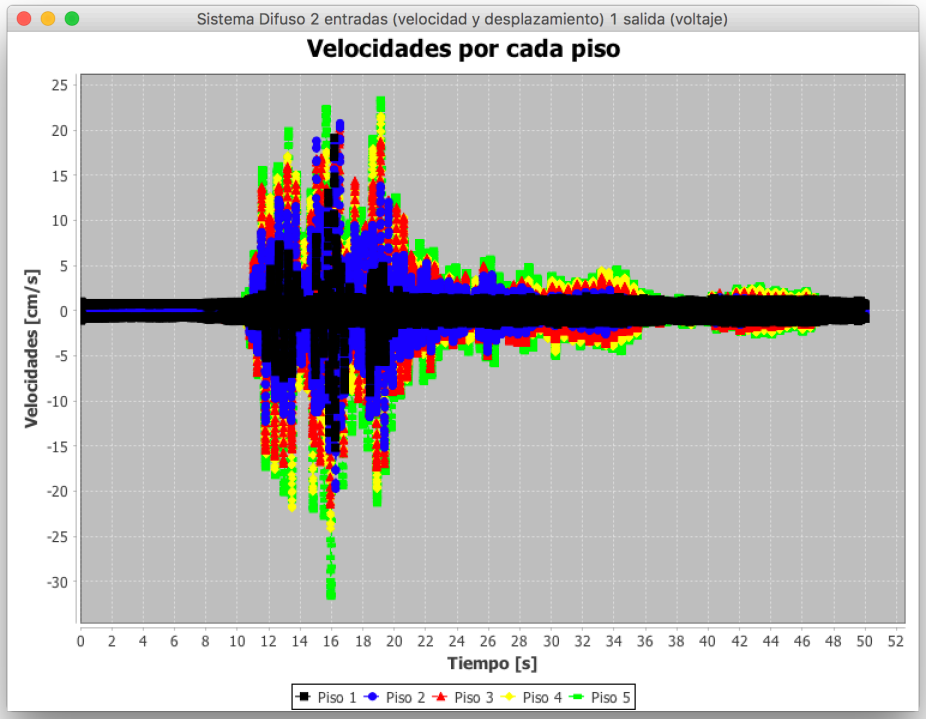

Figura 4.14 Respuesta con respecto a la velocidad del edificio usando el controlador difuso tipo C

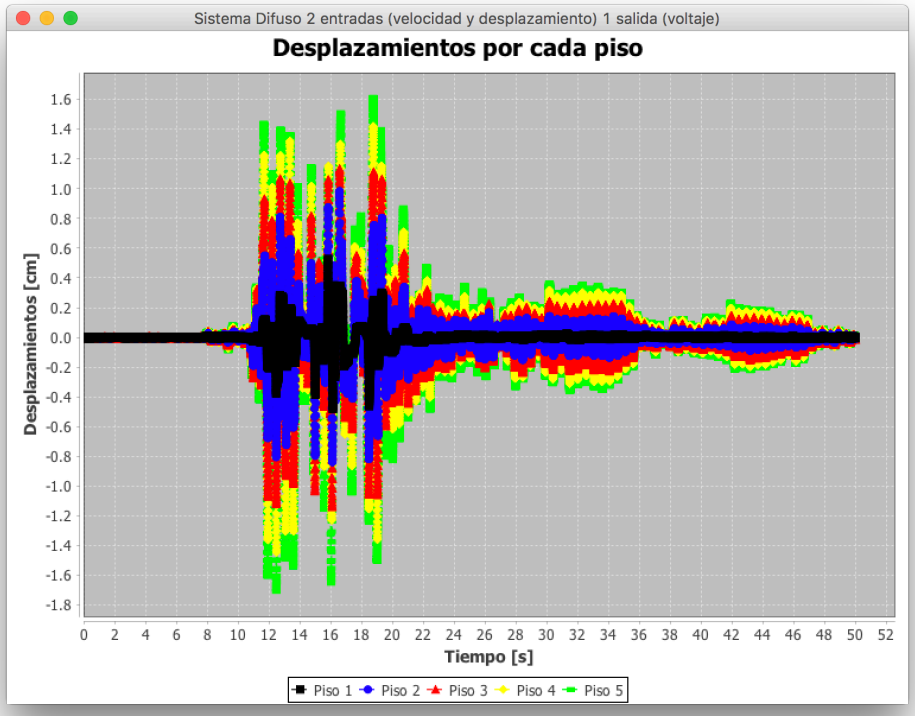

Figura 4.15 Respuesta con respecto al desplazamiento del edificio usando el controlador difuso tipo C

El valor más alto registrado de velocidad positiva es 24.333 [cm/s] y el valor más bajo que se registró de velocidad negativa es -30.540 [cm/s]. En cuanto a los desplazamientos, el valor positivo más alto registrado es 1.647 [cm] y el valor negativo más bajo es -1.709 [cm]. Usando la expresión para obtener el valor máximo de respuesta, se pueden calcular los valores máximos de respuesta con respecto a la velocidad y al desplazamiento del edificio con la intervención del controlador difuso tipo C:

$$ValorMáximoDeRespuesta_{Velocidad} = Max(24.333, |-30.540|)$$
$$ValorMáximoDeRespuesta_{Velocidad} = 30.540 \text{ [cm/s]}$$

$$ValorMáximoDeRespuesta_{Desplazamiento} = Max(1.647, |-1.709|)$$
$$ValorMáximoDeRespuesta_{Desplazamiento} = 1.709 \text{ [cm]}$$

Análisis de resultados

Después de llevar a cabo las simulaciones numéricas necesarias con cada controlador difuso, se debe elegir cuál es el más eficaz. La Tabla 4.4 muestra los resultados que arrojaron las simulaciones antes mencionadas.

Tabla 4.4 Resultados de las simulaciones con los controladores difusos

Respuesta	Valor máximo de respuesta de velocidad [cm/s]	Reducción en velocidad	Valor máximo de respuesta de desplazamiento [cm]	Reducción en desplazamiento
Libre del edificio	56.178	0.0%	3.619	0.0%
Edificio con controlador difuso tipo A	27.206	51.57%	1.834	49.32%
Edificio con controlador difuso tipo B	33.227	40.85%	1.795	50.40%

Respuesta	Valor máximo de respuesta de velocidad [cm/s]	Reducción en velocidad	Valor máximo de respuesta de desplazamiento [cm]	Reducción en desplazamiento
Edificio con controlador difuso tipo C	30.540	45.63%	1.709	52.77%

Como puede apreciarse en la tabla anterior, en cuanto a reducción de velocidad, el controlador difuso tipo A ha resultado ser el más eficaz. La mayor eficacia en la reducción del desplazamiento la tiene el controlador tipo C.

Los resultados obtenidos no exploran el problema del máximo desplazamiento que un piso del edificio puede soportar, que depende de la geometría y materiales de construcción. El enfoque seguido es que, independiente de este máximo, es conveniente reducirlo lo más posible. La experiencia práctica en Ingeniería Civil indica que reducciones del orden del 50% en los desplazamientos máximos son normalmente suficientes para prevenir daño permanente y evitar reparaciones futuras, siempre y cuando el edificio haya sido diseñado originalmente de acuerdo a las normas de construcción pertinentes.

Con base en lo anterior, el controlador difuso tipo C resulta ser el seleccionado debido a que tiene una reducción de más del 50% en el desplazamiento del edificio.

Tiempo de ejecución

La finalidad de llevar a cabo simulaciones numéricas es con el propósito de conocer la eficacia del controlador pero también, y no menos importante, qué tan viable resulta su implementación en modelos reales. Considerando que la mayor eficacia la tuvo el controlador difuso tipo C, se llevaron a cabo pruebas de tiempo de ejecución de los procesos internos de este controlador.

Estas pruebas consisten en medir el tiempo desde que ingresan los valores de velocidad y desplazamiento al sistema difuso hasta que sale el valor de voltaje para el amortiguador. A continuación se muestra una tabla con los promedios de tiempo de ejecución al término de una simulación completa.

Tabla 4.5 Promedios en los tiempos de ejecución de una simulación completa

Número de Ejecución	Promedio en el tiempo de ejecución [ms]
1	0.004872
2	0.005319
3	0.004661
4	0.004729
5	0.005515
6	0.005516
7	0.004676
8	0.005169
9	0.004567
10	0.00483

A partir de esta tabla, se puede calcular que en promedio, el tiempo de ejecución desde que entra el par de datos del edificio hasta que sale un valor de voltaje es de 0.0049854 [ms], por lo que es posible llevar este sistema difuso a un modelo físico, es decir, es posible su implementación en tiempo real.

Es fundamental añadir que las simulaciones fueron realizadas bajo el sistema operativo Mac Os versión 10.11.6, con un procesador Intel Core i5 Cuad Core a 3.5 GHz, con 16 GB en memoria RAM a una velocidad de 1600 MHz. El lenguaje de programación que se usó para implementar el sistema difuso es Java en su versión 8 actualización 91 usando Netbeans versión 8.0.2.

Optimización de la base de conocimiento

Considerando que la base de conocimiento del controlador difuso tipo C fue construida basándose en el conocimiento de un experto, existe la posibilidad de optimizarla. Para llevar a cabo dicha optimización es posible emplear Algoritmos Genéticos.

El objetivo de esta optimización es encontrar un conjunto adecuado de reglas difusas que gobiernen el controlador difuso para resolver el problema de forma adecuada. Existen dos formas de representar un cromosoma (o una posible solución) para llevar a cabo el conjunto de pasos que constituye un algoritmo genético (Ishibuchi, Nakashima y Mirata 1997):

• *Enfoque Pittsburgh*.- Un cromosoma representa la base de reglas completa (ver Figura 4.16).

• *Enfoque Michigan*.- Un cromosoma representa una regla y la población completa es la base de reglas.

Se ha seleccionado el enfoque Pittsburgh debido a que la base de reglas del controlador difuso no es tan grande, por lo tanto, un cromosoma se puede representar de la siguiente forma:

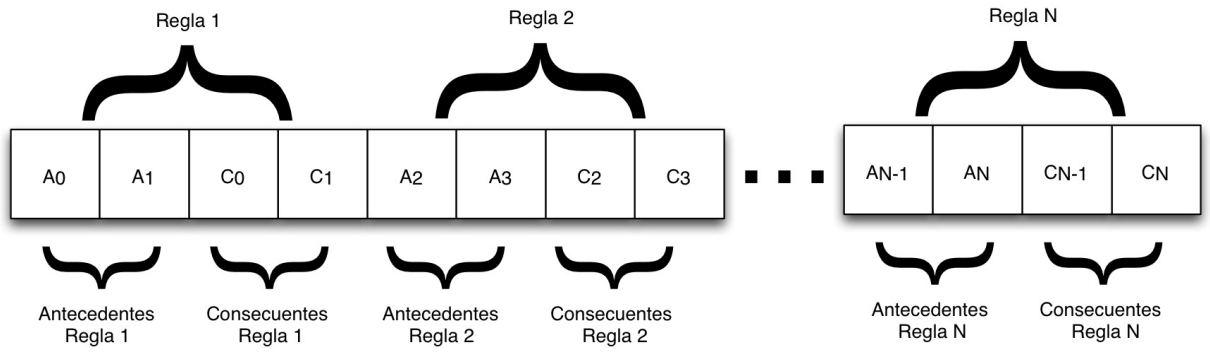

Tabla 4.16 Representación de un cromosoma desde el enfoque Pittsburgh

Función objetivo o "fitness"

Recordando que el objetivo del sistema difuso es evitar el movimiento en el edificio cuando éste se ve afectado por la aceleración del suelo, la función objetivo consiste en restar la suma de los valores de los consecuentes a la suma de los valores de los antecedentes, es decir,

$$Aptitud = (ValorAnt_1 + ValorAnt_2) - (ValorCon_1 + ValorCon_2)$$

Esto se aplica a cada regla que conforma la base de conocimiento, es decir, al cromosoma. Mientras más cercano sea el valor resultante a 0, la aptitud del cromosoma será mayor. Esto puede interpretarse de tal forma que si el voltaje es capaz de atenuar o neutralizar la velocidad y desplazamiento del edificio, la regla tiene una alta aptitud. *ValorAnt*$_1$ y *ValorAnt*$_2$ son los valores asignados a cada conjunto difuso asociado a las variables de entrada del controlador difuso, es decir, a la velocidad y el desplazamiento del edificio. Sucede algo análogo con *ValorCon*$_1$ y *ValorCon*$_2$ ya que son los valores asignados a cada conjunto difuso de la variable de salida del controlador difuso, o sea, al voltaje que gobierna al amortiguador. La Tabla 4.6 muestra el valor asignado a cada conjunto difuso asociado a las variables de entrada y salida del controlador difuso.

Tabla 4.6 Valores asignados a los conjuntos difusos asociados a las variables del controlador difuso

Tipo de variable	Nombre de la variable	Conjunto difuso (etiqueta lingüística)	Valor asignado
Entrada	Velocidad	Muy rápido negativo	1
Entrada	Velocidad	Rápido negativo	2
Entrada	Velocidad	Leve	3
Entrada	Velocidad	Rápido positivo	4
Entrada	Velocidad	Muy rápido positivo	5
Entrada	Desplazamiento	Mucho movimiento negativo	1
Entrada	Desplazamiento	Poco movimiento negativo	2
Entrada	Desplazamiento	Estático	3
Entrada	Desplazamiento	Poco movimiento positivo	4
Entrada	Desplazamiento	Mucho movimiento positivo	5
Salida	Voltaje	Nulo	0
Salida	Voltaje	Muy Poco	1
Salida	Voltaje	Poco	2
Salida	Voltaje	Moderado	3

Tipo de variable	Nombre de la variable	Conjunto difuso (etiqueta lingüística)	Valor asignado
Salida	Voltaje	Mucho	4
Salida	Voltaje	Demasiado	6

Los parámetros usados en el algoritmo genético se definieron principalmente para tener una buena diversidad en la población ya que el espacio de búsqueda es del orden 5^{50} para los antecedentes y 6^{50} para los consecuentes de las reglas que conforman la base de conocimiento. Estos parámetros son los siguientes:

Tabla 4.7 Parámetros empleados en el Algoritmo Genético para optimizar la base de conocimiento

Codificación	Individuos por generación	Número de generaciones	Método de selección	Probabilidad de cruza	Probabilidad de muta
Entera	500	400	Torneo	0.90	0.10

La optimización arroja una base de conocimiento de 25 reglas (el mismo número de reglas de la base original), sin embargo, algunas reglas no tuvieron cambio alguno dado que la suma de sus valores es muy cercana a 0. En ciertos casos, redujo la cantidad de consecuentes de algunas reglas y en otros casos modificó los consecuentes. La siguiente tabla muestra las reglas que fueron optimizadas.

Tabla 4.8 Conjunto de reglas optimizadas del controlador difuso tipo C

Número de regla	Regla sin optimizar	Regla optimizada
5	Si velocidad es “Muy rápido negativo” y desplazamiento es “Mucho movimiento positivo” entonces voltaje es “Muy poco” y “Poco”	Si velocidad es “Muy rápido negativo” y desplazamiento es “Mucho movimiento positivo” entonces voltaje es “Mucho”
8	Si velocidad es “Rápido negativo” y desplazamiento es “Estático” entonces voltaje es “Nulo”	Si velocidad es “Rápido negativo” y desplazamiento es “Estático” entonces voltaje es “Poco” y “Muy poco”
9	Si velocidad es “Rápido negativo” y desplazamiento es “Poco movimiento positivo” entonces voltaje es “Poco”	Si velocidad es “Rápido negativo” y desplazamiento es “Poco movimiento positivo” entonces voltaje es “Muy poco”

Número de regla	Regla sin optimizar	Regla optimizada
15	Si velocidad es “Leve” y desplazamiento es “Mucho movimiento positivo” entonces voltaje es “Moderado” y “Mucho”	Si velocidad es “Leve” y desplazamiento es “Mucho movimiento positivo” entonces voltaje es “Mucho”
17	Si velocidad es “Rápido positivo” y desplazamiento es “Poco movimiento negativo” entonces voltaje es “Poco”	Si velocidad es “Rápido positivo” y desplazamiento es “Poco movimiento negativo” entonces voltaje es “Demasiado”
20	Si velocidad es “Rápido positivo” y desplazamiento es “Mucho movimiento positivo” entonces voltaje es “Mucho” y “Moderado”	Si velocidad es “Rápido positivo” y desplazamiento es “Mucho movimiento positivo” entonces voltaje es “Demasiado”
22	Si velocidad es “Muy rápido positivo” y desplazamiento es “Poco movimiento negativo” entonces voltaje es “Mucho” y “Moderado”	Si velocidad es “Muy rápido positivo” y desplazamiento es “Poco movimiento negativo” entonces voltaje es “Demasiado”

Es importante añadir que la base de conocimiento optimizada resulta ser una solución aceptable debido a que la naturaleza del problema no permite definir una base de conocimiento que haga las veces de un mínimo, es decir, no hay una referencia conocida y aceptada (o benchmark) para determinar si la base de conocimiento que arroja el Algoritmo Genético tiene un desempeño mejor o peor al mínimo.

Por otro lado, esta optimización trae como consecuencia un aumento en la reducción del movimiento del edificio. La tabla 4.9 muestra dicho aumento.

Tabla 4.9 Comparativa entre los resultados del controlador difuso tipo C optimizado y sin optimizar

Respuesta	Valor máximo de respuesta de velocidad [cm/s]	Reducción en velocidad	Valor máximo de respuesta de desplazamiento [cm]	Reducción en desplazamiento
Libre del edificio	56.178	0.0%	3.619	0.0%
Edificio con controlador difuso tipo C	30.540	45.63%	1.709	52.77%

Respuesta	Valor máximo de respuesta de velocidad [cm/s]	Reducción en velocidad	Valor máximo de respuesta de desplazamiento [cm]	Reducción en desplazamiento
Edificio con controlador difuso tipo C optimizado	31.774	43.44%	1.614	55.40%

A pesar de que la reducción en la velocidad del edificio disminuyó, hubo un leve aumento en la reducción del desplazamiento. Por otro lado, el tiempo de ejecución también se vio reducido.

La Tabla 4.10 muestra el tiempo de ejecución promedio para un conjunto de simulaciones con la base de conocimiento optimizada.

Tabla 4.10 Promedios en los tiempos de ejecución de una simulación completa con la base de conocimiento optimizada

Número de Ejecución	Promedio en el tiempo de ejecución [ms]
1	0.004213
2	0.004319
3	0.004313
4	0.004197
5	0.004208
6	0.00451
7	0.004325
8	0.004553
9	0.004272
10	0.004161

Con la base de conocimiento optimizada, el tiempo promedio de ejecución es de 0.004307. Esto implica que la ejecución es en promedio un 13.60% más rápida que la ejecución sin haber optimizado la base de conocimiento.

Capítulo 5
Conclusiones.

El empleo de sistemas difusos hace posible resolver problemas complejos de una forma sencilla, pero robusta y eficaz. Como hasta ahora se ha visto, con un sistema difuso es posible atenuar las vibraciones en un edificio provocadas por un sismo en más de un 50%, empleando un dispositivo que presenta el fenómeno de histéresis debido a la naturaleza de los materiales con los que está construido y cuyo comportamiento es altamente no lineal.

Uno de las ventajas de esta técnica es que no requiere el conocimiento del modelo matemático del sistema físico, que es necesario al utilizar técnicas de control clásico y avanzado. La implementación con esta técnica de inteligencia artificial resultó relativamente sencillas y los resultados fueron más que aceptables.

Desde la perspectiva de software, esta implementación ha sido diseñada con la finalidad de no sólo usarse con este problema puntualmente, es decir, el diseño es modular por lo que fácilmente se puede modificar para usarse con otra planta y otro actuador. Así mismo, la base de conocimiento que rige todo el funcionamiento es de fácil modificación. Es necesario añadir que esta implementación es lo suficientemente rápida para poder usarse como un sistema en tiempo real. Considerando lo anterior, esta implementación puede ser una buena plataforma de investigación para el control basado en lógica difusa.

El presente trabajo genera también un vínculo más sólido entre la teoría de control y la inteligencia artificial ya que actualmente existe poco trabajo referente a la solución de este tipo de problemas. Al seguir explotando esta área, los resultados pueden llegar a ser muy satisfactorios debido a que los sistemas difusos simplifican considerablemente el control y los algoritmos genéticos optimizan la forma de controlar del sistema difuso.

Trabajo futuro

Hasta este punto, el sistema difuso atenúa de forma aceptable las vibraciones producidas por un sismo a través de un solo amortiguador magnetoreológico, sin embargo, existe la posibilidad de realizar simulaciones considerando más amortiguadores, ya sea uno más en la base del edificio o uno por cada piso del edificio. Al tener más amortiguadores en interacción, de forma natural se espera que la atenuación aumente de forma considerable. Se pueden llevar a cabo dichas simulaciones con diversas configuraciones y números de amortiguadores para encontrar la más óptima.

Por otra parte, este sistema está listo para llevar a cabo pruebas con modelos físicos, sin embargo, es necesario implementar la interfase con el exterior, es decir, implementar la comunicación entre el sistema difuso y los dispositivos que miden las aceleraciones y desplazamientos de cada piso del edificio.

La inteligencia artificial posee diversas técnicas para resolver problemas de clasificación, automatización, toma de decisiones, aprendizaje, etc. Tomando en cuenta lo anterior, a este sistema difuso puede adaptársele una red neuronal artificial para poder construir su propia base de conocimiento en función de cómo se comporte el fenómeno externo, es decir, el sistema sería capaz de definir su propio comportamiento para controlar y optimizarlo de forma automática.

Por último, pero no menos importante, es necesario desarrollar una interfase de usuario amigable e intuitiva ya que actualmente el sistema difuso es hasta hoy un aplicación de consola. Esto facilitará la manipulación de archivos que son necesarios para las simulaciones y presentar al usuario de forma concisa las gráficas que muestran el comportamiento del edificio.

Referencias y Apéndices

Bibliografía

• Bonifacio, Martín del Brio y Alfredo Sanz. 1997. *Redes neuronales y sistemas borrosos*. Madrid, España: RA-MA Editorial.

• Wang, Li-Xin. 1997. *A Course in fuzzy systems and Control*. United States of America: Prentice Hall PTR.

• R. Alavala, Chennakesava. 2000. *Fuzzy Logic and neural networks. Basic concepts & applications*. United States of America: New Age International Publishers.

• Takana, Kazuo y Hua O Wang. 2001. *Fuzzy control systems desing and analysis: a linear matrix inequality approach*. United States of America: John Wiley & Sons.

• Goldberg, David E. 1989. *Genetic algorithms in search, optimization, and machine learning*. United States of America: Addison-Wesley Publishing Company, Inc.

• De Jong, Kenneth, Lawrence Fogel y Hans-Paul Schwefel. 1997. *The handbook of evolutionary computation*. United States of America: IOP Publishing Ltd and Oxford University Press.

Artículos

• Duarte V, Oscar G. 2000. Aplicaciones de la lógica difusa. Revista Ingeniería e investigación No. 45

• Santos Peñas, Matilde y Edurne Miranda Suescun. 2012. Aplicación de la lógica difusa en el ámbito de las energías renovables. Revista Elementos Volumen 2, Número 1 (junio).

• Coronel Lemus, Martha Esmeralda y José Antonio Hernández Reyes. 2004. Simulación de sistema difuso para el control de velocidad de un motor C.D. Memorias del 3er Congreso de Cómputo AGECOMP, UAEM, México.

• Yasunobu, Seiji, Shoji Miyamoto e Hirokazu Ihara. 2002. A fuzzy control for train automatic stop control. 21ava Conferencia anual SICE. T.SICE volumen E-2

• Nuno de Castro, Rodrigo y M. Isabel Ribeiro. 2003. Target tracking using fuzzy control. 3er Festival Nacional de Robótica. Actas de encuentro científico (mayo), Lisboa, Portugal.

• García, Miguel Ángel y Luis Álvarez-Icaza. 2011. Control de vibraciones en edificios con base en estimadores. Instituto de Ingeniería, Universidad Nacional Autónoma de México (noviembre).

• Jimenez, René y Luis Álvarez-Icaza. 2004. LuGre friction model for a magnetorheological damper. Publicado en linea en Wiley InterScience (www.intersciense.wiley.com). DOI: 10.1002/stc.58

• Álvarez, Luis y René Jiménez. 2004. Civil structures semi-active control with limited measurements. Proceedings of the 2004 American Control Conference (junio), Boston, USA. Páginas 5467 - 5471

• Morales, Jesús y Luis Álvarez-Icaza. 2013. Identificación de edificios en tiempo real mediante ruido ambiental. Memorias del Congreso Nacional de Control Automático (octubre), México. Páginas 213 - 218.

• Ishibuchi, Hisao, Tomoharu Nakashima and Tadahiko Murata. 1997. Comparsion of the Michigan and Pittsburgh approaches to the desing of fuzzy classification systems. Electronics and Communications in Japan, Parte 3, Volumen 80, Número 12.

• Herrera, Francisco. 2004. Sistemas difusos evolutivos. Actas del XII congreso español sobre tecnologías y lógica fuzzy (septiembre), Jaén, España.

• Dyke, S., B. Spencer, M. Sain, and J. Carlson (1996a). Modelling and control of magnetorheological dampers for seismic response reduction. *Smart Materials and Structures 5(5)*, 565–575.

• Mousaad Aly, Aly. 2013. Vibration control of buildings using magnetorheological damper: a new control algorithm. Journal of Engineering. Hindawi Publishing Corporation. Volume 2013.

• Chang, Chia-Ming and Billie F. Spencer Jr. Active base isolation of buildings subjected to seismic excitations. 2010. Publicado en linea Wiley Online Library (http://onlinelibrary.wiley.com/). DOI: 10.1002/eqe.1040

Tesis

• Alcaraz Herrera, Hugo Israel. 2012. Control de un robot móvil aplicando control difuso y visión artificial desarrollado en LabVIEW. Tesis de Licenciatura. Facultad de Ingeniería, UNAM.

Apéndice A - Lógica Difusa

También conocida como *fuzzy logic,* la lógica difusa es una metodología que permite manejar información imprecisa, vaga o ambigua: "Mucha fuerza", "muy rápido", "un poco frío" y a partir de ésta, obtener una conclusión de forma simple y elocuente. La lógica difusa emula la forma en cómo los humanos razonan, es decir, emula la forma en cómo una persona toma una decisión basándose en determinada información cuyas características son las mencionadas anteriormente (Alcaraz 2012).

Lotfi Asker Zadeh, un eminente profesor de la Universidad de California, Berkeley, fue quien en 1965 desarrolló el concepto de lógica difusa basado en los *conjuntos difusos*. Dichos conjuntos permiten que un elemento siempre tenga un cierto grado de pertenencia a un conjunto, en comparación con los conjuntos clásicos, que sólo permiten al elemento formar parte o no de un conjunto.

Conjuntos Difusos

Un Conjunto Difuso A en el universo del discurso U, tiene como característica que su función de membresía o pertenencia puede tomar valores en el intervalo [0,1] y puede representarse como un conjunto de pares ordenados de un elemento x y su valor de pertenencia al conjunto:

$$A = \{(x, \mu_A(x)) \mid x \in U\}$$

A los conjuntos difusos se les puede aplicar ciertos operadores o pueden realizarse operaciones entre ellos. Cuando se aplica un operador sobre un conjunto difuso, se obtiene otro conjunto difuso. De forma análoga, sucede al combinar dos o más conjuntos difusos con una operación.

Sean los conjuntos difusos *A* y *B* asociados a una variable lingüística x, pueden definirse tres operaciones básicas: complemento, intersección y unión:

- Complemento

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

- Unión

$$\mu_{A \cup B}(x) = \max[\mu_A, \mu_B]$$

• Intersección

$$\mu_{A\cap B}(x) = \min[\mu_A, \mu_B]$$

Estas tres operaciones cumplen con la asociatividad, conmutatividad y distributividad como en la teoría de conjuntos clásica. Existen dos leyes de la teoría clásica de conjuntos que no se cumplen en la teoría de conjuntos difusos, El Principio de Contradicción: $A \cup \bar{A} = U$ y el Principio de Exclusión: $A \cap \bar{A} = \varnothing$. Una de las formas para describir en qué se diferencian la teoría clásica de conjuntos con respecto a la teoría de conjuntos difusos es justamente el incumplimiento de estas dos leyes.

A continuación se presentan algunas propiedades que sólo se cumplen en los conjuntos difusos:

$$U \cap \bar{U} \neq \varnothing$$

$$U \cup \bar{U} \neq N$$

$$A \cap \varnothing = \varnothing \text{ ó } \mu_A \wedge 0 = 0$$

$$A \cup \varnothing = A \text{ ó } \mu_A \vee 0 = \mu_A$$

$$A \cap N = A \text{ ó } \mu_A \wedge 1 = \mu_A$$

$$A \cup N = N \text{ ó } \mu_A \vee 1 = 1$$

Donde *N* es el conjunto unidad y ∅ es el conjunto vacío.

Las funciones que corresponden a la unión y a la intersección pueden generalizarse siempre y cuando se cumplan ciertas restricciones, dichas funciones son conocidas como Norma triangular (t-norma) y Conorma triangular (t-conorma). Algunos de los operadores que cumplen con las condiciones para ser Normas y Conormas son:

Tabla A1.1 Algunos operadores que son normas o conormas

Normas	Conormas
min(a,b)	max(a,b)
(a+b-ab)	(ab)
a * b = MAX(0,a+b-1)	$a \dot{+} b$ = MIN(1,a+b)

Función de Membresía o Pertenencia

La función de membresía o pertenencia es una simple medida del grado con que x pertenece al conjunto A. Esto se representa de la siguiente manera:

$$\mu A(x): U \rightarrow [0,1]$$

Para definir estas funciones, suelen usarse convencionalmente ciertas familias de formas estándar para coincidir con el significado lingüístico de las etiquetas más utilizadas aunque prácticamente pueden definirse de cualquier forma. Existen funciones que por su simplicidad matemática y computacional son frecuentemente usadas en los sistemas difusos, algunas de ellas son: trapezoidal, triangular, singletone, gamma, campana gaussiana, etc. A continuación, se presentan algunas definiciones de funciones de membresía o pertenencia mencionadas anteriormente.

• Función de membresía o pertenencia trapezoidal

Se define como:

$$A = \begin{cases} 0 & si\,(x \le a)\ o\ (x \ge d) \\ (x-a)/(b-a) & si\ x \in (a,b] \\ 1 & si\ x \in (b,c) \\ (d-x)/(d-c) & si\ x \in (b,d) \end{cases}$$

Y su gráfica característica se muestra en la Figura A1.1:

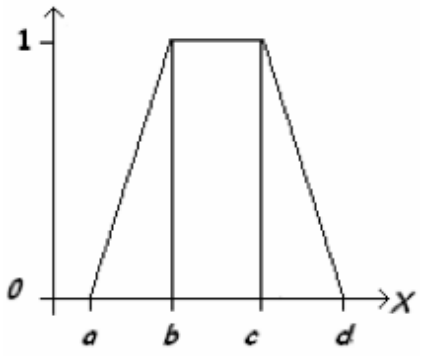

Figura A1.1 Función de pertenencia o membresía trapezoidal

Se utiliza básicamente para sistemas difusos sencillos ya que es posible definir un conjunto difuso con pocos datos y calcular el valor de pertenencia con pocas operaciones. Por estas características, se emplea en sistemas basados en un microprocesador.

• Función de membresía o pertenencia triangular

Se define como:

$$A = \begin{cases} 0 & si\ x \le a \\ (x-a)/(m-a) & si\ x \in (a,m] \\ (b-x)/(b-m) & si\ x \in (m,b) \\ 0 & si\ x \ge b \end{cases}$$

Y su gráfica característica se muestra en la Figura A1.2:

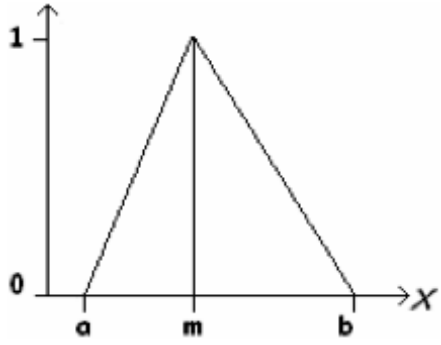

Figura A1.2 Función de pertenencia o membresía triangular

Esta función es adecuada para modelar propiedades con un valor de pertenencia distinto de cero para un rango de valores estrecho entorno a un punto *m*.

• Función de membresía o pertenencia gaussiana

Se define como:

$$A = e^{-k(x-m)^2}$$

donde k>0

La gráfica característica se muestra en la figura A1.3:

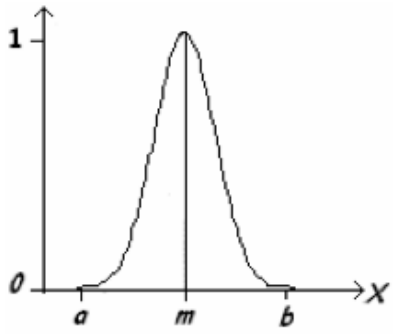

Figura A1.3 Función de pertenencia o membresía Gaussiana

Esta función tiene forma de campana, resulta adecuada para los conjuntos definidos en torno al valor *m* como *medio, normal, cero*, etc. Puede definirse también usando expresiones analíticas exponenciales o cuadráticas como la misma campana de Gauss.

Variable Lingüística

Un conjunto difuso se encuentra relacionado a un valor lingüístico que está definido por una palabra, un adjetivo o una etiqueta lingüística. Una variable lingüística es aquel concepto que será evaluado de forma difusa y que se le puede aplicar un adjetivo que define sus características mediante el lenguaje natural. Por ejemplo: La temperatura, la edad, la velocidad. Una variable difusa o lingüística es una variable cuyos valores se pueden considerar

etiquetas de los conjuntos difusos. En la Figura A1.4, se muestra la variable lingüística "Altura" y sus conjuntos difusos asociados "Bajo", "Mediano" y "Alto".

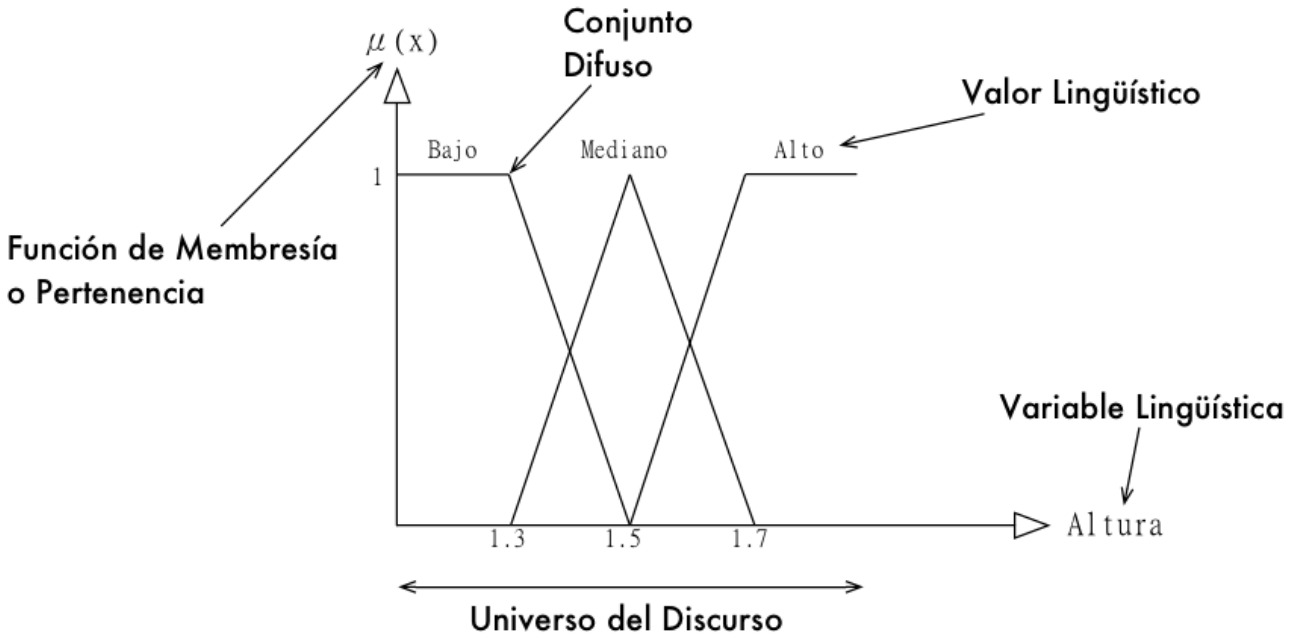

Figura A1.4 Variable lingüística y sus valores lingüísticos asociados a conjuntos difusos

Sistema de Inferencia Difusa o Controlador difuso

La lógica difusa se usa en sistemas de control difuso que usan expresiones ambiguas para generar reglas que controlan al sistema. Estos sistemas de control funcionan de forma muy diferente a los convencionales; usan el conocimiento de un experto para generar una base de conocimiento que dará al sistema la capacidad de elegir sobre ciertas situaciones. Los Sistemas de Control Difuso permiten diseñar un conjunto de reglas que utilizaría una persona (experto) para controlar un proceso, y a partir de las reglas generar un control sobre determinadas acciones. La estructura de un sistema de inferencia difusa o controlador difuso se muestra en la Figura A1.5.

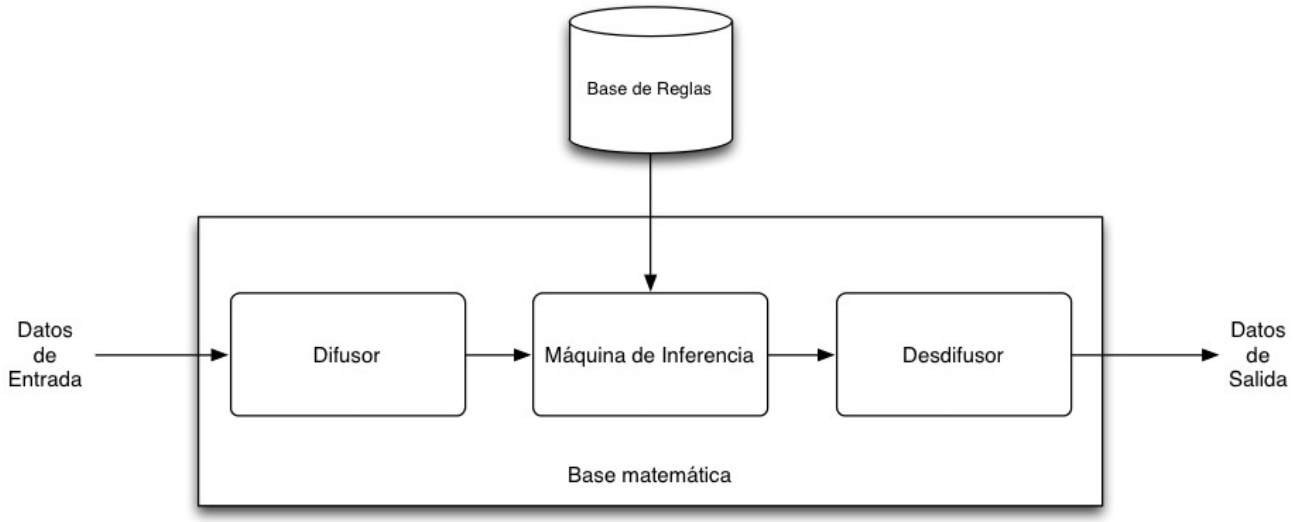

Figura A1.5 Diagrama general de un controlador difuso

• Difusor

Tiene como objetivo convertir valores reales o concretos en valores difusos, es decir, a las variables de entrada se les asignan grados de pertenencia en relación a los conjuntos difusos previamente definidos usando las funciones de pertenencia asociadas a los conjuntos difusos. Cada conjunto difuso producido por este bloque está definido sobre el universo del discurso de la variable lingüística respectiva y tiene una función de pertenencia cuya forma puede ser distinta para cada variable de entrada.

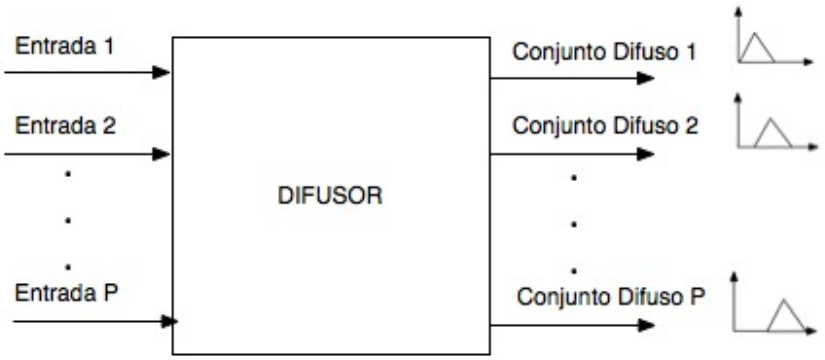

Figura A1.6 Diagrama del bloque difusor

Las entradas de este bloque son valores concretos o "crisp" de las variables de entrada y las salidas son grados de pertenencia a los conjuntos difusos considerados.

• *Máquina de inferencia*

Este bloque relaciona los conjuntos difusos de entrada con los de salida para representar las reglas que definirán al sistema difuso. En la máquina de inferencia (ver Figura A1.7), se usa la información de la base de reglas o base de conocimiento para generar reglas mediante el uso de condiciones. La base de reglas se obtiene para *n* variables de entrada y *m* reglas puede representarse como:

$$Si\ x_1\ es\ A_1\ ,\ \ldots\ ,\ x_n\ es\ A_n\ entonces\ "y"\ es\ B_m$$

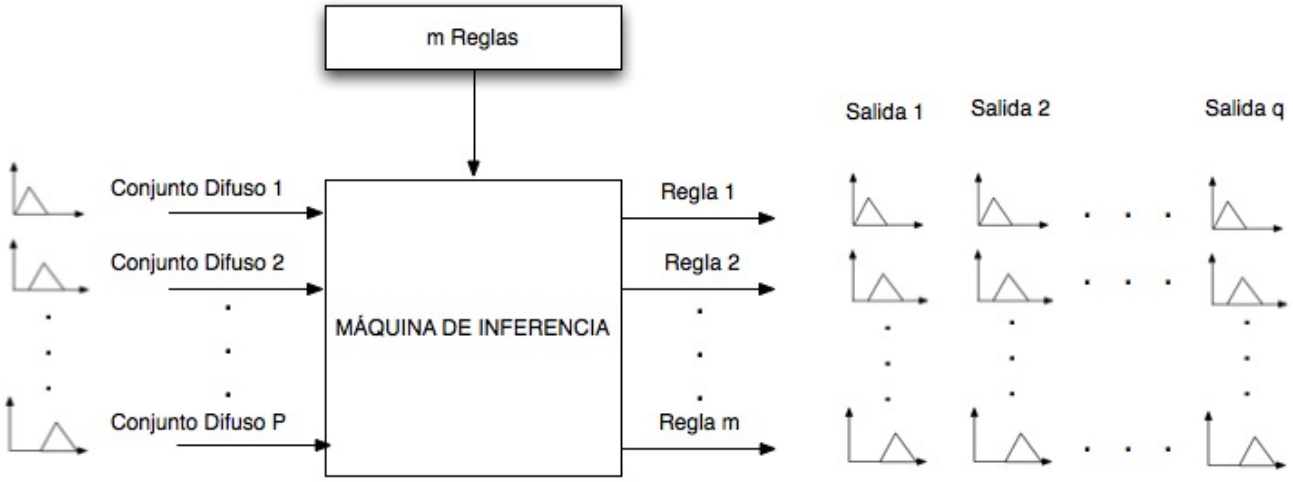

Figura A1.7 Diagrama del bloque de la Máquina de Inferencia

En este bloque, la máquina de inferencia recibe *P* conjuntos difusos producidos por el bloque difusor y los aplica a cada una de las *m* reglas de la base de reglas para producir *m*q* conjuntos difusos, definidos sobre los universos de discurso de las variables lingüísticas de salida.

Las entradas de este bloque son conjuntos difusos (grados de pertenencia) y las salidas son también conjuntos difusos, asociados a las variables de salida.

• *Desdifusor*

Este bloque que se ilustra en la Figura A1.8, tiene como objetivo obtener un resultado, es decir, un valor real o concreto de la variable de salida a partir del conjunto difuso obtenido de la máquina de inferencia mediante métodos matemáticos de desdifusión.

En general, para producir cada uno de los *q* valores concretos, el difusor toma *m* conjuntos difusos correspondientes a cada variable de salida y mediante algún método o algoritmo, se produce un valor real o concreto.

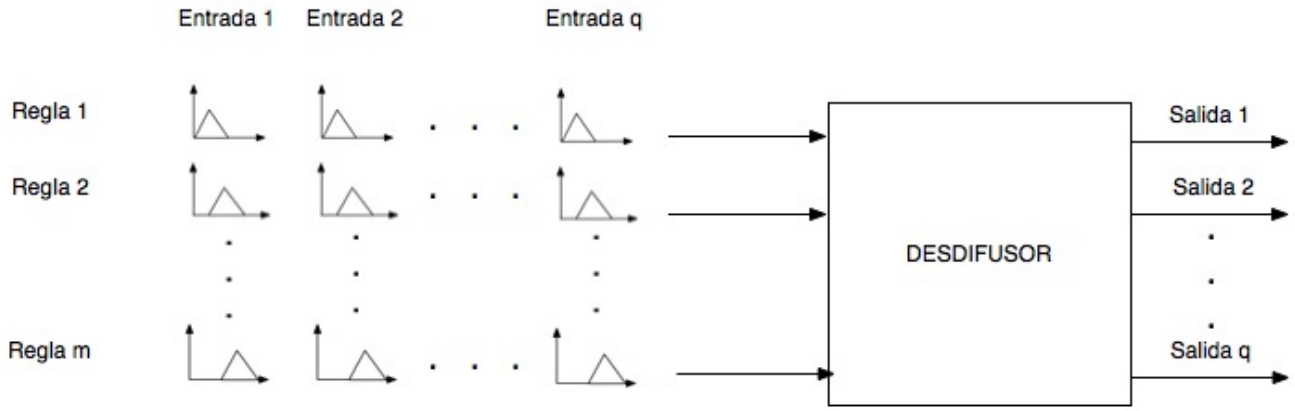

Figura A1.8 Diagrama del bloque del desdifusor

Existen diversos métodos o algoritmos de desdifusión, los más utilizados son:

- Método del máximo

Se elige como valor para la variable de salida, la función característica del conjunto difuso de salida con el máximo valor. Este método no es óptimo y puede generar demasiados errores ya que el valor máximo puede ser alcanzado por varias salidas.

- Método de altura

Se calcula para cada regla el centro de gravedad del conjunto difuso de salida B_m y después se calcula la salida del sistema como la media ponderada. Está definido por:

$$y_h = \frac{\int \bar{y}_m \mu_{B_m}(\bar{y}_m) dy}{\int \mu_{B_m}(\bar{y}_m) dy}$$

- Método del centroide

Utiliza como salida del sistema, el centro del área bajo la función de pertenencia que se ha obtenido para los conjuntos difusos de salida. Para obtener el centroide, se utiliza la siguiente ecuación.

$$\hat{y} = \frac{\int y\mu_B(y)dy}{\int \mu_B(y)dy}$$

Donde μ_B(y) es la función de pertenencia de cada salida obtenida, mientras que la integral es valuada en el intervalo del soporte del conjunto de salida. El soporte de un conjunto difuso es el conjunto que contiene todos los elementos del universo del discurso U que tienen un valor de pertenencia distinto de 0.

Este método es el más utilizado en las aplicaciones de lógica difusa y en la ingeniería ya que con este método se obtiene una solución única.

Reglas Difusas

Las reglas difusas combinan uno o más conjuntos difusos de entrada, llamados *antecedentes o premisas*, y se les asocia un conjunto de salida llamado *consecuente o consecuencia.* A las premisas se les puede asociar mediante conectivos lógicos como *y, o,* etc. Una regla típica tiene la forma IF-THEN.

Las reglas difusas permiten expresar el conocimiento acerca de la relación entre antecedentes y consecuentes. Dicho conocimiento no se basa en sólo una regla, normalmente son varias reglas que se pueden agrupar formando una *base de reglas o base de conocimiento.*

Formalmente, una base de reglas difusas está definida como una colección de reglas $R^{(L)}$:

$$R^{(L)}: \text{IF } x_1 \text{ es } F_1^L, \ldots, x_n \text{ is } F_n^L \text{ THEN } y \text{ es } G^L$$

donde

F_i^L y G^L son conjuntos difusos en $U_i \subset \Re$ y $V \subset \Re$, respectivamente y

$\mathbf{x} = [x_1 \ldots x_n]^T \in U_1 \times U_2 \ldots U_n$

$y \in V$

Tal que **x**, y son variables lingüísticas.

Este formato de reglas es conocido como tipo *Mandami*. En él, la función de salida es un conjunto difuso.

Existe otro formato que es llamado tipo *Sugeno*. Se caracteriza porque la función de salida es una combinación lineal de las variables de entrada.

$$R^{(L)}: IF\ x_1\ es\ F_1{}^L, \ldots, x_n\ es\ F_n{}^L\ THEN\ y^L = f^L(x)$$

En ambos casos, nombramos *S* al número de reglas IF-ELSE de la base de reglas, entonces L= 1,2, ... ,S. El vector **x** representa el conjunto de entradas mientras que "*y*" representa la salida del sistema difuso.

Los sistemas difusos con x_n entradas y una salida "*y*", son llamados *MISO(Multiple Input Single Output)* y los sistemas difusos con múltiples entradas y múltiples salidas (de 1 hasta *k*) son conocidos como *MIMO (Multiple Input Multiple Output)*.

Una forma típica usada en el *control difuso* para una base de conocimiento está representada de la forma:

$R^{(L)}$: IF A_1 THEN B_1
OR
IF A_2 THEN B_2
OR
.
.
.
IF A_N THEN B_N

Donde A_1, A_2, ... , A_n son conjuntos difusos de *U* y B_1, B_2, ... , B_n son conjuntos difusos de V, correspondientes a los antecedentes y consecuentes, respectivamente.

Apéndice B - Código del amortiguador

ModeloMatematico.java

```
public class ModeloMatematico {

    private final double intervaloTiempo = 0.005;
    private final double sigma0 = 105900.0;
    private final double sigma1 = 5700.0;
    private final double sigma2 = 2300.0;
    private final double a0 = 0.0030;
    private final double a1 = -0.1444;
    private double zetaKActual = -0.0004*0.0;
    private double zetaKPuntoActual = 0.0;
    private double fuerzaAmortiguador = 0.0;

    public ModeloMatematico() {

    }

    public double getZetaKActual() {
        return zetaKActual;
    }

    public void setZetaKActual(double zetaKActual) {
        this.zetaKActual = zetaKActual;
    }

    public double getZetaKPuntoActual() {
        return zetaKPuntoActual;
    }

    public void setZetaKPuntoActual(double zetaKPuntoActual) {
        this.zetaKPuntoActual = zetaKPuntoActual;
    }

    public double getFuerzaAmortiguador() {
        return fuerzaAmortiguador;
    }

    public void setFuerzaAmortiguador(double fuerzaAmortiguador) {
        this.fuerzaAmortiguador = fuerzaAmortiguador;
    }


    public void evaluaModelo(double velocidadPiso1, double voltaje) {

        double zetaKMasUnoT;
        double velocidadZeta;

        zetaKMasUnoT = evaluaEcuacionZetaKMasUno();
        velocidadZeta = evaluaEcuacionVelocidadZeta(zetaKMasUnoT, velocidadPiso1, voltaje);
        evaluaEcuacionFuerzaAmortiguador(zetaKMasUnoT,velocidadZeta,
                                                                velocidadPiso1, voltaje);

    }
```

```
    private double evaluaEcuacionZetaKMasUno() {

        double zetaKT = zetaKActual;
        double zetaKMasUno;

        zetaKMasUno = zetaKT+(intervaloTiempo*zetaKPuntoActual);
        zetaKActual = zetaKMasUno;

        return zetaKMasUno;

    }

    private double evaluaEcuacionVelocidadZeta(double zetaKMasUno, double velocidad,
                                                                                double voltaje) {

        double velocidadKT;
        double sigma0A0;
        double unoA1Voltaje;
        double velocidadKTAbsoluta;
        double zetaKT;
        double velocidadZeta;

        velocidadKT = velocidad;
        sigma0A0 = sigma0*a0;
        unoA1Voltaje = 1+a1*voltaje;
        velocidadKTAbsoluta = Math.abs(velocidadKT);
        zetaKT = zetaKMasUno;
        velocidadZeta = velocidadKT-(sigma0A0*unoA1Voltaje*velocidadKTAbsoluta*zetaKT);
        zetaKPuntoActual = velocidadZeta;

        return velocidadZeta;

    }

    private void evaluaEcuacionFuerzaAmortiguador(double zetaKMasUnoT,double velocidadZeta,
                                                        double velocidad, double voltaje) {

        double sigma0ZetaVoltaje;
        double sigma1VelocidadZeta;
        double sigma2Velocidad;
        double fuerzaAmortiguadorModelo;

        sigma0ZetaVoltaje = sigma0*zetaKMasUnoT*voltaje;
        sigma1VelocidadZeta = sigma1*velocidadZeta;
        sigma2Velocidad = sigma2*velocidad;
        fuerzaAmortiguadorModelo = sigma0ZetaVoltaje+sigma1VelocidadZeta+sigma2Velocidad;

        if(fuerzaAmortiguadorModelo > 2725.5292961479017){

            fuerzaAmortiguadorModelo = 2725.5292961479017;

        } else {

            if(fuerzaAmortiguadorModelo < -2725.5292961479017){

                fuerzaAmortiguadorModelo = -2725.5292961479017;

            }

        }

        fuerzaAmortiguador = fuerzaAmortiguadorModelo;

    }

}
```

Amortiguador.java

```
public class Amortiguador {

    private ModeloMatematico modelo = new ModeloMatematico();
    private double fuerzaAmortiguadorActual = 0.0;

    public Amortiguador() {

    }

    public ModeloMatematico getModelo() {
        return modelo;
    }

    public void setModelo(ModeloMatematico modelo) {
        this.modelo = modelo;
    }

    public double getFuerzaAmortiguadorActual() {
        return fuerzaAmortiguadorActual;
    }

    public void setFuerzaAmortiguadorActual(double fuerzaAmortiguadorActual) {
        this.fuerzaAmortiguadorActual = fuerzaAmortiguadorActual;
    }

    public void evaluaModeloMatematico(double velocidadPiso1, double voltaje) {

        modelo.evaluaModelo(velocidadPiso1, acondicionaVoltaje(voltaje));
        fuerzaAmortiguadorActual = modelo.getFuerzaAmortiguador();
    }

    private double acondicionaVoltaje(double preVoltaje) {

        double voltaje = (preVoltaje/2)+4.5;

        return voltaje;

    }

}
```

Apéndice C - Código del edificio

ModeloMatematico.java

```
public class ModeloMatematico {

    private int numeroDePisosDelModelo;
    private final double intervaloTiempo =0.005;
    private final double masa = 9.5;
    private double [][] matrizMasa;
    private double [][] matrizRigidez;
    private double [][] matrizAmortiguamiento;
    private double [][] matrizMasaRigidez;
    private double [][] matrizMasaAmortiguamiento;
    private double [] vectorXPuntoActual = {0.0,0.0,0.0,0.0,0.0};
    private double [] vectorXActual = {0.0,0.0,0.0,0.0,0.0};

    public ModeloMatematico() {

    }

    public double[][] getMatrizMasa() {
        return matrizMasa;
    }

    public void setMatrizMasa(double[][] matrizMasa) {
        this.matrizMasa = matrizMasa;
    }

    public double[][] getMatrizRigidez() {
        return matrizRigidez;
    }

    public int getNumeroDePisosDelModelo() {
        return numeroDePisosDelModelo;
    }

    public void setNumeroDePisosDelModelo(int numeroDePisosDelModelo) {
        this.numeroDePisosDelModelo = numeroDePisosDelModelo;
    }

    public void setMatrizRigidez(double[][] matrizRigidez) {
        this.matrizRigidez = matrizRigidez;
    }

    public double[][] getMatrizAmortiguamiento() {
        return matrizAmortiguamiento;
    }

    public void setMatrizAmortiguamiento(double[][] matrizAmortiguamiento) {
        this.matrizAmortiguamiento = matrizAmortiguamiento;
    }

    public double[] getVectorXPuntoActual() {
        return vectorXPuntoActual;
    }


    public void setVectorXPuntoActual(double[] vectorXPuntoActual) {
        this.vectorXPuntoActual = vectorXPuntoActual;
    }
```

```
public double[] getVectorXActual() {
    return vectorXActual;
}

public void setVectorXActual(double[] vectorXActual) {
    this.vectorXActual = vectorXActual;
}

public double[][] getMatrizMasaRigidez() {
    return matrizMasaRigidez;
}

public void setMatrizMasaRigidez(double[][] matrizMasaRigidez) {
    this.matrizMasaRigidez = matrizMasaRigidez;
}

public double[][] getMatrizMasaAmortiguamiento() {
    return matrizMasaAmortiguamiento;
}

public void setMatrizMasaAmortiguamiento(double[][] matrizMasaAmortiguamiento) {
    this.matrizMasaAmortiguamiento = matrizMasaAmortiguamiento;
}


public void preparaModelo(int numeroDePisos){

    numeroDePisosDelModelo = numeroDePisos;
    matrizMasa = new double [numeroDePisosDelModelo][numeroDePisosDelModelo];
    matrizRigidez = new double [numeroDePisosDelModelo]
                                                    [numeroDePisosDelModelo];
    matrizAmortiguamiento = new double [numeroDePisosDelModelo]
                                                    [numeroDePisosDelModelo];
    matrizMasaRigidez = new double [numeroDePisosDelModelo]
                                                    [numeroDePisosDelModelo];
    matrizMasaAmortiguamiento = new double [numeroDePisosDelModelo]
                                                    [numeroDePisosDelModelo];
    construyeMatrices(numeroDePisosDelModelo);

}

public void evaluaModelo(double aceleracionTierra, double fuerzaAmortiguador) {

    double [] vectorXPunto = calculaVelocidadesPisos(aceleracionTierra, fuerzaAmortiguador);
    vectorXPuntoActual = vectorXPunto;
    double [] vectorX = calculaDesplazamientosPisos(vectorXPunto);
    vectorXActual = vectorX;

}

private void construyeMatrices(int numeroDePisos) {

    construyeMatrizMasaRigidez(numeroDePisos);
    construyeMatrizMasaAmortiguamiento();

}
private void construyeMatrizMasaRigidez(int numeroDePisos) {

    construyeMatrizMasa(numeroDePisos);
    construyeMatrizRigidez();

    for(int h=0; h<numeroDePisosDelModelo; h++){

        for(int i=0; i<numeroDePisosDelModelo; i++){

            matrizMasaRigidez[h][i] = 0.0;
```

```
            for(int a=0; a<numeroDePisosDelModelo; a++){
                if(matrizMasa[h][a] == 0.0){
                    matrizMasaRigidez[h][i] += 0.0*matrizRigidez[a][i];
                } else {
                matrizMasaRigidez[h][i] += (-1/matrizMasa[h][a])*matrizRigidez[a][i];
                }
            }
        }
    }
}
private void construyeMatrizMasaAmortiguamiento() {
    construyeMatrizAmortiguamiento();
    for(int h=0; h<numeroDePisosDelModelo; h++){
        for(int i=0; i<numeroDePisosDelModelo; i++){
            matrizMasaAmortiguamiento[h][i] = 0.0;
            for(int a=0; a<numeroDePisosDelModelo; a++) {
                if(matrizMasa[h][a] == 0.0) {
                    matrizMasaAmortiguamiento[h][i] += 0.0*matrizAmortiguamiento[a][i];
                } else {
                    matrizMasaAmortiguamiento[h][i] = (-1/matrizMasa[h[a])*
                                                        matrizAmortiguamiento[a][i];
                }
            }
        }
    }
}
private void construyeMatrizMasa(int numeroDePisos) {
    for(int h=0; h<numeroDePisos; h++){
        for(int i=0; i<numeroDePisos; i++){
            if(h == i){
                matrizMasa[h][i] = masa;
            }else{
                matrizMasa[h][i] = 0.0;
            }
        }
```

```
    }

}

private void construyeMatrizRigidez() {

    double k = 10000;

    matrizRigidez[0][0]=k*2.6315; matrizRigidez[0][1]=k*-1.0748;
    matrizRigidez[0][2]=0.0; matrizRigidez[0][3]=0.0;
    matrizRigidez[0][4]=0.0; matrizRigidez[1][0]=k*-1.1796;
    matrizRigidez[1][1]=k*2.4327; matrizRigidez[1][2]=k*-1.2530;
    matrizRigidez[1][3]=0.0; matrizRigidez[1][4]=0.0;
    matrizRigidez[2][0]=0.0; matrizRigidez[2][1]=k*-1.2800;
    matrizRigidez[2][2]=k*2.5215; matrizRigidez[2][3]=k*-1.2416;
    matrizRigidez[2][4]=0.0; matrizRigidez[3][0]=0.0;
    matrizRigidez[3][1]=0.0; matrizRigidez[3][2]=k*-1.2863;
    matrizRigidez[3][3]=k*2.5946; matrizRigidez[3][4]=k*-1.3083;
    matrizRigidez[4][0]=0.0; matrizRigidez[4][1]=0.0;
    matrizRigidez[4][2]=0.0; matrizRigidez[4][3]=k*-1.2427;
    matrizRigidez[4][4]=k*1.2427;

}

private void construyeMatrizAmortiguamiento() {

    matrizAmortiguamiento[0][0]=18.5325; matrizAmortiguamiento[0][1]=-6.0971;
    matrizAmortiguamiento[0][2]=0.0; matrizAmortiguamiento[0][3]=0.0;
    matrizAmortiguamiento[0][4]=0.0; matrizAmortiguamiento[1][0]=-24.5452;
    matrizAmortiguamiento[1][1]=33.6949; matrizAmortiguamiento[1][2]=-9.1498;
    matrizAmortiguamiento[1][3]=0.0; matrizAmortiguamiento[1][4]=0.0;
    matrizAmortiguamiento[2][0]=0.0; matrizAmortiguamiento[2][1]=-11.8545;
    matrizAmortiguamiento[2][2]=16.9426; matrizAmortiguamiento[2][3]=-5.0881;
    matrizAmortiguamiento[2][4]=0.0; matrizAmortiguamiento[3][0]=0.0;
    matrizAmortiguamiento[3][1]=0.0; matrizAmortiguamiento[3][2]=-5.4720;
    matrizAmortiguamiento[3][3]=6.1502; matrizAmortiguamiento[3][4]=-0.6783;
    matrizAmortiguamiento[4][0]=0.0; matrizAmortiguamiento[4][1]=0.0;
    matrizAmortiguamiento[4][2]=0.0; matrizAmortiguamiento[4][3]=-2.5259;
    matrizAmortiguamiento[4][4]=2.5259;

}

private double[] calculaVelocidadesPisos(double aceleracionTierra,
                                                                    double fuerzaAmortiguador) {

    double [] matrizMaRiPosicion = new double [numeroDePisosDelModelo];

    for(int h=0; h<numeroDePisosDelModelo; h++){

        for(int i=0; i<numeroDePisosDelModelo; i++){

            matrizMaRiPosicion [h] += matrizMasaRigidez[h][i]*vectorXActual[i];

        }

    }

    double [] matrizMaAmVelocidad = new double [numeroDePisosDelModelo];

            for(int h=0; h<numeroDePisosDelModelo; h++){

                for(int i=0; i<numeroDePisosDelModelo; i++){

                        matrizMaAmVelocidad [h] +=
                        matrizMasaAmortiguamiento[h][i]*vectorXPuntoActual[i];
```

```
            }

        }

        double [] matrizMaAmVelSuelo = new double [numeroDePisosDelModelo];
        double [] matrizAceleracionesSuelo = new double [numeroDePisosDelModelo];

        for(int h=0; h<numeroDePisosDelModelo; h++){

            matrizAceleracionesSuelo[h] = aceleracionTierra;

        }

        for(int i=0; i<numeroDePisosDelModelo; i++){

            matrizMaAmVelSuelo[i] = matrizMaAmVelocidad[i] - matrizAceleracionesSuelo[i];

        }

        double [] matrizFuerzaAmortiguador = new double [numeroDePisosDelModelo];
        matrizFuerzaAmortiguador[0] = fuerzaAmortiguador;

        for(int z=1; z<numeroDePisosDelModelo; z++){

            matrizFuerzaAmortiguador[z] = 0.0;

        }

        double [] matrizXPunto = new double [numeroDePisosDelModelo];

        for(int h=0; h<numeroDePisosDelModelo; h++){

            matrizXPunto[h] = vectorXPuntoActual[h] + intervaloTiempo*(matrizMaRiPosicion[h]+
                                matrizMaAmVelSuelo[h]-(matrizFuerzaAmortiguador[h]/masa));

        }

        return matrizXPunto;

    }

    private double [] calculaDesplazamientosPisos(double[] velPis){

        double [] matrizX = new double [numeroDePisosDelModelo];

        for(int h=0; h<numeroDePisosDelModelo; h++){

            matrizX[h] = vectorXActual[h] + intervaloTiempo*(velPis[h]);

        }

        return matrizX;

    }

}
```

Apéndice D - Código del controlador difuso

ControladorDifuso.java

```
import controladorDifuso.desdifusor.Desdifusor;
import controladorDifuso.difusor.Difusor;
import controladorDifuso.maquinaDeInferencia.MaquinaDeInferencia;


public class ControladorDifuso {

    private double valorDeEntradaVelocidad;
    private double ValorDeEntradaDesplazamiento;
    private double valorDeSalidaVoltaje;
    private Difusor difusor = new Difusor(-50.0, 50.0, -0.75, 0.75);
    private MaquinaDeInferencia maquinaDeInferencia = new MaquinaDeInferencia();
    private Desdifusor desdifusor = new Desdifusor(0.25, 5.0);

    public ControladorDifuso() {

    }

    public double getValorDeEntradaVelocidad() {
        return valorDeEntradaVelocidad;
    }

    public void setValorDeEntradaVelocidad(double valorDeEntradaVelocidad) {
        this.valorDeEntradaVelocidad = valorDeEntradaVelocidad;
    }

    public double getValorDeEntradaDesplazamiento() {
        return ValorDeEntradaDesplazamiento;
    }

    public void setValorDeEntradaDesplazamiento(double ValorDeEntradaDesplazamiento) {
        this.ValorDeEntradaDesplazamiento = ValorDeEntradaDesplazamiento;
    }

    public double getValorDeSalidaVoltaje() {
        return valorDeSalidaVoltaje;
    }

    public void setValorDeSalidaVoltaje(double valorDeSalidaVoltaje) {
        this.valorDeSalidaVoltaje = valorDeSalidaVoltaje;
    }


    public Difusor getDifusor() {
        return difusor;
    }

    public void setDifusor(Difusor difusor) {
        this.difusor = difusor;
    }

    public MaquinaDeInferencia getMaquinaDeInferencia() {
        return maquinaDeInferencia;
    }

    public void setMaquinaDeInferencia(MaquinaDeInferencia maquinaDeInferencia) {
        this.maquinaDeInferencia = maquinaDeInferencia;
    }
```

```
    public Desdifusor getDesdifusor() {
        return desdifusor;
    }

    public void setDesdifusor(Desdifusor desdifusor) {
        this.desdifusor = desdifusor;
    }

    public void configuraControladorDifuso() {

        difusor.configuraDifusor();
        maquinaDeInferencia.getBase().construyeBaseDeConocimiento();
        desdifusor.configuraDesdifusor();

    }

    public void obtenVoltajeAPartirDeVelocidadYDesplazamiento(double velocidad,
                                                              double desplazamiento) {

        valorDeEntradaVelocidad = velocidad;
        ValorDeEntradaDesplazamiento = desplazamiento;

        difusor.calculaValorDePertenenciaYConjuntoDifuso(velocidad, desplazamiento);
        maquinaDeInferencia.realizaInferencia(
                            difusor.getValoresDePertenenciaVelocidadYDesplazamiento(),
                            difusor.getEtiquetasVelocidadYDesplazamiento());

        desdifusor.calculaValorReal(maquinaDeInferencia.getValoresDeInferenciaSalida(),
                            maquinaDeInferencia.getEtiquetasDeInferenciaSalida());

        valorDeSalidaVoltaje = desdifusor.getValorDeSalida();

    }

    public void limpiaValores() {

        difusor.limpiaValoresDePertenenciaYEtiquetas();
        maquinaDeInferencia.limpiaValoresDeInferenciaYEtiquetasDeInferencia();
        desdifusor.limpiaValorDeSalida();

    }

}
```

Difusor.java

```
import controladorDifuso.difusor.funciones.FuncionGaussianaEntradaDesplazamiento;
import controladorDifuso.difusor.funciones.FuncionGaussianaEntradaVelocidad;
import controladorDifuso.difusor.funciones.FuncionTrapezoidalEntradaDesplazamiento;
import controladorDifuso.difusor.funciones.FuncionTrapezoidalEntradaVelocidad;
import controladorDifuso.difusor.funciones.FuncionTriangularEntradaDesplazamiento;
import controladorDifuso.difusor.funciones.FuncionTriangularEntradaVelocidad;
import java.util.ArrayList;

public class Difusor {

    private ArrayList<FuncionDePertenenciaEntrada> universoDelDiscursoVariableVelocidad =
                                                                        new ArrayList<>();
    private ArrayList<FuncionDePertenenciaEntrada> universoDelDiscursoVariableDesplazamiento
                                                                        = new ArrayList<>();
    private double limiteInferiorVariableVelocidad;
    private double limiteSuperiorVariableVelocidad;
    private double limiteInferiorVariableDesplazamiento;
    private double limiteSuperiorVariableDesplazamiento;
    private ArrayList<Double> valoresDePertenenciaVelocidadYDesplazamiento = new
                                                                           ArrayList<>();
    private ArrayList<String> etiquetasVelocidadYDesplazamiento = new ArrayList<>();

    public Difusor() {

    }

    public Difusor(double limiteInferiorVel, double limiteSuperiorVel, double
                      limiteInferiorDes, double limiteSuperiorDes) {

        this.limiteInferiorVariableVelocidad = limiteInferiorVel;
        this.limiteSuperiorVariableVelocidad = limiteSuperiorVel;
        this.limiteInferiorVariableDesplazamiento = limiteInferiorDes;
        this.limiteSuperiorVariableDesplazamiento = limiteSuperiorDes;

    }

    public ArrayList<FuncionDePertenenciaEntrada> getUniversoDelDiscursoVariableVelocidad() {
        return universoDelDiscursoVariableVelocidad;
    }

    public void setUniversoDelDiscursoVariableVelocidad(ArrayList<FuncionDePertenenciaEntrada>
                                                    universoDelDiscursoVariableVelocidad) {
        this.universoDelDiscursoVariableVelocidad = universoDelDiscursoVariableVelocidad;
    }

    public ArrayList<FuncionDePertenenciaEntrada> getUniversoDelDiscursoVariable
                                                                        Desplazamiento() {
        return universoDelDiscursoVariableDesplazamiento;
    }

    public void setUniversoDelDiscursoVariableDesplazamiento(
              ArrayList<FuncionDePertenenciaEntrada> universoDelDiscursoVariableDesplazamiento) {
        this.universoDelDiscursoVariableDesplazamiento =
                                                   universoDelDiscursoVariableDesplazamiento;
    }

    public double getLimiteInferiorVariableVelocidad() {
        return limiteInferiorVariableVelocidad;
    }

    public void setLimiteInferiorVariableVelocidad(double limiteInferiorVariableVelocidad) {
        this.limiteInferiorVariableVelocidad = limiteInferiorVariableVelocidad;
    }
```

```
public double getLimiteSuperiorVariableVelocidad() {
    return limiteSuperiorVariableVelocidad;
}

public void setLimiteSuperiorVariableVelocidad(double limiteSuperiorVariableVelocidad) {
    this.limiteSuperiorVariableVelocidad = limiteSuperiorVariableVelocidad;
}

public double getLimiteInferiorVariableDesplazamiento() {
    return limiteInferiorVariableDesplazamiento;
}

public void setLimiteInferiorVariableDesplazamiento(
                                            double limiteInferiorVariableDesplazamiento) {
    this.limiteInferiorVariableDesplazamiento = limiteInferiorVariableDesplazamiento;
}

public double getLimiteSuperiorVariableDesplazamiento() {
    return limiteSuperiorVariableDesplazamiento;
}

public void setLimiteSuperiorVariableDesplazamiento(
                                            double limiteSuperiorVariableDesplazamiento) {
    this.limiteSuperiorVariableDesplazamiento = limiteSuperiorVariableDesplazamiento;
}

public ArrayList<Double> getValoresDePertenenciaVelocidadYDesplazamiento() {
    return valoresDePertenenciaVelocidadYDesplazamiento;
}

public void setValoresDePertenenciaVelocidadYDesplazamiento(
                          ArrayList<Double> valoresDePertenenciaVelocidadYDesplazamiento) {
    this.valoresDePertenenciaVelocidadYDesplazamiento =
                                              valoresDePertenenciaVelocidadYDesplazamiento;
}

public ArrayList<String> getEtiquetasVelocidadYDesplazamiento() {
    return etiquetasVelocidadYDesplazamiento;
}

public void setEtiquetasVelocidadYDesplazamiento(ArrayList<String>
                                                    etiquetasVelocidadYDesplazamiento) {
    this.etiquetasVelocidadYDesplazamiento = etiquetasVelocidadYDesplazamiento;

}

public void limpiaValoresDePertenenciaYEtiquetas() {

    valoresDePertenenciaVelocidadYDesplazamiento.clear();
    etiquetasVelocidadYDesplazamiento.clear();

}

public void configuraDifusor() {

    int numeroDeFunciones = 5;
    double rangoFuncionVelocidad = (Math.abs(limiteInferiorVariableVelocidad)+Math.abs(
                                       limiteSuperiorVariableVelocidad))/numeroDeFunciones;
    double rangoFuncionDesplazamiento = (Math.abs(limiteInferiorVariableDesplazamiento)
                          +Math.abs(limiteSuperiorVariableDesplazamiento))/numeroDeFunciones;

    FuncionDePertenenciaEntrada a1 = new
    FuncionTrapezoidalEntradaVelocidad("Trapezoidal","Muy Rápido
    Negativo",limiteInferiorVariableVelocidad,
    limiteInferiorVariableVelocidad+rangoFuncionVelocidad);
    a1.calculaCoordenadas();
    a1.construyeArreglosParaGrafica();
```

```
FuncionDePertenenciaEntrada b1 = new
FuncionGaussianaEntradaVelocidad("Gaussiana","Rápido
Negativo",limiteInferiorVariableVelocidad+rangoFuncionVelocidad,
limiteInferiorVariableVelocidad+rangoFuncionVelocidad*2);
b1.calculaCoordenadas();
b1.construyeArreglosParaGrafica();

FuncionDePertenenciaEntrada c1 = new
FuncionTriangularEntradaVelocidad("Triangular","Leve",limiteInferiorVariableVelocidad+
rangoFuncionVelocidad*2,limiteInferiorVariableVelocidad+rangoFuncionVelocidad*3);
c1.calculaCoordenadas();
c1.construyeArreglosParaGrafica();

FuncionDePertenenciaEntrada d1 = new
FuncionGaussianaEntradaVelocidad("Gaussiana","Rápido
Positivo",limiteInferiorVariableVelocidad+rangoFuncionVelocidad*3,
limiteInferiorVariableVelocidad+rangoFuncionVelocidad*4);
d1.calculaCoordenadas();
d1.construyeArreglosParaGrafica();

FuncionDePertenenciaEntrada e1 = new
FuncionTrapezoidalEntradaVelocidad("Trapezoidal","Muy Rápido
Positivo",limiteInferiorVariableVelocidad+rangoFuncionVelocidad*4,
limiteInferiorVariableVelocidad+rangoFuncionVelocidad*5);
e1.calculaCoordenadas();
e1.construyeArreglosParaGrafica();

FuncionDePertenenciaEntrada a2 = new
FuncionTrapezoidalEntradaDesplazamiento("Trapezoidal","Mucho Movimiento
Negativo",limiteInferiorVariableDesplazamiento,limiteInferiorVariableDesplazamiento+
rangoFuncionDesplazamiento);
a2.calculaCoordenadas();
a2.construyeArreglosParaGrafica();

FuncionDePertenenciaEntrada b2 = new
FuncionGaussianaEntradaDesplazamiento("Gaussiana","Poco Movimiento
Negativo",limiteInferiorVariableDesplazamiento+rangoFuncionDesplazamiento,
limiteInferiorVariableDesplazamiento+rangoFuncionDesplazamiento*2);
b2.calculaCoordenadas();
b2.construyeArreglosParaGrafica();

FuncionDePertenenciaEntrada c2 = new
FuncionTriangularEntradaDesplazamiento("Triangular","Estático",
limiteInferiorVariableDesplazamiento+rangoFuncionDesplazamiento*2,
limiteInferiorVariableDesplazamiento+rangoFuncionDesplazamiento*3);
c2.calculaCoordenadas();
c2.construyeArreglosParaGrafica();

FuncionDePertenenciaEntrada d2 = new
FuncionGaussianaEntradaDesplazamiento("Gaussiana","Poco Movimiento Positivo",
limiteInferiorVariableDesplazamiento+rangoFuncionDesplazamiento*3,
limiteInferiorVariableDesplazamiento+rangoFuncionDesplazamiento*4);
d2.calculaCoordenadas();
d2.construyeArreglosParaGrafica();

FuncionDePertenenciaEntrada e2 = new
FuncionTrapezoidalEntradaDesplazamiento("Trapezoidal","Mucho Movimiento Positivo",
limiteInferiorVariableDesplazamiento+rangoFuncionDesplazamiento*4,
limiteInferiorVariableDesplazamiento+rangoFuncionDesplazamiento*5);
e2.calculaCoordenadas();
e2.construyeArreglosParaGrafica();

universoDelDiscursoVariableVelocidad.add(a1);
universoDelDiscursoVariableDesplazamiento.add(a2);
universoDelDiscursoVariableVelocidad.add(b1);
universoDelDiscursoVariableDesplazamiento.add(b2);
```

```
    universoDelDiscursoVariableVelocidad.add(c1);
    universoDelDiscursoVariableDesplazamiento.add(c2);
    universoDelDiscursoVariableVelocidad.add(d1);
    universoDelDiscursoVariableDesplazamiento.add(d2);
    universoDelDiscursoVariableVelocidad.add(e1);
    universoDelDiscursoVariableDesplazamiento.add(e2);

}

public void calculaValorDePertenenciaYConjuntoDifuso (double velocidad, double
                                                                        desplazamiento) {

    double [] conjuntoDeLimitesVelocidad = new double
                                            [universoDelDiscursoVariableVelocidad.size()*2];
    double [] conjuntoDeLimitesDesplazamiento = new double
                                        [universoDelDiscursoVariableDesplazamiento.size()*2];

    for(int h=0; h<universoDelDiscursoVariableVelocidad.size(); h++) {

        conjuntoDeLimitesVelocidad[h*2] = universoDelDiscursoVariableVelocidad.get(h).
                                                                  coordenadas.get(0);
        conjuntoDeLimitesDesplazamiento[h*2] = universoDelDiscursoVariableDesplazamiento.
                                                          get(h).coordenadas.get(0);
        conjuntoDeLimitesVelocidad[h*2+1] = universoDelDiscursoVariableVelocidad.get(h).
                                        coordenadas.get(universoDelDiscursoVariableVelocidad.
                                        get(h).coordenadas.size()-1);
        conjuntoDeLimitesDesplazamiento[h*2+1] =
                            universoDelDiscursoVariableDesplazamiento.get(h).coordenadas.get(
                            universoDelDiscursoVariableDesplazamiento.get(h).
                            coordenadas.size()-1);

    }

    if(velocidad >= conjuntoDeLimitesVelocidad[conjuntoDeLimitesVelocidad.length-3]) {

        valoresDePertenenciaVelocidadYDesplazamiento.add(
        universoDelDiscursoVariableVelocidad.get(
        universoDelDiscursoVariableVelocidad.size()-1).
        calculaValorDePertenencia(velocidad));
        etiquetasVelocidadYDesplazamiento.add(
        universoDelDiscursoVariableVelocidad.get(
        universoDelDiscursoVariableVelocidad.size()-1).
        getEtiqueta());

    } else {

        for(int h=0; h<universoDelDiscursoVariableVelocidad.size()-1; h++) {

            if(velocidad>conjuntoDeLimitesVelocidad[h*2] &&
                 velocidad<conjuntoDeLimitesVelocidad[h*2+1]) {
                if(velocidad>conjuntoDeLimitesVelocidad[h*2+2]){

                    calculaValorDePertenenciaConDosConjuntosVelocidad(velocidad,h,h+1);
                    break;

                }

            valoresDePertenenciaVelocidadYDesplazamiento.add(
            universoDelDiscursoVariableVelocidad.get(h).
            calculaValorDePertenencia(velocidad));
            etiquetasVelocidadYDesplazamiento.add(
            universoDelDiscursoVariableVelocidad.get(h).
            getEtiqueta());
            break;

            }
```

```
        }

    }

    if(desplazamiento >= conjuntoDeLimitesDesplazamiento
          [conjuntoDeLimitesDesplazamiento.length-3]) {

        valoresDePertenenciaVelocidadYDesplazamiento.add(
        universoDelDiscursoVariableDesplazamiento.get(
        universoDelDiscursoVariableDesplazamiento.size()-1).
        calculaValorDePertenencia(desplazamiento));
        etiquetasVelocidadYDesplazamiento.add(
        universoDelDiscursoVariableDesplazamiento.get(
        universoDelDiscursoVariableDesplazamiento.size()-1).
        getEtiqueta());

    } else {

        for(int h=0; h<universoDelDiscursoVariableDesplazamiento.size()-1; h++) {

            if(desplazamiento>conjuntoDeLimitesDesplazamiento[h*2] &&
                        desplazamiento<conjuntoDeLimitesDesplazamiento[h*2+1]) {

                if(desplazamiento>conjuntoDeLimitesDesplazamiento[h*2+2]){

                    calculaValorDePertenenciaConDosConjuntosDesplazamiento(
                                                    desplazamiento,h,h+1);
                    break;

                }

            valoresDePertenenciaVelocidadYDesplazamiento.add(
            universoDelDiscursoVariableDesplazamiento.get(h).
            calculaValorDePertenencia(desplazamiento));
            etiquetasVelocidadYDesplazamiento.add(
            universoDelDiscursoVariableDesplazamiento.get(h).
            getEtiqueta());
            break;

            }

        }

    }

}

private void calculaValorDePertenenciaConDosConjuntosVelocidad(double valorDeEntrada, int
                                                                funcion1, int funcion2) {

    double valorFuncion1 = universoDelDiscursoVariableVelocidad.get(
                            funcion1).calculaValorDePertenencia(valorDeEntrada);
    double valorFuncion2 = universoDelDiscursoVariableVelocidad.get(
                            funcion2).calculaValorDePertenencia(valorDeEntrada);

    if(valorFuncion1 > valorFuncion2) {

        valoresDePertenenciaVelocidadYDesplazamiento.add(valorFuncion1);
        etiquetasVelocidadYDesplazamiento.add(
                        universoDelDiscursoVariableVelocidad.get(funcion1).getEtiqueta());

    } else {

        valoresDePertenenciaVelocidadYDesplazamiento.add(valorFuncion2);
        etiquetasVelocidadYDesplazamiento.add(
                        universoDelDiscursoVariableVelocidad.get(funcion2).getEtiqueta());
```

```
        }

    }

    private void calculaValorDePertenenciaConDosConjuntosDesplazamiento(double valorDeEntrada,
                                                                        int funcion1, int funcion2) {

        double valorFuncion1 = universoDelDiscursoVariableDesplazamiento.get(
                                            funcion1).calculaValorDePertenencia(valorDeEntrada);
        double valorFuncion2 = universoDelDiscursoVariableDesplazamiento.get(
                                            funcion2).calculaValorDePertenencia(valorDeEntrada);

        if(valorFuncion1 > valorFuncion2) {

            valoresDePertenenciaVelocidadYDesplazamiento.add(valorFuncion1);
            etiquetasVelocidadYDesplazamiento.add(
                        universoDelDiscursoVariableDesplazamiento.get(funcion1).getEtiqueta());


        } else {

            valoresDePertenenciaVelocidadYDesplazamiento.add(valorFuncion2);
            etiquetasVelocidadYDesplazamiento.add(
                        universoDelDiscursoVariableDesplazamiento.get(funcion2).getEtiqueta());

        }

    }

}
```

FuncionDePertenenciaEntrada.java

```
import java.util.ArrayList;

public abstract class FuncionDePertenenciaEntrada implements InterfazFuncionDePertenenciaEntrada
{

    protected String tipo;
    protected String etiqueta;
    protected double rangoInferior;
    protected double rangoSuperior;
    protected ArrayList<Double> coordenadas = new ArrayList<>();
    protected ArrayList<double []> arreglosGrafica = new ArrayList<>();

    public FuncionDePertenenciaEntrada() {

    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public String getEtiqueta() {
        return etiqueta;
    }

    public void setEtiqueta(String etiqueta) {
        this.etiqueta = etiqueta;
    }
```

```
    public double getRangoInferior() {
        return rangoInferior;
    }

    public void setRangoInferior(double rangoInferior) {
        this.rangoInferior = rangoInferior;
    }

    public double getRangoSuperior() {
        return rangoSuperior;
    }

    public void setRangoSuperior(double rangoSuperior) {
        this.rangoSuperior = rangoSuperior;
    }

    public ArrayList<Double> getCoordenadas() {
        return coordenadas;
    }

    public void setCoordenadas(ArrayList<Double> coordenadas) {
        this.coordenadas = coordenadas;
    }

    public ArrayList<double[]> getArreglosGrafica() {
        return arreglosGrafica;
    }

    public void setArreglosGrafica(ArrayList<double[]> arreglosGrafica) {
        this.arreglosGrafica = arreglosGrafica;
    }

    @Override
    public void calculaCoordenadas() {

    }

    @Override
    public double calculaValorDePertenencia(double valorReal) {

        return 0;

    }

    @Override
    public void construyeArreglosParaGrafica() {

    }

}
```

InterfazFuncionDePertenenciaEntrada.java

```
public interface InterfazFuncionDePertenenciaEntrada {

    void calculaCoordenadas();

    double calculaValorDePertenencia(double valorReal);

    void construyeArreglosParaGrafica();

}
```

FuncionTrapezoidalEntradaVelocidad.java

```
import controladorDifuso.difusor.FuncionDePertenenciaEntrada;

public class FuncionTrapezoidalEntradaVelocidad extends FuncionDePertenenciaEntrada {

    private double a;
    private double b;
    private double c;
    private double d;

    public FuncionTrapezoidalEntradaVelocidad() {

    }

    public FuncionTrapezoidalEntradaVelocidad(String tipo, String etiqueta, double rangoInferior,
                                                                            double rangoSuperior) {

        this.tipo = tipo;
        this.etiqueta = etiqueta;
        this.rangoInferior = rangoInferior;
        this.rangoSuperior = rangoSuperior;

    }

    @Override
    public void calculaCoordenadas() {

        a = this.rangoInferior-10;
        b = this.rangoInferior;
        c = this.rangoSuperior;
        d = this.rangoSuperior+10;

        this.coordenadas.add(a); this.coordenadas.add(b);
        this.coordenadas.add(c); this.coordenadas.add(d);

    }

    @Override
    public double calculaValorDePertenencia(double valorReal) {

        double xa = Math.abs(valorReal)-Math.abs(a);
        double ba = Math.abs(b)-Math.abs(a);
        double dx = Math.abs(d)-Math.abs(valorReal);
        double dc = Math.abs(d)-Math.abs(c);
        double div1 = xa/ba;
        double div2 = dx/dc;
        double min1 = Math.min(div1, 1.0);
        double min2 = Math.min(min1, div2);
        double max = Math.max(min2, 0.0);
        double pertenencia = max;

        return pertenencia;

    }

    @Override
    public void construyeArreglosParaGrafica() {

        double dimensionBase = 0.0;

        if(a<0.0 && d<0.0) {

            dimensionBase = Math.abs(a)-Math.abs(d);

        }
```

```
        if(a<0.0 && d>0.0) {

            dimensionBase += Math.abs(a);
            dimensionBase += d;

        }

        dimensionBase = d-a;
        int muestras = 50;
        int m = muestras -1;
        double delta = dimensionBase/m;
        double [] valoresX = new double [muestras];
        double [] valoresY = new double [muestras];
        double auxX = 0.0;

        for(int h=0; h<muestras; h++) {

            double aux = a+auxX;
            valoresX[h] = aux;
            valoresY[h] = calculaValorDePertenencia(aux);
            auxX += delta;

        }

        this.arreglosGrafica.add(valoresX);
        this.arreglosGrafica.add(valoresY);

    }

}
```

FuncionTrapezoidalEntradaDesplazamiento.java

```
import controladorDifuso.difusor.FuncionDePertenenciaEntrada;

public class FuncionTrapezoidalEntradaDesplazamiento extends FuncionDePertenenciaEntrada {

    private double a;
    private double b;
    private double c;
    private double d;

    public FuncionTrapezoidalEntradaDesplazamiento() {

    }

    public FuncionTrapezoidalEntradaDesplazamiento(String tipo, String etiqueta, double
                                                            rangoInferior, double rangoSuperior) {

        this.tipo = tipo;
        this.etiqueta = etiqueta;
        this.rangoInferior = rangoInferior;
        this.rangoSuperior = rangoSuperior;

    }

    @Override
    public void calculaCoordenadas() {

        a = this.rangoInferior-0.125;
        b = this.rangoInferior;
        c = this.rangoSuperior;
        d = this.rangoSuperior+0.125;

        this.coordenadas.add(a); this.coordenadas.add(b);
        this.coordenadas.add(c); this.coordenadas.add(d);
```

```
    }

    @Override
    public double calculaValorDePertenencia(double valorReal) {

        double xa = Math.abs(valorReal)-Math.abs(a);
        double ba = Math.abs(b)-Math.abs(a);
        double dx = Math.abs(d)-Math.abs(valorReal);
        double dc = Math.abs(d)-Math.abs(c);
        double div1 = xa/ba;
        double div2 = dx/dc;
        double min1 = Math.min(div1, 1.0);
        double min2 = Math.min(min1, div2);
        double max = Math.max(min2, 0.0);
        double pertenencia = max;

        return pertenencia;

    }

    @Override
    public void construyeArreglosParaGrafica() {

        double dimensionBase = 0.0;

        if(a<0.0 && d<0.0) {

            dimensionBase = Math.abs(a)-Math.abs(d);

        }

        if(a<0.0 && d>0.0) {

            dimensionBase += Math.abs(a)+d;

        }

        dimensionBase = d-a;
        int muestras = 50;
        int m = muestras -1;
        double delta = dimensionBase/m;
        double [] valoresX = new double [muestras];
        double [] valoresY = new double [muestras];
        double auxX = 0.0;

        for(int h=0; h<muestras; h++) {

            double aux = a+auxX;
            valoresX[h] = aux;
            valoresY[h] = calculaValorDePertenencia(aux);
            auxX += delta;

        }

        this.arreglosGrafica.add(valoresX);
        this.arreglosGrafica.add(valoresY);

    }

}
```

FuncionGaussianaEntradaVelocidad.java

```
import controladorDifuso.difusor.FuncionDePertenenciaEntrada;

public class FuncionGaussianaEntradaVelocidad extends FuncionDePertenenciaEntrada {

    private double inferior;
    private double superior;
    private double centro;
    private final double sigma = 4.65;

    public FuncionGaussianaEntradaVelocidad() {

    }

    public FuncionGaussianaEntradaVelocidad(String tipo, String etiqueta, double rangoInferior,
                                                                          double rangoSuperior) {

        this.tipo = tipo;
        this.etiqueta = etiqueta;
        this.rangoInferior = rangoInferior;
        this.rangoSuperior = rangoSuperior;

    }

    @Override
    public void calculaCoordenadas() {

        inferior = this.rangoInferior-10;
        superior = this.rangoSuperior+10;

        if (Math.abs(Math.abs(inferior)-Math.abs(superior))/2 == 0.0) {

            centro = 0.0;

        }

        if(inferior<0.0 && superior<0.0) {

            centro = (Math.abs(inferior)-Math.abs(superior))/2;

        }

        if(inferior<0.0 && superior>0.0) {

            centro = (Math.abs(inferior)+superior)/2;

        }

        centro = (inferior+superior)/2;

        this.coordenadas.add(inferior); this.coordenadas.add(centro);
        this.coordenadas.add(sigma); this.coordenadas.add(superior);

    }

    @Override
    public double calculaValorDePertenencia(double valorReal) {

        double xc = valorReal-centro;
        double xcs = xc/sigma;
        double xcs2 = Math.pow(xcs, 2);
        double mxcs2 = -0.5*xcs2;
        double pertenencia = Math.pow(Math.E, mxcs2);

        return pertenencia;
```

```
    }

    @Override
    public void construyeArreglosParaGrafica() {

        int muestras = 50;
        int m = muestras -1;
        double delta = (Math.abs(inferior-superior))/m;
        double [] valoresX = new double [muestras];
        double [] valoresY = new double [muestras];
        double auxX = 0.0;

        for(int h=0; h<muestras; h++) {

            double aux = inferior+auxX;
            valoresX[h] = aux;
            valoresY[h] = calculaValorDePertenencia(aux);
            auxX += delta;

        }

        this.arreglosGrafica.add(valoresX);
        this.arreglosGrafica.add(valoresY);

    }

}
```

FuncionGaussianaEntradaDesplazamiento.java

```
import controladorDifuso.difusor.FuncionDePertenenciaEntrada;

public class FuncionGaussianaEntradaDesplazamiento extends FuncionDePertenenciaEntrada {

    private double inferior;
    private double superior;
    private double centro;
    private final double sigma = 0.1;

    public FuncionGaussianaEntradaDesplazamiento() {

    }

    public FuncionGaussianaEntradaDesplazamiento(String tipo, String etiqueta, double
                                                        rangoInferior, double rangoSuperior) {

        this.tipo = tipo;
        this.etiqueta = etiqueta;
        this.rangoInferior = rangoInferior;
        this.rangoSuperior = rangoSuperior;

    }

    @Override
    public void calculaCoordenadas() {

        inferior = this.rangoInferior-0.125;
        superior = this.rangoSuperior+0.125;

        if (Math.abs(Math.abs(inferior)-Math.abs(superior))/2 == 0.0) {

            centro = 0.0;

        }
```

```
        if(inferior<0.0 && superior<0.0) {

            centro = (Math.abs(inferior)-Math.abs(superior))/2;

        }

        if(inferior<0.0 && superior>0.0) {

            centro = (Math.abs(inferior)+superior)/2;

        }

        centro = (inferior+superior)/2;

        this.coordenadas.add(inferior); this.coordenadas.add(centro);
        this.coordenadas.add(sigma); this.coordenadas.add(superior);

    }

    @Override
    public double calculaValorDePertenencia(double valorReal) {

        double xc = valorReal-centro;
        double xcs = xc/sigma;
        double xcs2 = Math.pow(xcs, 2);
        double mxcs2 = -0.5*xcs2;
        double pertenencia = Math.pow(Math.E, mxcs2);

        return pertenencia;

    }

    @Override
    public void construyeArreglosParaGrafica() {

        int muestras = 50;
        int m = muestras -1;
        double delta = (Math.abs(inferior-superior))/m;
        double [] valoresX = new double [muestras];
        double [] valoresY = new double [muestras];
        double auxX = 0.0;

        for(int h=0; h<muestras; h++) {

            double aux = inferior+auxX;
            valoresX[h] = aux;
            valoresY[h] = calculaValorDePertenencia(aux);
            auxX += delta;

        }

        this.arreglosGrafica.add(valoresX);
        this.arreglosGrafica.add(valoresY);

    }

}
```

FuncionTriangularEntradaVelocidad.java

```
import controladorDifuso.difusor.FuncionDePertenenciaEntrada;

public class FuncionTriangularEntradaVelocidad extends FuncionDePertenenciaEntrada {

    private double a;
    private double b;
    private double c;

    public FuncionTriangularEntradaVelocidad() {

    }

    public FuncionTriangularEntradaVelocidad(String tipo, String etiqueta, double rangoInferior,
                                                                            double rangoSuperior) {

        this.tipo = tipo;
        this.etiqueta = etiqueta;
        this.rangoInferior = rangoInferior;
        this.rangoSuperior = rangoSuperior;

    }

    @Override
    public void calculaCoordenadas() {

        a = this.rangoInferior-10;
        c = this.rangoSuperior+10;

        if (Math.abs(Math.abs(a)-Math.abs(c))/2 == 0.0) {

            b = 0.0;

        }

        if( a<0.0 && c<0.0) {

            double aux = Math.abs(Math.abs(a)-Math.abs(c))/2;
            b = a +aux;

        }

        if(a<0.0 && c>0.0) {

            double aux = (Math.abs(a)+c)/2;
            b = a + aux;

        } else {

            b = (a+c)/2;

        }

        this.coordenadas.add(a); this.coordenadas.add(b); this.coordenadas.add(c);

    }

    @Override
    public double calculaValorDePertenencia(double valorReal) {

        double xa = Math.abs(valorReal)-Math.abs(a);
        double ba = Math.abs(b)-Math.abs(a);
        double cx = Math.abs(c)-Math.abs(valorReal);
        double cb = Math.abs(c)-Math.abs(b);
        double div1 = xa/ba;
```

```
        double div2 = cx/cb;
        double min = Math.min(div1, div2);
        double max = Math.max(min, 0.0);
        double pertenencia = max;

        return pertenencia;

    }

    @Override
    public void construyeArreglosParaGrafica() {

        int muestras = 50;
        int m = muestras -1;
        double delta = (Math.abs(b-c)*2)/m;
        double [] valoresX = new double [muestras];
        double [] valoresY = new double [muestras];
        double auxX = 0.0;

        for(int h=0; h<muestras; h++) {

            double aux = a+auxX;
            valoresX[h] = aux;
            valoresY[h] = calculaValorDePertenencia(aux);
            auxX += delta;
        }

        this.arreglosGrafica.add(valoresX);
        this.arreglosGrafica.add(valoresY);

    }

}
```

FuncionTriangularEntradaDesplazamiento.java

```
import controladorDifuso.difusor.FuncionDePertenenciaEntrada;

public class FuncionTriangularEntradaDesplazamiento extends FuncionDePertenenciaEntrada {

    private double a;
    private double b;
    private double c;

    public FuncionTriangularEntradaDesplazamiento() {

    }

    public FuncionTriangularEntradaDesplazamiento(String tipo, String etiqueta, double
                                                        rangoInferior, double rangoSuperior) {

        this.tipo = tipo;
        this.etiqueta = etiqueta;
        this.rangoInferior = rangoInferior;
        this.rangoSuperior = rangoSuperior;

    }

    @Override
    public void calculaCoordenadas() {

        a = this.rangoInferior-0.125;
        c = this.rangoSuperior+0.125;

        if (Math.abs(Math.abs(a)-Math.abs(c))/2 == 0.0) {
```

```
            b = 0.0;

        }

        if( a<0.0 && c<0.0) {

            double aux = Math.abs(Math.abs(a)-Math.abs(c))/2;
            b = a +aux;

        }

        if(a<0.0 && c>0.0) {

            double aux = (Math.abs(a)+c)/2;
            b = a + aux;

        } else {

            b = (a+c)/2;

        }

        this.coordenadas.add(a); this.coordenadas.add(b); this.coordenadas.add(c);

    }

    @Override
    public double calculaValorDePertenencia(double valorReal) {

        double xa = Math.abs(valorReal)-Math.abs(a);
        double ba = Math.abs(b)-Math.abs(a);
        double cx = Math.abs(c)-Math.abs(valorReal);
        double cb = Math.abs(c)-Math.abs(b);
        double div1 = xa/ba;
        double div2 = cx/cb;
        double min = Math.min(div1, div2);
        double max = Math.max(min, 0.0);
        double pertenencia = max;

        return pertenencia;

    }

    @Override
    public void construyeArreglosParaGrafica() {

        int muestras = 50;
        int m = muestras -1;
        double delta = (Math.abs(b-c)*2)/m;
        double [] valoresX = new double [muestras];
        double [] valoresY = new double [muestras];
        double auxX = 0.0;

        for(int h=0; h<muestras; h++) {

            double aux = a+auxX;
            valoresX[h] = aux;
            valoresY[h] = calculaValorDePertenencia(aux);
            auxX += delta;
        }

        this.arreglosGrafica.add(valoresX);
        this.arreglosGrafica.add(valoresY);

    }

}
```

MaquinaDeInferencia.java

```
import java.util.ArrayList;

public class MaquinaDeInferencia {

    private BaseDeConocimiento base = new BaseDeConocimiento();
    private ArrayList<Double> valoresDeInferenciaSalida = new ArrayList<>();
    private ArrayList<String> etiquetasDeInferenciaSalida = new ArrayList<>();

    public MaquinaDeInferencia() {

    }

    public BaseDeConocimiento getBase() {
        return base;
    }

    public void setBase(BaseDeConocimiento base) {
        this.base = base;
    }

    public ArrayList<Double> getValoresDeInferenciaSalida() {
        return valoresDeInferenciaSalida;
    }

    public void setValoresDeInferenciaSalida(ArrayList<Double> valoresDeInferenciaSalida) {
        this.valoresDeInferenciaSalida = valoresDeInferenciaSalida;
    }

    public ArrayList<String> getEtiquetasDeInferenciaSalida() {
        return etiquetasDeInferenciaSalida;
    }

    public void setEtiquetasDeInferenciaSalida(ArrayList<String> etiquetasDeInferenciaSalida) {
        this.etiquetasDeInferenciaSalida = etiquetasDeInferenciaSalida;
    }

    public void limpiaValoresDeInferenciaYEtiquetasDeInferencia() {

        valoresDeInferenciaSalida.clear();
        etiquetasDeInferenciaSalida.clear();

    }

    public void realizaInferencia(ArrayList<Double> valoresDePertenencia,
                                                                    ArrayList<String> etiquetas) {

        String [] etiquetasInferencia = obtenConsecuentes(etiquetas);

        for(int h=0; h<etiquetasInferencia.length; h++) {

            valoresDeInferenciaSalida.add(valoresDePertenencia.get(h));
            etiquetasDeInferenciaSalida.add(etiquetasInferencia[h]);

        }

    }

    private String [] obtenConsecuentes(ArrayList<String> etiquetas) {

        ArrayList<String> preEtiquetasInferencia = new ArrayList<>();
        boolean bandera = false;
        int contador = 0;
```

```
        while(bandera == false) {

                if(etiquetas.get(0).equals(base.getConjuntoDeReglas().get(
                     contador).getAntecedentes().get(0)) &&
                   etiquetas.get(1).equals(base.getConjuntoDeReglas().get(
                     contador).getAntecedentes().get(1))) {

                    preEtiquetasInferencia = base.getConjuntoDeReglas().get(
                                                contador).getConsecuentes();
                    bandera = true;

                } else {

                    contador++;

                }

        }

        String [] etiquetasInferencia = preEtiquetasInferencia.toArray(
                                        new String[preEtiquetasInferencia.size()]);

        return etiquetasInferencia;

    }

}
```

BaseDeConocimiento.java

```
import java.util.ArrayList;

public class BaseDeConocimiento {

    private ArrayList<Regla> conjuntoDeReglas = new ArrayList<>();

    public BaseDeConocimiento(){

    }

    public ArrayList<Regla> getConjuntoDeReglas() {
        return conjuntoDeReglas;
    }

    public void setConjuntoDeReglas(ArrayList<Regla> conjuntoDeReglas) {
        this.conjuntoDeReglas = conjuntoDeReglas;
    }

    public void construyeBaseDeConocimiento(){

        ArrayList<String> antecedentesRegla1 = new ArrayList<>();
        antecedentesRegla1.add("Muy Rápido Negativo");
        antecedentesRegla1.add("Mucho Movimiento Negativo");
        ArrayList<String> consecuentesRegla1 = new ArrayList<>();
        consecuentesRegla1.add("Nulo");
        Regla regla1 = new Regla(antecedentesRegla1, consecuentesRegla1);

        ArrayList<String> antecedentesRegla2 = new ArrayList<>();
        antecedentesRegla2.add("Muy Rápido Negativo");
        antecedentesRegla2.add("Poco Movimiento Negativo");
        ArrayList<String> consecuentesRegla2 = new ArrayList<>();
        consecuentesRegla2.add("Nulo");
        Regla regla2 = new Regla(antecedentesRegla2, consecuentesRegla2);
```

```
ArrayList<String> antecedentesRegla3 = new ArrayList<>();
antecedentesRegla3.add("Muy Rápido Negativo");
antecedentesRegla3.add("Estático");
ArrayList<String> consecuentesRegla3 = new ArrayList<>();
consecuentesRegla3.add("Muy Poco");
Regla regla3 = new Regla(antecedentesRegla3, consecuentesRegla3);

ArrayList<String> antecedentesRegla4 = new ArrayList<>();
antecedentesRegla4.add("Muy Rápido Negativo");
antecedentesRegla4.add("Poco Movimiento Positivo");
ArrayList<String> consecuentesRegla4 = new ArrayList<>();
consecuentesRegla4.add("Muy Poco");
Regla regla4 = new Regla(antecedentesRegla4, consecuentesRegla4);

ArrayList<String> antecedentesRegla5 = new ArrayList<>();
antecedentesRegla5.add("Muy Rápido Negativo");
antecedentesRegla5.add("Mucho Movimiento Positivo");
ArrayList<String> consecuentesRegla5 = new ArrayList<>();
consecuentesRegla5.add("Mucho");
Regla regla5 = new Regla(antecedentesRegla5, consecuentesRegla5);

ArrayList<String> antecedentesRegla6 = new ArrayList<>();
antecedentesRegla6.add("Rápido Negativo");
antecedentesRegla6.add("Mucho Movimiento Negativo");
ArrayList<String> consecuentesRegla6 = new ArrayList<>();
consecuentesRegla6.add("Nulo");
Regla regla6 = new Regla(antecedentesRegla6, consecuentesRegla6);

ArrayList<String> antecedentesRegla7 = new ArrayList<>();
antecedentesRegla7.add("Rápido Negativo");
antecedentesRegla7.add("Poco Movimiento Negativo");
ArrayList<String> consecuentesRegla7 = new ArrayList<>();
consecuentesRegla7.add("Nulo");
Regla regla7 = new Regla(antecedentesRegla7, consecuentesRegla7);

ArrayList<String> antecedentesRegla8 = new ArrayList<>();
antecedentesRegla8.add("Rápido Negativo"); antecedentesRegla8.add("Estático");
ArrayList<String> consecuentesRegla8 = new ArrayList<>();
consecuentesRegla8.add("Poco"); consecuentesRegla8.add("Muy Poco");
Regla regla8 = new Regla(antecedentesRegla8, consecuentesRegla8);

ArrayList<String> antecedentesRegla9 = new ArrayList<>();
antecedentesRegla9.add("Rápido Negativo");
antecedentesRegla9.add("Poco Movimiento Positivo");
ArrayList<String> consecuentesRegla9 = new ArrayList<>();
consecuentesRegla9.add("Muy Poco");
Regla regla9 = new Regla(antecedentesRegla9, consecuentesRegla9);

ArrayList<String> antecedentesRegla10 = new ArrayList<>();
antecedentesRegla10.add("Rápido Negativo");
antecedentesRegla10.add("Mucho Movimiento Positivo");
ArrayList<String> consecuentesRegla10 = new ArrayList<>();
consecuentesRegla10.add("Poco"); consecuentesRegla10.add("Muy Poco");
Regla regla10 = new Regla(antecedentesRegla10, consecuentesRegla10);

ArrayList<String> antecedentesRegla11 = new ArrayList<>();
antecedentesRegla11.add("Leve"); antecedentesRegla11.add("Mucho Movimiento Negativo");
ArrayList<String> consecuentesRegla11 = new ArrayList<>();
consecuentesRegla11.add("Muy Poco"); consecuentesRegla11.add("Poco");
Regla regla11 = new Regla(antecedentesRegla11, consecuentesRegla11);

ArrayList<String> antecedentesRegla12 = new ArrayList<>();
antecedentesRegla12.add("Leve"); antecedentesRegla12.add("Poco Movimiento Negativo");
ArrayList<String> consecuentesRegla12 = new ArrayList<>();
consecuentesRegla12.add("Poco"); consecuentesRegla12.add("Muy Poco");
Regla regla12 = new Regla(antecedentesRegla12, consecuentesRegla12);
```

```
ArrayList<String> antecedentesRegla13 = new ArrayList<>();
antecedentesRegla13.add("Leve"); antecedentesRegla13.add("Estático");
ArrayList<String> consecuentesRegla13 = new ArrayList<>();
consecuentesRegla13.add("Moderado");
Regla regla13 = new Regla(antecedentesRegla13, consecuentesRegla13);

ArrayList<String> antecedentesRegla14 = new ArrayList<>();
antecedentesRegla14.add("Leve"); antecedentesRegla14.add("Poco Movimiento Positivo");
ArrayList<String> consecuentesRegla14 = new ArrayList<>();
consecuentesRegla14.add("Moderado"); consecuentesRegla14.add("Poco");
Regla regla14 = new Regla(antecedentesRegla14, consecuentesRegla14);

ArrayList<String> antecedentesRegla15 = new ArrayList<>();
antecedentesRegla15.add("Leve"); antecedentesRegla15.add("Mucho Movimiento Positivo");
ArrayList<String> consecuentesRegla15 = new ArrayList<>();
consecuentesRegla15.add("Mucho");
Regla regla15 = new Regla(antecedentesRegla15, consecuentesRegla15);

ArrayList<String> antecedentesRegla16 = new ArrayList<>();
antecedentesRegla16.add("Rápido Positivo");
antecedentesRegla16.add("Mucho Movimiento Negativo");
ArrayList<String> consecuentesRegla16 = new ArrayList<>();
consecuentesRegla16.add("Poco"); consecuentesRegla16.add("Muy Poco");
Regla regla16 = new Regla(antecedentesRegla16, consecuentesRegla16);

ArrayList<String> antecedentesRegla17 = new ArrayList<>();
antecedentesRegla17.add("Rápido Positivo");
antecedentesRegla17.add("Poco Movimiento Negativo");
ArrayList<String> consecuentesRegla17 = new ArrayList<>();
consecuentesRegla17.add("Demasiado");
Regla regla17 = new Regla(antecedentesRegla17, consecuentesRegla17);

ArrayList<String> antecedentesRegla18 = new ArrayList<>();
antecedentesRegla18.add("Rápido Positivo"); antecedentesRegla18.add("Estático");
ArrayList<String> consecuentesRegla18 = new ArrayList<>();
consecuentesRegla18.add("Poco"); consecuentesRegla18.add("Moderado");
Regla regla18 = new Regla(antecedentesRegla18, consecuentesRegla18);

ArrayList<String> antecedentesRegla19 = new ArrayList<>();
antecedentesRegla19.add("Rápido Positivo");
antecedentesRegla19.add("Poco Movimiento Positivo");
ArrayList<String> consecuentesRegla19 = new ArrayList<>();
consecuentesRegla19.add("Mucho");
Regla regla19 = new Regla(antecedentesRegla19, consecuentesRegla19);

ArrayList<String> antecedentesRegla20 = new ArrayList<>();
antecedentesRegla20.add("Rápido Positivo");
antecedentesRegla20.add("Mucho Movimiento Positivo");
ArrayList<String> consecuentesRegla20 = new ArrayList<>();
consecuentesRegla20.add("Demasiado");
Regla regla20 = new Regla(antecedentesRegla20, consecuentesRegla20);

ArrayList<String> antecedentesRegla21 = new ArrayList<>();
antecedentesRegla21.add("Muy Rápido Positivo");
antecedentesRegla21.add("Mucho Movimiento Negativo");
ArrayList<String> consecuentesRegla21 = new ArrayList<>();
consecuentesRegla21.add("Moderado");
Regla regla21 = new Regla(antecedentesRegla21, consecuentesRegla21);

ArrayList<String> antecedentesRegla22 = new ArrayList<>();
antecedentesRegla22.add("Muy Rápido Positivo");
antecedentesRegla22.add("Poco Movimiento Negativo");
ArrayList<String> consecuentesRegla22 = new ArrayList<>();
consecuentesRegla22.add("Demasiado");
Regla regla22 = new Regla(antecedentesRegla22, consecuentesRegla22);
```

```
        ArrayList<String> antecedentesRegla23 = new ArrayList<>();
        antecedentesRegla23.add("Muy Rápido Positivo"); antecedentesRegla23.add("Estático");
        ArrayList<String> consecuentesRegla23 = new ArrayList<>();
        consecuentesRegla23.add("Demasiado");
        Regla regla23 = new Regla(antecedentesRegla23, consecuentesRegla23);

        ArrayList<String> antecedentesRegla24 = new ArrayList<>();
        antecedentesRegla24.add("Muy Rápido Positivo");
        antecedentesRegla24.add("Poco Movimiento Positivo");
        ArrayList<String> consecuentesRegla24 = new ArrayList<>();
        consecuentesRegla24.add("Mucho"); consecuentesRegla24.add("Demasiado");
        Regla regla24 = new Regla(antecedentesRegla24, consecuentesRegla24);

        ArrayList<String> antecedentesRegla25 = new ArrayList<>();
        antecedentesRegla25.add("Muy Rápido Positivo");
        antecedentesRegla25.add("Mucho Movimiento Positivo");
        ArrayList<String> consecuentesRegla25 = new ArrayList<>();
        consecuentesRegla25.add("Demasiado"); consecuentesRegla25.add("Mucho");
        Regla regla25 = new Regla(antecedentesRegla25, consecuentesRegla25);


        conjuntoDeReglas.add(regla1); conjuntoDeReglas.add(regla2); conjuntoDeReglas.add(regla3);
        conjuntoDeReglas.add(regla4); conjuntoDeReglas.add(regla5); conjuntoDeReglas.add(regla6);
        conjuntoDeReglas.add(regla7); conjuntoDeReglas.add(regla8); conjuntoDeReglas.add(regla9);
        conjuntoDeReglas.add(regla10);conjuntoDeReglas.add(regla11);
        conjuntoDeReglas.add(regla12); conjuntoDeReglas.add(regla13);
        conjuntoDeReglas.add(regla14); conjuntoDeReglas.add(regla15);
        conjuntoDeReglas.add(regla16); conjuntoDeReglas.add(regla17);
        conjuntoDeReglas.add(regla18); conjuntoDeReglas.add(regla19);
        conjuntoDeReglas.add(regla20); conjuntoDeReglas.add(regla21);
        conjuntoDeReglas.add(regla22); conjuntoDeReglas.add(regla23);
        conjuntoDeReglas.add(regla24); conjuntoDeReglas.add(regla25);


    }

}
```

Regla.java

```
public class Regla {

    private ArrayList<String> antecedentes = new ArrayList<>();
    private ArrayList<String> consecuentes = new ArrayList<>();

    public Regla() {

    }

    public Regla(ArrayList<String> antecedentes, ArrayList<String> consecuentes) {

        this.antecedentes = antecedentes;
        this.consecuentes = consecuentes;

    }

    public ArrayList<String> getAntecedentes() {
        return antecedentes;
    }

    public void setAntecedentes(ArrayList<String> antecedentes) {
        this.antecedentes = antecedentes;
    }
    public ArrayList<String> getConsecuentes() {
        return consecuentes;
    }
```

```
    public void setConsecuentes(ArrayList<String> consecuentes) {
        this.consecuentes = consecuentes;
    }

}
```

Desdifusor.java

```
mport controladorDifuso.desdifusor.funciones.FuncionGaussianaSalidaVoltaje;
import controladorDifuso.desdifusor.funciones.FuncionSingletoneSalidaVoltaje;
import controladorDifuso.desdifusor.funciones.FuncionTrapezoidalSalidaVoltaje;
import controladorDifuso.desdifusor.funciones.FuncionTriangularSalidaVoltaje;
import java.util.ArrayList;

public class Desdifusor {

    private ArrayList<FuncionDePertenenciaSalida> universoDelDiscursoVariableVoltaje =
                                                                                new ArrayList<>();
    private double limiteInferior;
    private double limiteSuperior;
    private double valorDeSalida;

    public Desdifusor() {

    }

    public Desdifusor(double limiteInferior, double limiteSuperior) {

        this.limiteInferior = limiteInferior;
        this.limiteSuperior = limiteSuperior;

    }

    public ArrayList<FuncionDePertenenciaSalida> getUniversoDelDiscursoVariableVoltaje() {
        return universoDelDiscursoVariableVoltaje;
    }

    public void setUniversoDelDiscursoVariableVoltaje(ArrayList<FuncionDePertenenciaSalida>
                                                        universoDelDiscursoVariableVoltaje) {
        this.universoDelDiscursoVariableVoltaje = universoDelDiscursoVariableVoltaje;
    }

    public double getLimiteInferior() {
        return limiteInferior;
    }

    public void setLimiteInferior(double limiteInferior) {
        this.limiteInferior = limiteInferior;
    }

    public double getLimiteSuperior() {
        return limiteSuperior;
    }

    public void setLimiteSuperior(double limiteSuperior) {
        this.limiteSuperior = limiteSuperior;
    }

    public double getValorDeSalida() {
        return valorDeSalida;
    }

    public void setValorDeSalida(double valorDeSalida) {
        this.valorDeSalida = valorDeSalida;
    }
```

```
public void limpiaValorDeSalida() {

    valorDeSalida = 0.0;

}

public void configuraDesdifusor() {

    int numeroDeFunciones = 5;
    double rangoFuncion = (Math.abs(limiteInferior)+
                           Math.abs(limiteSuperior))/numeroDeFunciones;

    FuncionDePertenenciaSalida a = new
    FuncionTrapezoidalSalidaVoltaje("Trapezoidal","Muy Poco",
    limiteInferior,limiteInferior+rangoFuncion);
    a.calculaCoordenadas();
    a.construyeArreglosParaGrafica();

    FuncionDePertenenciaSalida b = new
    FuncionGaussianaSalidaVoltaje("Gaussiana","Poco",
    limiteInferior+rangoFuncion,limiteInferior+rangoFuncion*2);
    b.calculaCoordenadas();
    b.construyeArreglosParaGrafica();

    FuncionDePertenenciaSalida c = new
    FuncionTriangularSalidaVoltaje("Triangular","Moderado",
    limiteInferior+rangoFuncion*2,limiteInferior+rangoFuncion*3);
    c.calculaCoordenadas();
    c.construyeArreglosParaGrafica();

    FuncionDePertenenciaSalida d = new
    FuncionGaussianaSalidaVoltaje("Gaussiana","Mucho",
    limiteInferior+rangoFuncion*3,limiteInferior+rangoFuncion*4);
    d.calculaCoordenadas();
    d.construyeArreglosParaGrafica();

    FuncionDePertenenciaSalida e = new
    FuncionTrapezoidalSalidaVoltaje("Trapezoidal","Demasiado",
    limiteInferior+rangoFuncion*4,limiteInferior+rangoFuncion*5);
    e.calculaCoordenadas();
    e.construyeArreglosParaGrafica();

    FuncionDePertenenciaSalida f = new
    FuncionSingletoneSalidaVoltaje("SingleTone","Nulo",0.0,0.0);
    f.calculaCoordenadas();
    f.construyeArreglosParaGrafica();

    universoDelDiscursoVariableVoltaje.add(a);
    universoDelDiscursoVariableVoltaje.add(b);
    universoDelDiscursoVariableVoltaje.add(c);
    universoDelDiscursoVariableVoltaje.add(d);
    universoDelDiscursoVariableVoltaje.add(e);
    universoDelDiscursoVariableVoltaje.add(f);

}

public void calculaValorReal(ArrayList<Double> datosMaquinaDeInferencia,
                                          ArrayList<String>etiquetasMaquinaDeInferencia) {

    if(etiquetasMaquinaDeInferencia.get(0).equals("Nulo")) {

        valorDeSalida = universoDelDiscursoVariableVoltaje.get(
                                       universoDelDiscursoVariableVoltaje.size()-1).
                                       obtenCentroideDeLaFuncion();

    } else {
```

```
    if(etiquetasMaquinaDeInferencia.size() == 1) {

        String etiquetaMaquina = etiquetasMaquinaDeInferencia.get(0);

        for(int h=0; h<universoDelDiscursoVariableVoltaje.size(); h++) {

            if(etiquetaMaquina.equals(
                    universoDelDiscursoVariableVoltaje.get(h).getEtiqueta())){

                valorDeSalida = universoDelDiscursoVariableVoltaje.get(h).
                                        obtenCentroideDeLaFuncion();

                break;

            }

        }

    } else {

        ArrayList<Double> datosValoresDePertenencia = datosMaquinaDeInferencia;
        ArrayList<String> etiquetas  = etiquetasMaquinaDeInferencia;
        double [] datosCurva1 = new double [etiquetas.size()];
        double [] datosCurva2 = new double [etiquetas.size()];

        for(int h=0; h<universoDelDiscursoVariableVoltaje.size(); h++) {

             if(etiquetas.get(0).equals(
                    universoDelDiscursoVariableVoltaje.get(h).getEtiqueta())) {

                datosCurva1 = universoDelDiscursoVariableVoltaje.get(h).
                                    obtenDatosDelAreaBajoLaCurva(
                                    datosValoresDePertenencia.get(0));
                break;
            }

        }

        for(int h=0; h<universoDelDiscursoVariableVoltaje.size(); h++) {


            if(etiquetas.get(1).equals(
                    universoDelDiscursoVariableVoltaje.get(h).getEtiqueta())) {

                datosCurva2 = universoDelDiscursoVariableVoltaje.get(h).
                                    obtenDatosDelAreaBajoLaCurva(
                                    datosValoresDePertenencia.get(1));
                break;
            }

        }

        double [] datosDelAreaBajoLasCurvas = new double [etiquetas.size()*2];

        for(int h=0; h<etiquetas.size(); h++) {

            datosDelAreaBajoLasCurvas[h] = datosCurva1[h];
            datosDelAreaBajoLasCurvas[h+2] = datosCurva2[h];

        }

        valorDeSalida = calculaCentroideDeDosFunciones(
                            datosDelAreaBajoLasCurvas, datosValoresDePertenencia);

    }

}
```

```
    }

    private double calculaCentroideDeDosFunciones(double [] datosDelAreaBajoLaCurva,
                                                  ArrayList<Double> valoresDePertenencia) {

        double valorCalculado;
        double acumuladorFuncion1 = 0.0;
        double acumuladorFuncion2 = 0.0;


        for(int h=0; h<datosDelAreaBajoLaCurva.length/2; h++) {

            acumuladorFuncion1 += datosDelAreaBajoLaCurva[h];
            acumuladorFuncion2 += datosDelAreaBajoLaCurva[h+2];

        }

        double x1 = acumuladorFuncion1 * valoresDePertenencia.get(0);
        double x2 = acumuladorFuncion2 * valoresDePertenencia.get(1);
        double numerador = x1 + x2;
        double y1 = valoresDePertenencia.get(0) * datosDelAreaBajoLaCurva.length/2;
        double y2 = valoresDePertenencia.get(1) * datosDelAreaBajoLaCurva.length/2;
        double denominador = y1+ y2;
        valorCalculado = numerador/denominador;

        return valorCalculado;

    }

}
```

FuncionDePertenenciaSalida.java

```
import java.util.ArrayList;

public abstract class FuncionDePertenenciaSalida implements InterfazFuncionDePertenenciaSalida {

    protected String tipo;
    protected String etiqueta;
    protected double rangoInferior;
    protected double rangoSuperior;
    protected double valorDeSalida;
    protected ArrayList<Double> coordenadas = new ArrayList<>();
    protected ArrayList<double []> arreglosGrafica = new ArrayList<>();

    public FuncionDePertenenciaSalida() {

    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public String getEtiqueta() {
        return etiqueta;
    }

    public void setEtiqueta(String etiqueta) {
        this.etiqueta = etiqueta;
    }
```

```
public double getRangoInferior() {
    return rangoInferior;
}

public void setRangoInferior(double rangoInferior) {
    this.rangoInferior = rangoInferior;
}

public double getRangoSuperior() {
    return rangoSuperior;
}

public void setRangoSuperior(double rangoSuperior) {
    this.rangoSuperior = rangoSuperior;
}

public double getValorDeSalida() {
    return valorDeSalida;
}

public void setValorDeSalida(double valorDeSalida) {
    this.valorDeSalida = valorDeSalida;
}

public ArrayList<Double> getCoordenadas() {
    return coordenadas;
}

public void setCoordenadas(ArrayList<Double> coordenadas) {
    this.coordenadas = coordenadas;
}

public ArrayList<double[]> getArreglosGrafica() {
    return arreglosGrafica;
}

public void setArreglosGrafica(ArrayList<double[]> arreglosGrafica) {
    this.arreglosGrafica = arreglosGrafica;
}

@Override
public void calculaCoordenadas() {

}

@Override
public double [] obtenDatosDelAreaBajoLaCurva(double valoresFuncionDePertenencia) {

    return null;

}

@Override
public double calculaValorDePertenencia(double valorReal) {

    return 0;

}

@Override
public void construyeArreglosParaGrafica() {

}

@Override
public double obtenCentroideDeLaFuncion() {
```

```
        return 0;

    }

}
```

InterfazFuncionDePertenenciaSalida.java

```
public interface InterfazFuncionDePertenenciaSalida {

    void calculaCoordenadas();

    double calculaValorDePertenencia(double valorReal);

    double [] obtenDatosDelAreaBajoLaCurva(double valorFuncionDePertenencia);

    double obtenCentroideDeLaFuncion();

    void construyeArreglosParaGrafica();

}
```

FuncionTrapezoidalSalidaVoltaje.java

```
import controladorDifuso.desdifusor.FuncionDePertenenciaSalida;

public class FuncionTrapezoidalSalidaVoltaje extends FuncionDePertenenciaSalida {

    private double a;
    private double b;
    private double c;
    private double d;

    public FuncionTrapezoidalSalidaVoltaje() {

    }

    public FuncionTrapezoidalSalidaVoltaje(String tipo, String etiqueta, double rangoInferior,
                                                                          double rangoSuperior) {

        this.tipo = tipo;
        this.etiqueta = etiqueta;
        this.rangoInferior = rangoInferior;
        this.rangoSuperior = rangoSuperior;

    }

    @Override
    public void calculaCoordenadas() {

        a = this.rangoInferior-0.25;
        b = this.rangoInferior;
        c = this.rangoSuperior;
        d = this.rangoSuperior+0.25;

        this.coordenadas.add(a); this.coordenadas.add(b);
        this.coordenadas.add(c); this.coordenadas.add(d);


    }
    @Override
    public double calculaValorDePertenencia(double valorReal) {

        double xa = Math.abs(valorReal)-Math.abs(a);
```

```
    double ba = Math.abs(b)-Math.abs(a);
    double dx = Math.abs(d)-Math.abs(valorReal);
    double dc = Math.abs(d)-Math.abs(c);
    double div1 = xa/ba;
    double div2 = dx/dc;
    double min1 = Math.min(div1, 1.0);
    double min2 = Math.min(min1, div2);
    double max = Math.max(min2, 0.0);
    double pertenencia = max;

    return pertenencia;

}

@Override
public double [] obtenDatosDelAreaBajoLaCurva(double valorFuncionDePertenencia) {

    double [] valoresDelAreaBajoLaCurva = new double [2];
    double valorPrimero = a+(10*valorFuncionDePertenencia);
    double valorUltimo = d-(10*valorFuncionDePertenencia);

    if(valorUltimo-valorPrimero == 0.0) {

        valoresDelAreaBajoLaCurva[0] = Math.abs(valorPrimero);
        valoresDelAreaBajoLaCurva[1] = valorUltimo;

    } else {

        valoresDelAreaBajoLaCurva[0] = valorPrimero;
        valoresDelAreaBajoLaCurva[1] = valorUltimo;

    }

    return valoresDelAreaBajoLaCurva;

}

@Override
public double obtenCentroideDeLaFuncion () {

    double centroide;

    if(b+c == 0.0) {

        centroide = 0.0;

    }

    centroide = b+((c-b)/2);

    return centroide;

}

@Override
public void construyeArreglosParaGrafica() {

    double dimensionBase = 0.0;
    if(a<0.0 && d<0.0) {

        dimensionBase = Math.abs(a)-Math.abs(d);

    }

    if(a<0.0 && d>0.0) {

        dimensionBase += Math.abs(a)+d;
```

```
        }

        dimensionBase = d-a;
        int muestras = 50;
        int m = muestras -1;
        double delta = dimensionBase/m;
        double [] valoresX = new double [muestras];
        double [] valoresY = new double [muestras];
        double auxX = 0.0;

        for(int h=0; h<muestras; h++) {

            double aux = a+auxX;
            valoresX[h] = aux;
            valoresY[h] = calculaValorDePertenencia(aux);
            auxX += delta;

        }

        this.arreglosGrafica.add(valoresX);
        this.arreglosGrafica.add(valoresY);

    }

}
```

FuncionGaussianaSalidaVoltaje.java

```
import controladorDifuso.desdifusor.FuncionDePertenenciaSalida;

public class FuncionTrapezoidalSalidaVoltaje extends FuncionDePertenenciaSalida {

    private double a;
    private double b;
    private double c;
    private double d;

    public FuncionTrapezoidalSalidaVoltaje() {

    }

    public FuncionTrapezoidalSalidaVoltaje(String tipo, String etiqueta, double rangoInferior,
                                                                          double rangoSuperior) {

        this.tipo = tipo;
        this.etiqueta = etiqueta;
        this.rangoInferior = rangoInferior;
        this.rangoSuperior = rangoSuperior;

    }

    @Override
    public void calculaCoordenadas() {

        a = this.rangoInferior-0.25;
        b = this.rangoInferior;
        c = this.rangoSuperior;
        d = this.rangoSuperior+0.25;

        this.coordenadas.add(a); this.coordenadas.add(b);
        this.coordenadas.add(c); this.coordenadas.add(d);


    }
```

```
@Override
public double calculaValorDePertenencia(double valorReal) {

    double xa = Math.abs(valorReal)-Math.abs(a);
    double ba = Math.abs(b)-Math.abs(a);
    double dx = Math.abs(d)-Math.abs(valorReal);
    double dc = Math.abs(d)-Math.abs(c);
    double div1 = xa/ba;
    double div2 = dx/dc;
    double min1 = Math.min(div1, 1.0);
    double min2 = Math.min(min1, div2);
    double max = Math.max(min2, 0.0);
    double pertenencia = max;

    return pertenencia;

}

@Override
public double [] obtenDatosDelAreaBajoLaCurva(double valorFuncionDePertenencia) {

    double [] valoresDelAreaBajoLaCurva = new double [2];
    double valorPrimero = a+(10*valorFuncionDePertenencia);
    double valorUltimo = d-(10*valorFuncionDePertenencia);

    if(valorUltimo-valorPrimero == 0.0) {

        valoresDelAreaBajoLaCurva[0] = Math.abs(valorPrimero);
        valoresDelAreaBajoLaCurva[1] = valorUltimo;

    } else {

        valoresDelAreaBajoLaCurva[0] = valorPrimero;
        valoresDelAreaBajoLaCurva[1] = valorUltimo;

    }

    return valoresDelAreaBajoLaCurva;

}

@Override
public double obtenCentroideDeLaFuncion () {

    double centroide;

    if(b+c == 0.0) {

        centroide = 0.0;

    }

    centroide = b+((c-b)/2);

    return centroide;

}

@Override
public void construyeArreglosParaGrafica() {

    double dimensionBase = 0.0;

    if(a<0.0 && d<0.0) {

        dimensionBase = Math.abs(a)-Math.abs(d);
```

```
        }

        if(a<0.0 && d>0.0) {

            dimensionBase += Math.abs(a)+d;

        }

        dimensionBase = d-a;
        int muestras = 50;
        int m = muestras -1;
        double delta = dimensionBase/m;
        double [] valoresX = new double [muestras];
        double [] valoresY = new double [muestras];
        double auxX = 0.0;

        for(int h=0; h<muestras; h++) {

            double aux = a+auxX;
            valoresX[h] = aux;
            valoresY[h] = calculaValorDePertenencia(aux);
            auxX += delta;

        }

        this.arreglosGrafica.add(valoresX);
        this.arreglosGrafica.add(valoresY);

    }

}
```

FuncionTriangularSalidaVoltaje.java

```
import controladorDifuso.desdifusor.FuncionDePertenenciaSalida;

public class FuncionTriangularSalidaVoltaje extends FuncionDePertenenciaSalida {

    private double a;
    private double b;
    private double c;

    public FuncionTriangularSalidaVoltaje() {

    }

    public FuncionTriangularSalidaVoltaje(String tipo, String etiqueta, double rangoInferior,
                                                                        double rangoSuperior) {

        this.tipo = tipo;
        this.etiqueta = etiqueta;
        this.rangoInferior = rangoInferior;
        this.rangoSuperior = rangoSuperior;

    }

    @Override
    public void calculaCoordenadas() {

        a = this.rangoInferior-0.25;
        c = this.rangoSuperior+0.25;

        if (Math.abs(Math.abs(a)-Math.abs(c))/2 == 0.0) {

            b = 0.0;
```

```
    }

    if( a<0.0 && c<0.0) {

        double aux = Math.abs(Math.abs(a)-Math.abs(c))/2;
        b = a +aux;

    }

    if(a<0.0 && c>0.0) {

        double aux = (Math.abs(a)+c)/2;
        b = a + aux;

    } else {

        b = (a+c)/2;

    }

    this.coordenadas.add(a); this.coordenadas.add(b); this.coordenadas.add(c);

}

@Override
public double calculaValorDePertenencia(double valorReal) {

    double xa = Math.abs(valorReal)-Math.abs(a);
    double ba = Math.abs(b)-Math.abs(a);
    double cx = Math.abs(c)-Math.abs(valorReal);
    double cb = Math.abs(c)-Math.abs(b);
    double div1 = xa/ba;
    double div2 = cx/cb;
    double min = Math.min(div1, div2);
    double max = Math.max(min, 0.0);
    double pertenencia = max;

    return pertenencia;

}

@Override
public double [] obtenDatosDelAreaBajoLaCurva(double valorFuncionDePertenencia) {

    double [] valoresDelAreaBajoLaCurva = new double [2];

    double baseTriangulo = Math.abs(Math.abs(a)-Math.abs(b));
    double valorPrimero = a+(baseTriangulo*valorFuncionDePertenencia);
    double valorUltimo = c-(baseTriangulo*valorFuncionDePertenencia);

    if(b == 0.0) {

        valoresDelAreaBajoLaCurva[0] = Math.abs(valorPrimero);
        valoresDelAreaBajoLaCurva[1] = valorUltimo;

    } else {

        valoresDelAreaBajoLaCurva[0] = valorPrimero;
        valoresDelAreaBajoLaCurva[1] = valorUltimo;

    }
    return valoresDelAreaBajoLaCurva;

}

@Override
public double obtenCentroideDeLaFuncion () {
```

```
        double centroide;
        centroide = b;

        return centroide;

    }


    @Override
    public void construyeArreglosParaGrafica() {

        int muestras = 50;
        int m = muestras -1;
        double delta = (Math.abs(b-c)*2)/m;
        double [] valoresX = new double [muestras];
        double [] valoresY = new double [muestras];
        double auxX = 0.0;

        for(int h=0; h<muestras; h++) {

            double aux = a+auxX;
            valoresX[h] = aux;
            valoresY[h] = calculaValorDePertenencia(aux);
            auxX += delta;

        }

        this.arreglosGrafica.add(valoresX);
        this.arreglosGrafica.add(valoresY);

    }

}
```

FuncionSingletoneSalidaVoltaje.java

```
import controladorDifuso.desdifusor.FuncionDePertenenciaSalida;

public class FuncionSingletoneSalidaVoltaje extends FuncionDePertenenciaSalida {

    private double unicaCoordenada;

    public FuncionSingletoneSalidaVoltaje() {

    }

    public FuncionSingletoneSalidaVoltaje(String tipo, String etiqueta, double rangoInferior,
                                                                        double rangoSuperior) {

        this.tipo = tipo;
        this.etiqueta = etiqueta;
        this.rangoInferior = rangoInferior;
        this.rangoSuperior = rangoSuperior;

    }

    @Override
    public void calculaCoordenadas() {

        unicaCoordenada = 0.0;
        this.coordenadas.add(unicaCoordenada);

    }
```

```
    @Override
    public double calculaValorDePertenencia(double valorReal) {

        double pertenencia = 1.0;

        return pertenencia;

    }

    @Override
    public double [] obtenDatosDelAreaBajoLaCurva(double valorFuncionDePertenencia) {

        double [] valoresDelAreaBajoLaCurva = new double [1];
        valoresDelAreaBajoLaCurva[0] = 0.0;

        return valoresDelAreaBajoLaCurva;

    }

    @Override
    public double obtenCentroideDeLaFuncion () {

        double centroide;
        centroide = 0.0;

        return centroide;

    }

    @Override
    public void construyeArreglosParaGrafica() {

        double [] valoresX = {0.0,0.0};
        double [] valoresY = {0.0,1.0};

        this.arreglosGrafica.add(valoresX);
        this.arreglosGrafica.add(valoresY);

    }

}
```