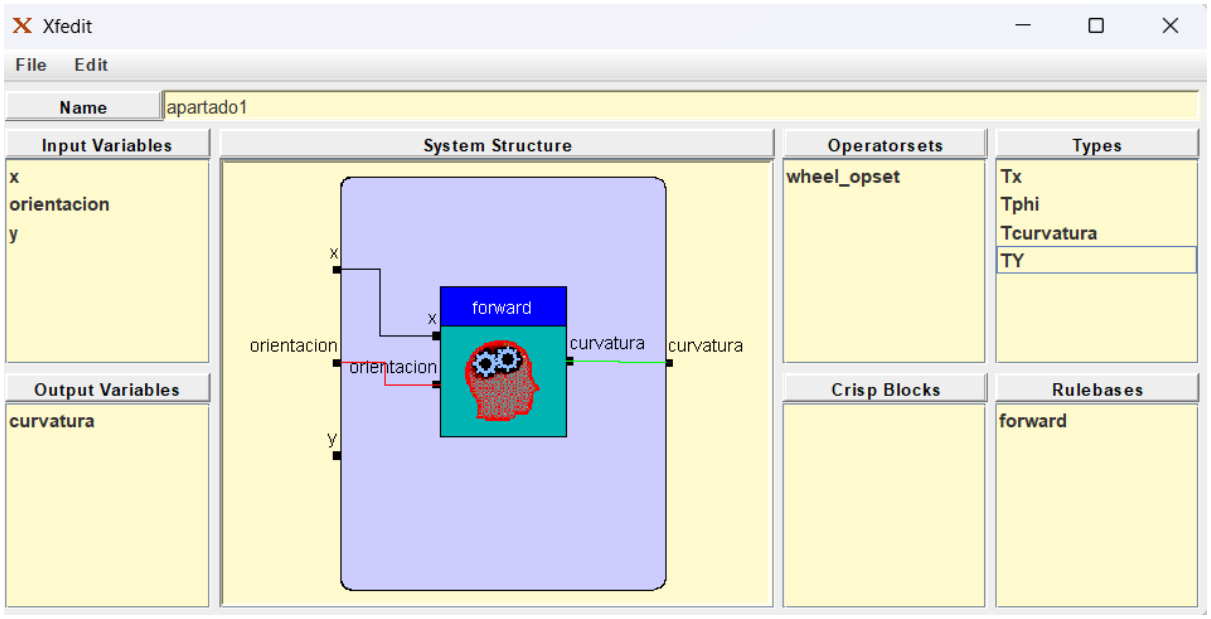
Entrega Bloque Fuzzy

Autor: Ángel Manuel Ferrer Álvarez

Asignatura: Aplicaciones de Soft Computing

Fecha: 12/10/2025

Apartado 1:



1.1 Diseño heurístico del controlador:

El objetivo del primer controlador es regular el giro del volante de un vehículo con el fin de dirigirlo hacia la posición x = 0 m y la orientación ϕ = 0°, manteniendo una velocidad constante de 1 m/s Para ello, se emplea un sistema de inferencia difusa basado en conocimiento heurístico.

1.2 Definición de variables:

Se han definido las siguientes variables lingüísticas:

Variable	**Tipo**	**Rango**	**Nº de funciones**	**Etiquetas**
x	Entrada	[-10, 10]	5	LB, LS, ZE, RS, RB
φ (orientación)	Entrada	[-180, 180]	5	LB, LM, ZE, RM, RB
y	Entrada auxiliar	[-2, 30]	3	mf0, mf1, mf2
curvatura (δ)	Salida	[-0.4, 0.4]	5	NB, NM, ZE, PM, PB

Las funciones de pertenencia empleadas son principalmente triangulares, distribuidas de forma uniforme sobre el rango de cada variable. En las variables orientación (ϕ) y curvatura (δ) se han utilizado funciones trapezoidales en los extremos (LB, RB / NB, PB).

Se ha escogido este diseño por su simplicidad.

1.3 Conjunto de operadores

El conjunto de operadores empleados, denominado wheel_opset, se muestra en la Figura 2.

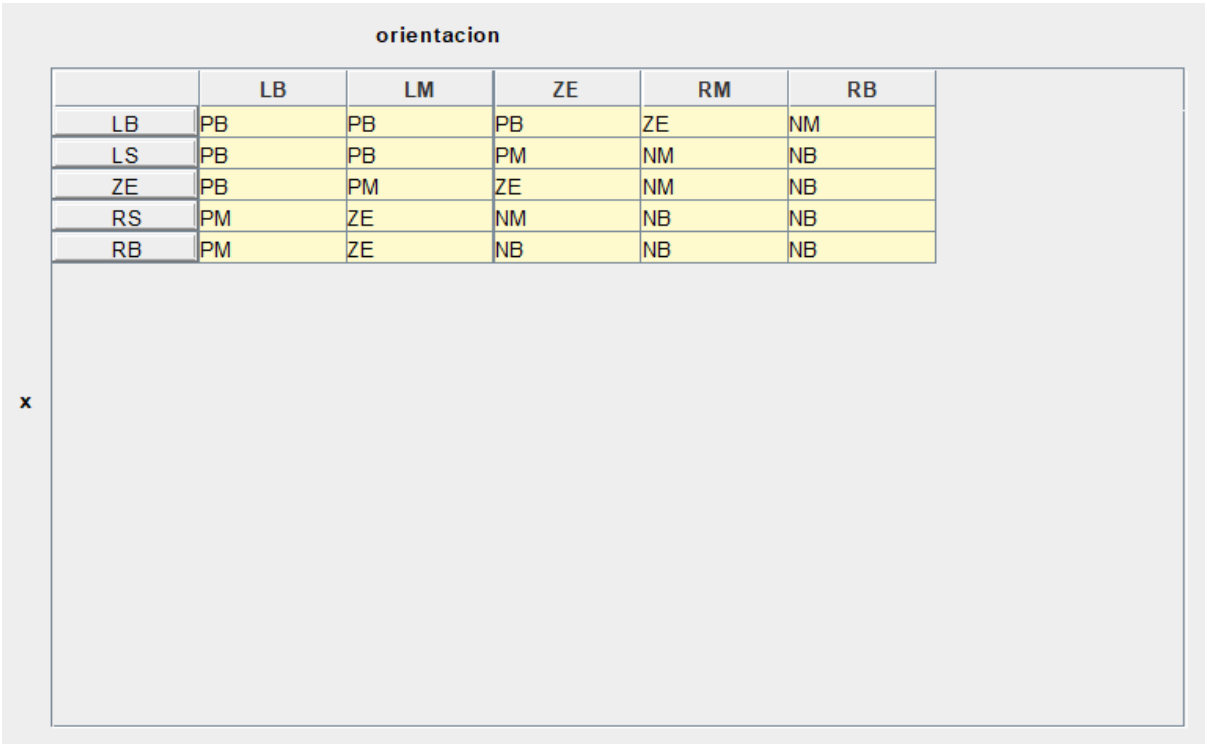
- AND: producto (xfl.prod())
- OR: máximo (xfl.max())
- Defuzzification: método de la media difusa (xfl.FuzzyMean())

Estos operadores permiten un comportamiento suave y continuo del controlador, adecuado para un sistema dinámico como la conducción.

1.4 Base de reglas:

La base de reglas forward, se diseñó de manera heurística, considerando la experiencia como conductor:

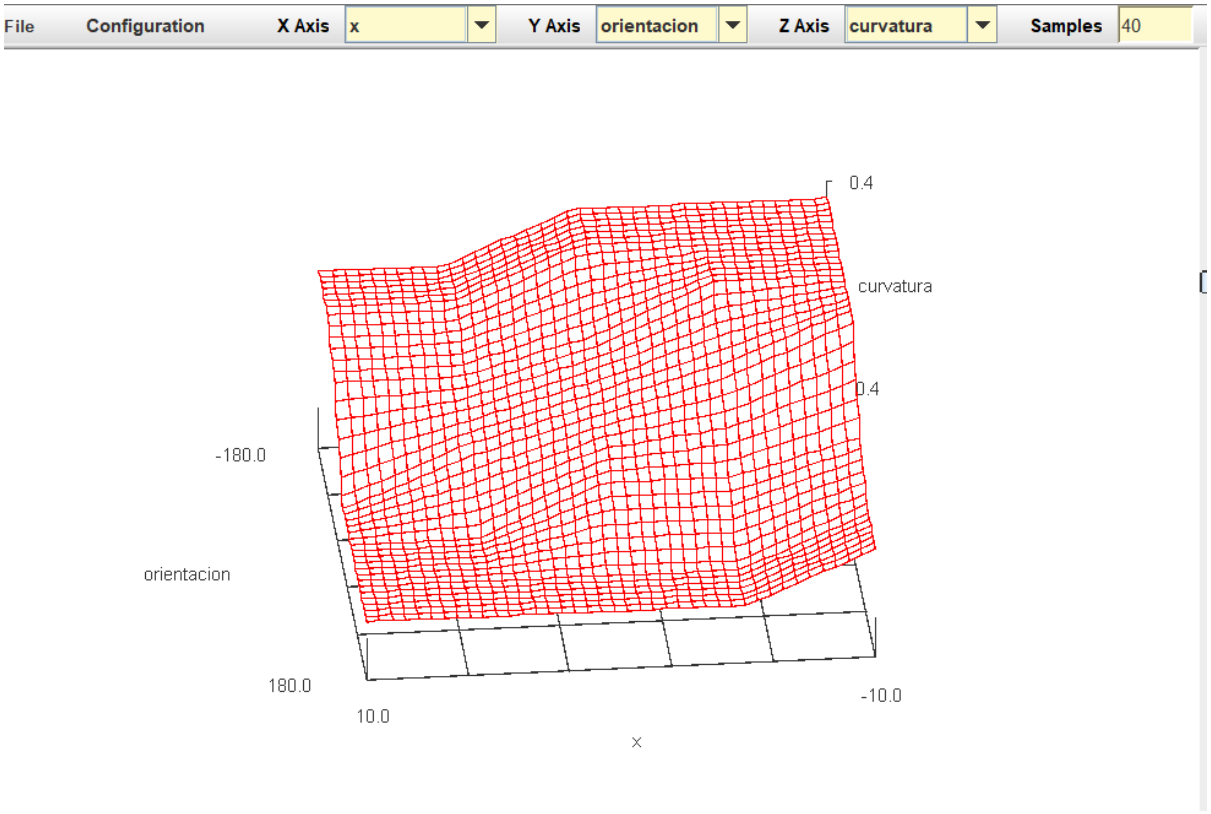
- Si el vehículo está muy desviado o muy mal orientado, se aplican giros bruscos (etiquetas NB/PB).
- Si la desviación es pequeña o moderada, se aplican correcciones suaves (etiquetas NM/PM).
- En torno al eje central, se mantiene curvatura neutra (ZE).



	LB	LM	ZE	RM	RB
LB	PB	PB	PB	ZE	NM
LS	PB	PB	PM	NM	NB
ZE	PB	PM	ZE	NM	NB
RS	PM	ZE	NM	NB	NB
RB	PM	ZE	NB	NB	NB

La Figura muestra la matriz de reglas implementada, donde las filas representan los valores lingüísticos de x y las columnas los de orientación.

1.5 Estudio de la superficie de control:



Al analizar la superficie de control se observa que en los puntos extremos x = -10, ϕ = -180° y x = 10, ϕ = 180° la salida alcanza los valores máximos de curvatura (-0.4 y 0.4, respectivamente). Estos puntos corresponden a las situaciones de mayor error en posición y orientación, donde el vehículo se encuentra completamente desalineado con la trayectoria deseada. En consecuencia, el controlador activa las reglas asociadas a los giros máximos del volante, generando una corrección brusca que busca recentrar el vehículo.

A medida que las variables x y ϕ se aproximan a cero, la superficie muestra una pendiente progresiva, indicando una reducción gradual de la curvatura. Esto refleja un comportamiento suave y continuo: el controlador disminuye la intensidad de la acción correctiva conforme la posición y la orientación mejoran.

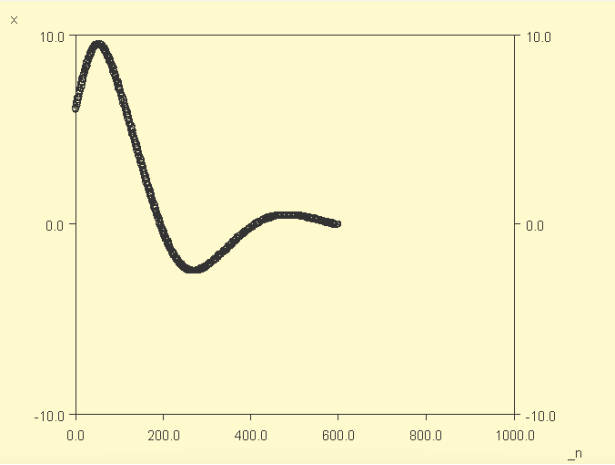
En el entorno del punto x = 0, ϕ = 0, la curvatura se aproxima a cero, lo que implica que no se aplican correcciones una vez alcanzada la posición y orientación deseadas. Esta respuesta demuestra que la

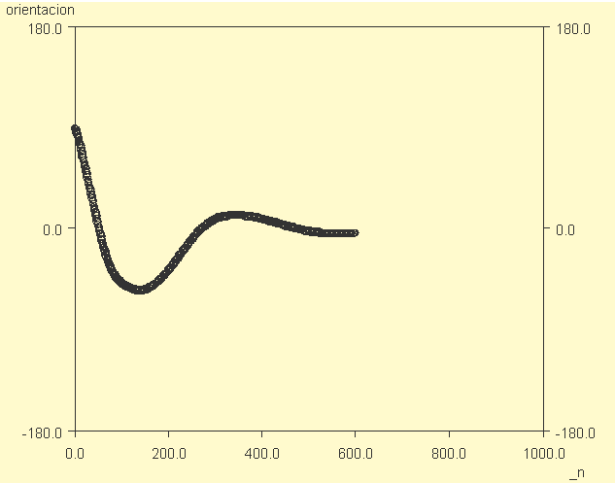
base de reglas y las funciones de pertenencia están correctamente ajustadas para proporcionar una transición estable entre las zonas de corrección intensa y la zona de equilibrio.

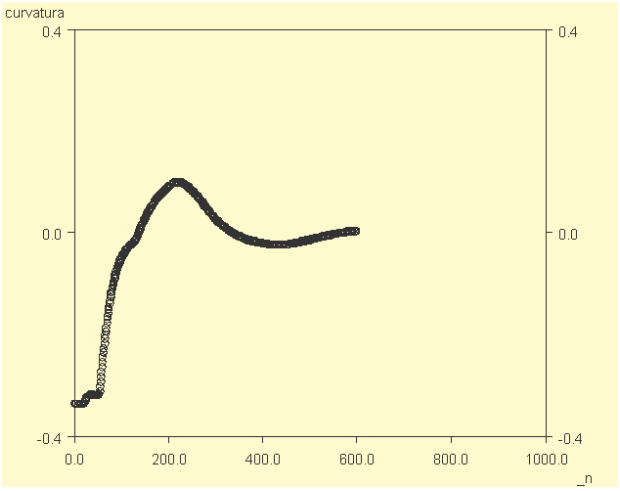
Que la superficie de control no haya salido simétrica quiere decir que no hemos seguido por igual las reglas en ambos extremos.

1.6 Estudio de simulación:

Sim-1 x0=6, ϕ0=90°

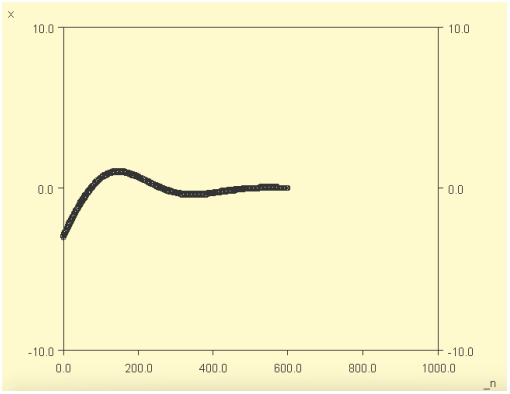


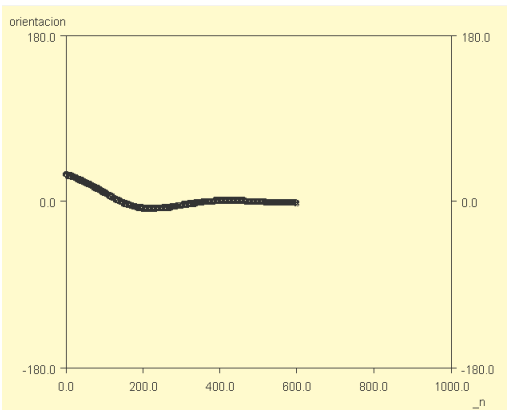


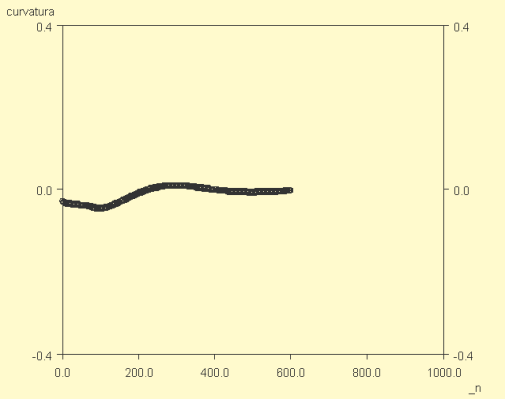


Viendo las gráficas obtenidas, se observa que al inicio, debido a la mala posición y orientación del vehículo, la curvatura alcanza valores cercanos al límite inferior, lo que indica una corrección fuerte inicial. En la gráfica de x(n) se aprecia cómo el vehículo corrige bruscamente su trayectoria al principio. A medida que se alinea con el eje, la curvatura tiende hacia cero, mostrando una acción de control más suave. Sin embargo, al sobrepasar ligeramente x = 0, la curvatura cambia de signo con un valor pequeño y positivo para realizar una corrección final y estabilizar la posición en torno al objetivo.

Sim-2 x0=-3, ϕ0=30°







En la segunda simulación se parte de una posición y orientación inicial más suaves, por lo que el comportamiento del sistema es más progresivo. La curvatura alcanza valores pequeños y se mantiene cerca de cero durante casi toda la simulación. En la gráfica de x(n) se observa una corrección gradual hacia el eje central. De forma similar, la orientación ϕ(n) converge lentamente hacia cero. El controlador mantiene así una respuesta equilibrada, con correcciones suaves que permiten alcanzar la posición y orientación deseadas de manera estable.

Evolution	
Iteration (_...	50000.
Time (_t)	897.0 ms
Plant state	
x	-4.0449E-130
orientacion	-1.1611E-129
y	5002.1
Fuzzy system output	
curvatura	1.3489E-131

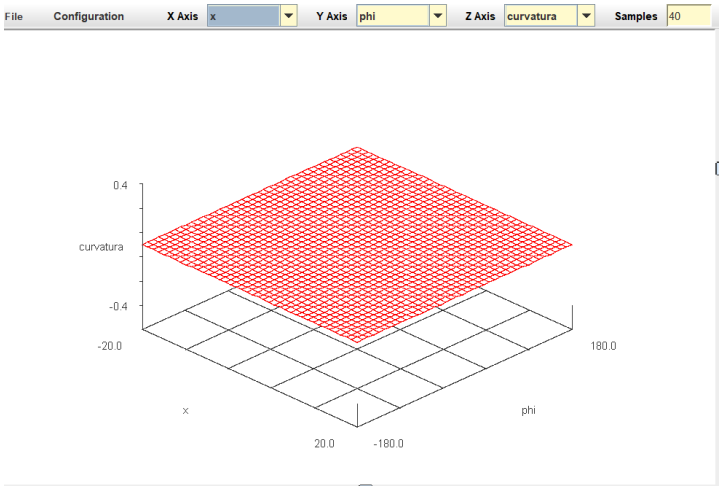
A modo de prueba, se ha establecido como límite esta vez 50000 iteraciones a ver si consigue quedarse quieto en x=0 y ϕ = 0. Se puede ver que el controlador consigue acercar la posición y la orientación mucho al cero, pero no llega a dejarlas exactamente en x = 0 ni ϕ = 0. Esto pasa porque al ser un sistema difuso, la salida se va ajustando poco a poco y no hay una acción que obligue al sistema a quedarse justo en el punto objetivo. En la práctica se queda oscilando muy cerca del equilibrio, lo que indica que el control es estable y que el error final es muy pequeño, aunque no completamente nulo.

Apartado 2:

1.1 FlatSystem5x5:

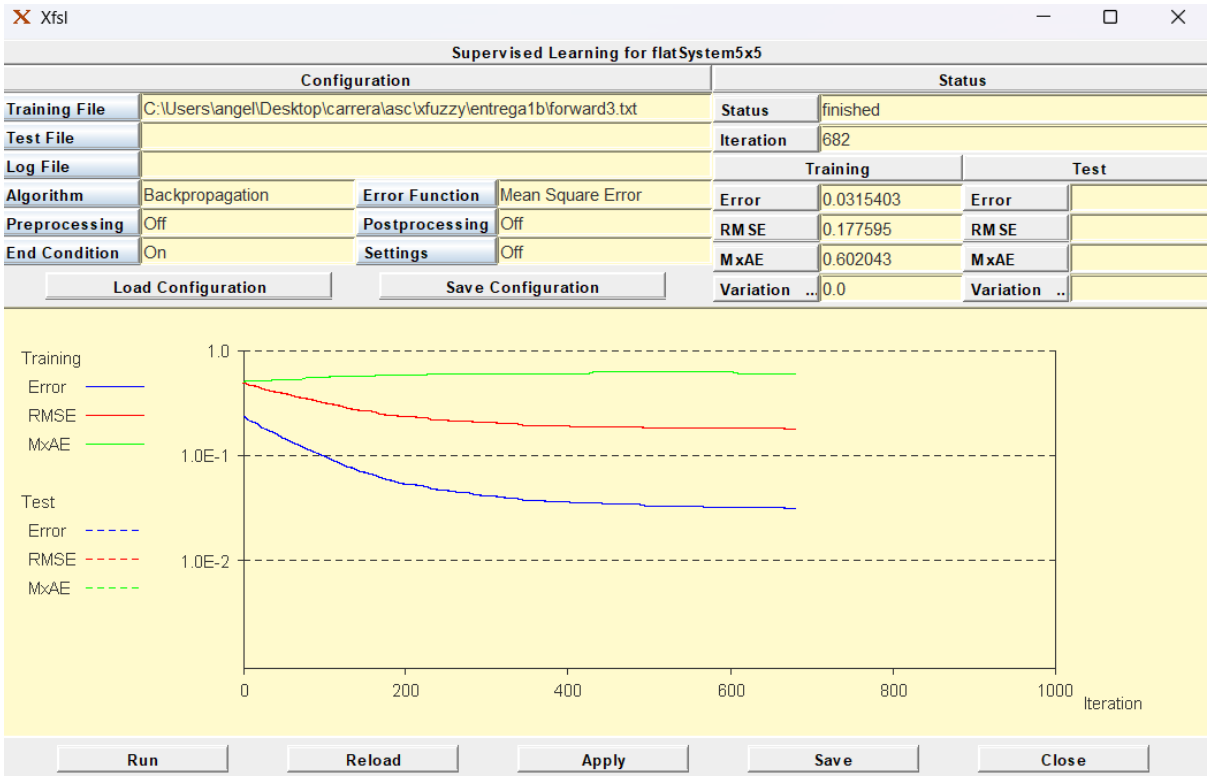
El sistema diseñado es un flat system con dos entradas, x y ϕ, y una salida, curvatura. Para cada variable de entrada se han definido 5 funciones de pertenencia triangulares.
La salida, correspondiente a la curvatura, se ha configurado con la estructura Generate para obtener

automáticamente las reglas a partir de los datos del fichero de entrenamiento, y con el método de salida FuzzyMean, que calcula la media ponderada de los valores de salida según el grado de activación de las reglas.

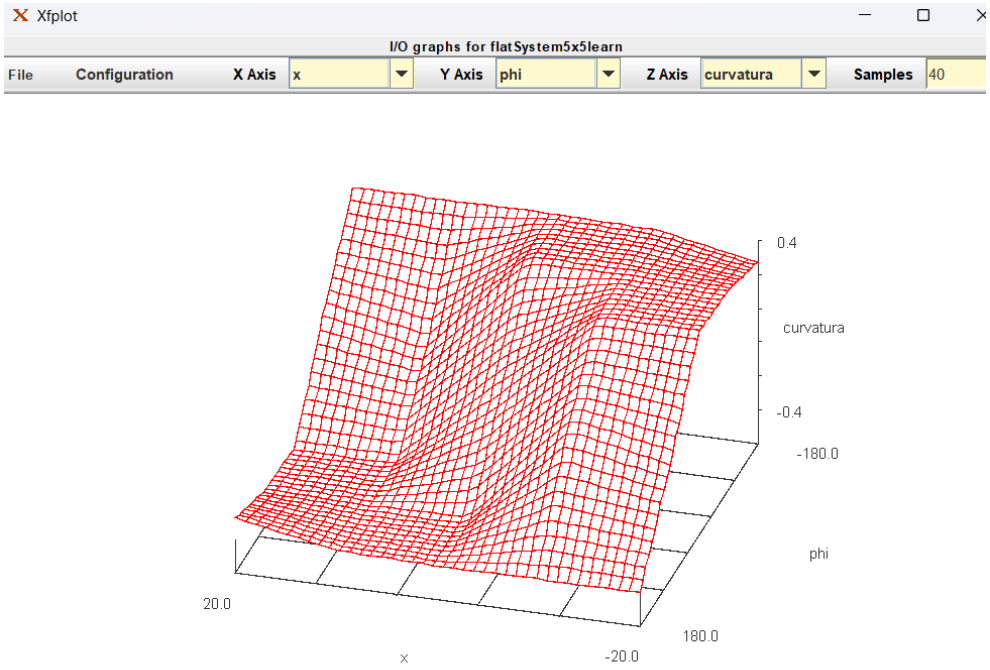


Como vemos nuestra superficie de control es plana, ya que aún no hemos entrenado al sistema.

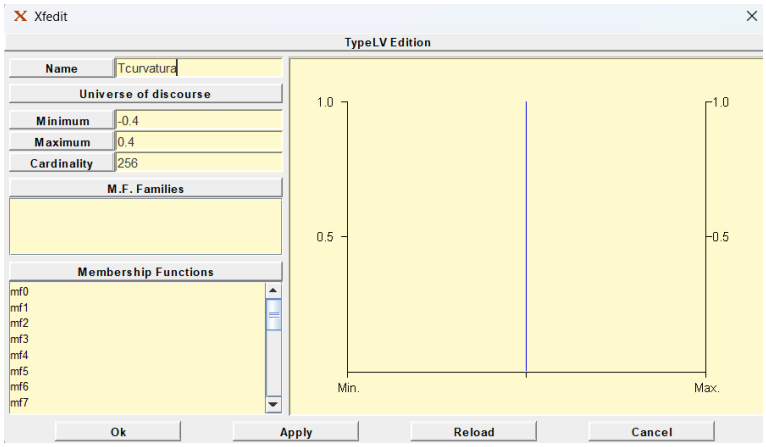
Para el proceso de aprendizaje mediante backpropagation se ha utilizado una tasa de aprendizaje de 0.05, ya que permite un ajuste progresivo y estable de los parámetros del sistema. Como condición de parada, se ha establecido una variación en el error de 0.0001, considerando que, al alcanzarse dicho umbral, el sistema ya no mejora significativamente. Se ha permitido el aprendizaje tanto en las entradas como en las salidas, y se ha empleado como función de error el Mean Square Error (MSE).

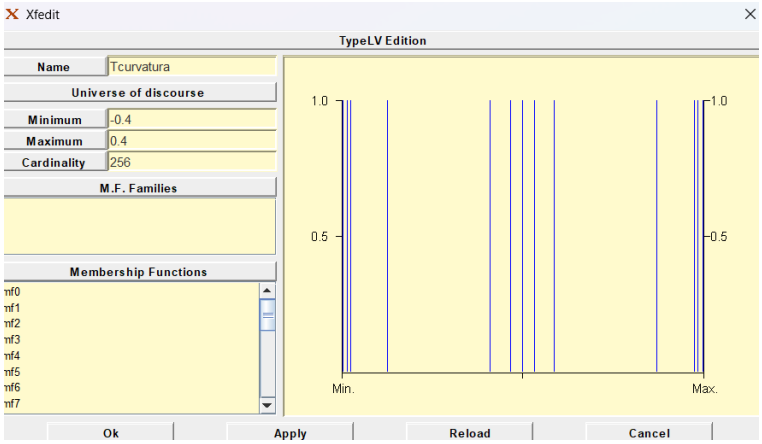


El proceso de aprendizaje mediante backpropagation se ha detenido en la iteración 682, aunque como se ve en la iteración 400 ya alcanzaba casi el mismo error cuadrático medio. El sistema presenta un Error cuadrático medio (RMSE) de 0.1776, lo cual indica que el modelo ha conseguido aprender correctamente. Además, se observa que el error disminuye de manera continua a lo largo del entrenamiento, lo que confirma una convergencia adecuada y un comportamiento coherente del sistema.



Como podemos observar, la superficie de control obtenida tras el entrenamiento ya no es plana, sino que se asemeja a la del apartado 1, mostrando un comportamiento más realista. El sistema aplica ajustes fuertes cuando el vehículo se encuentra muy desalineado o mal posicionado, mientras que en las zonas intermedias presenta una pendiente suave que va corrigiendo progresivamente la curvatura.



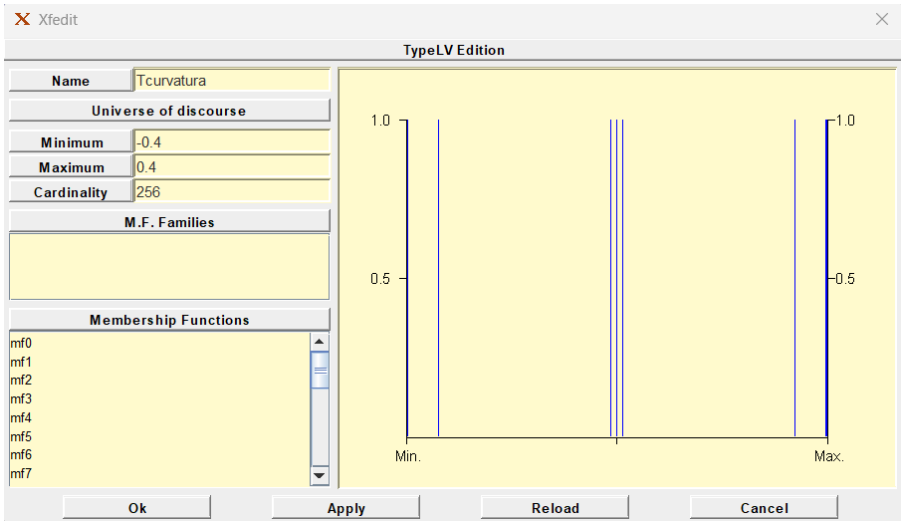


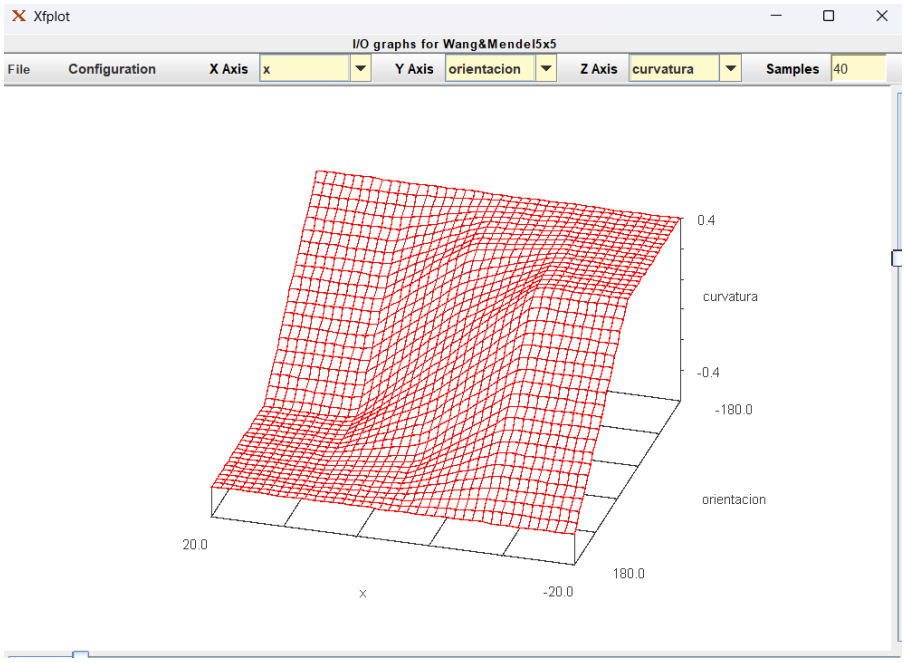
Tras el proceso de aprendizaje, el sistema ha generado un total de 25 reglas difusas, correspondientes a las 5 funciones de pertenencia definidas para cada una de las dos entradas. La variable de salida, curvatura, se ha representado mediante 25 funciones tipo singleton, una por cada regla generada. Como puede observarse en la Figura X, estas funciones, que antes estaban todas posicionadas en el 0, se distribuyen en el rango comprendido entre -0.4 y 0.4, cubriendo adecuadamente el universo de discurso y permitiendo representar diferentes niveles de corrección en la dirección del vehículo. (figura 1 antes de aprender, figura dos después de aprender)

1.2 Wang & Mendel 5x5:

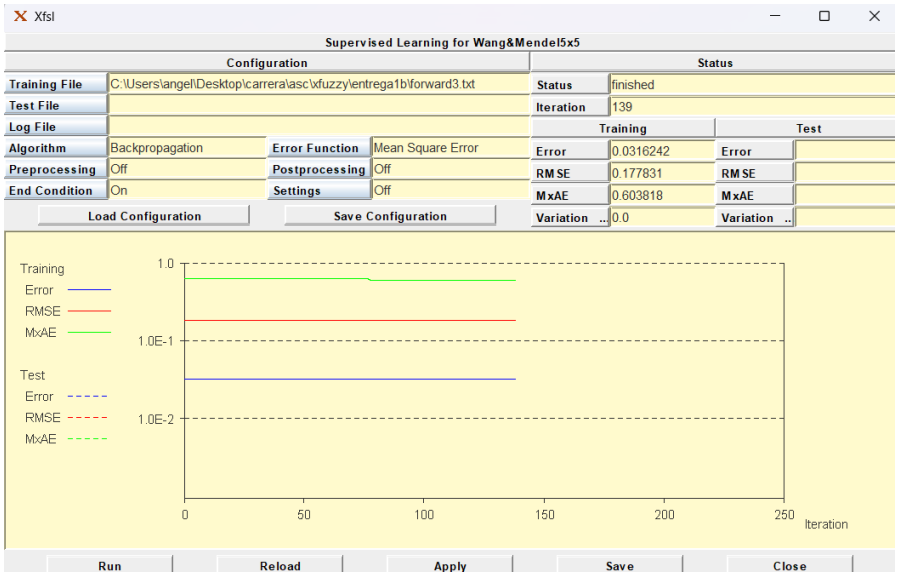
En este caso, se ha diseñado un sistema igual que el anterior, pero utilizando el algoritmo de Wang & Mendel para la generación inicial de reglas. Este algoritmo construye automáticamente una base de reglas difusas a partir de los datos de entrenamiento, sin necesidad de un ajuste previo. Para cada par de valores del conjunto de datos, el algoritmo determina a qué funciones de pertenencia pertenecen más los valores de entrada, y genera una regla del tipo: “Si x es A_i y ϕ es B_j entonces la curvatura es C_k”.

De esta forma, el sistema ya parte con las reglas y los singleton de salida distribuidos según los datos, es decir, ya tiene una estructura inicial coherente antes del proceso de entrenamiento.

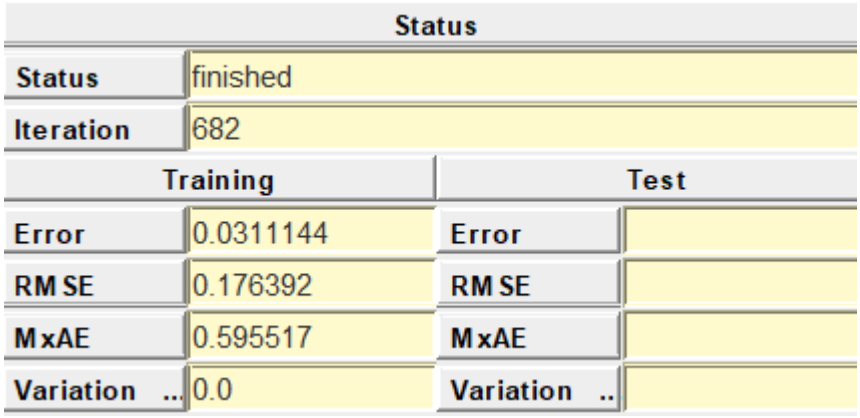




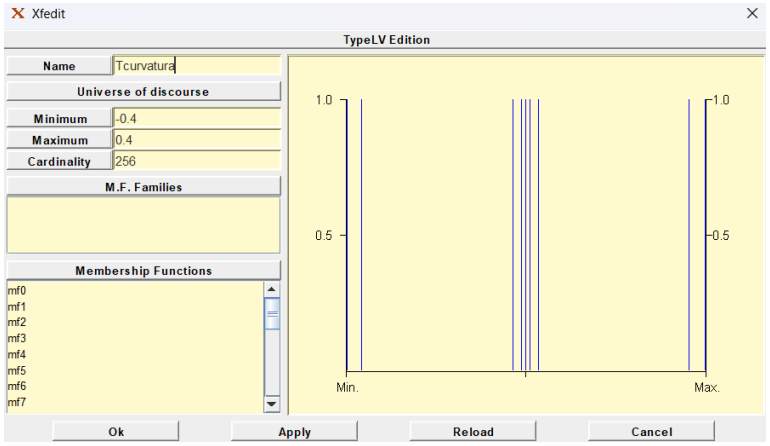
Como ya hemos comentado el sistema ya parte de una base, esto se puede apreciar muy bien en la superficie de control, que ya tiene una forma parecida a lo que buscamos.



Como ya podíamos prever por lo comentado anteriormente, el sistema no ha mostrado capacidad de aprendizaje adicional mediante backpropagation, ya que el algoritmo de Wang & Mendel genera desde el inicio una base de reglas que cumple adecuadamente con el objetivo del control. En otras palabras, el sistema ya se encontraba correctamente ajustado, por lo que el entrenamiento posterior no produce mejoras significativas en el error.



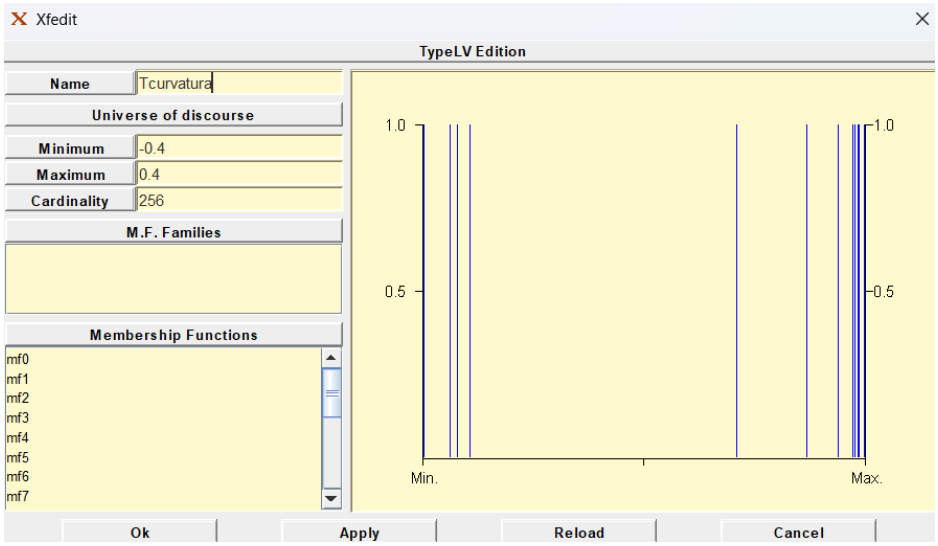
Tras realizar el mismo número de iteraciones que en el system flat, se ha alcanzado un Mean Square Error (MSE) de 0.1763, muy similar al obtenido anteriormente.

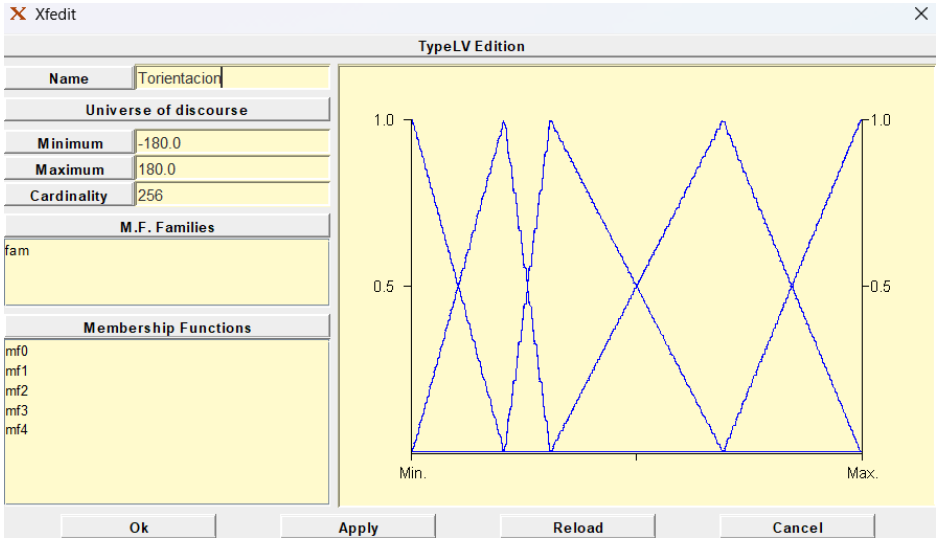


La distribución de los singleton en la salida ha sido ligeramente modificada tras el proceso, repartiéndose algunos valores adicionales en la zona central.
Esto indica que el sistema ha realizado un ajuste leve en las reglas más activas, concentrando mayor precisión en las situaciones más frecuentes del conjunto de datos, donde las correcciones de curvatura son más suaves. Respecto a la superficie de control, como cabía esperar, no ha habido cambios significativos.

1.3 Incremental Grid 5x5:

Se ha utilizado el algoritmo Incremental Grid limitando el número máximo de funciones de pertenencia a 5 y el número de reglas a 25, manteniendo así la misma complejidad que en los sistemas anteriores.
Además, se ha fijado un límite de RMSE de 0.17 como criterio de parada, buscando un rendimiento comparable o ligeramente mejor que el obtenido con el método de Wang & Mendel. Además, se ha activado la opción de aprendizaje para permitir un ajuste fino de los consecuentes.



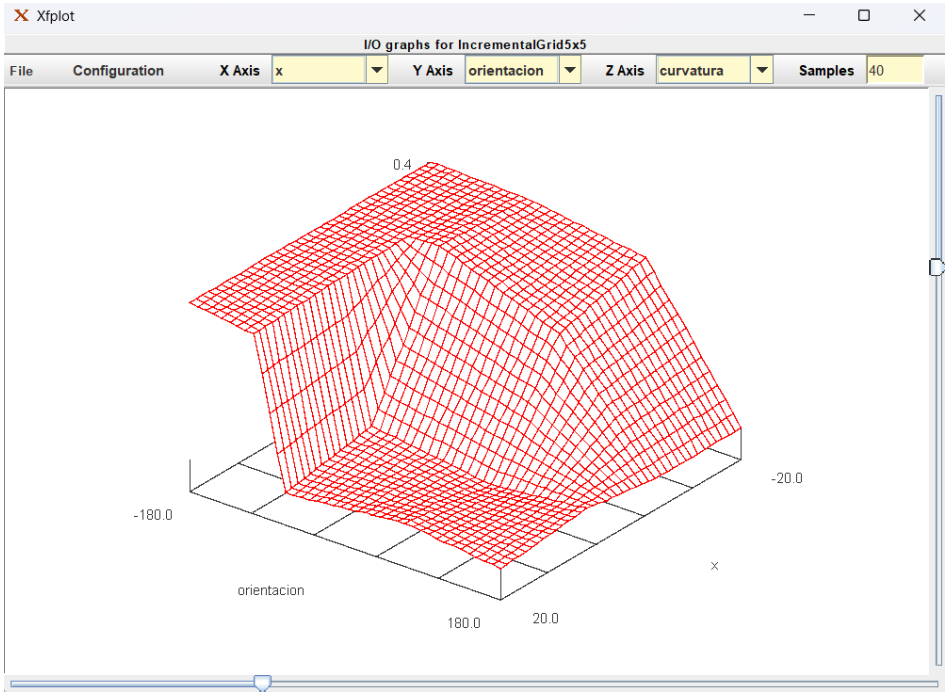


En este caso, al aplicar el algoritmo Incremental Grid, el sistema ha generado 20 reglas en lugar de las 25 del sistema flat. Esto se debe a que este algoritmo va creando solo las reglas necesarias según los

datos de entrenamiento, sin llegar a formar todas las combinaciones posibles, ya que habrá conseguido el Mean Square Error (MSE) que le habíamos puesto.

He podido observar también que el sistema ha reajustado las funciones triangulares de la orientación, desplazándolas ligeramente para adaptarse mejor a las zonas donde hay más variación en los datos.

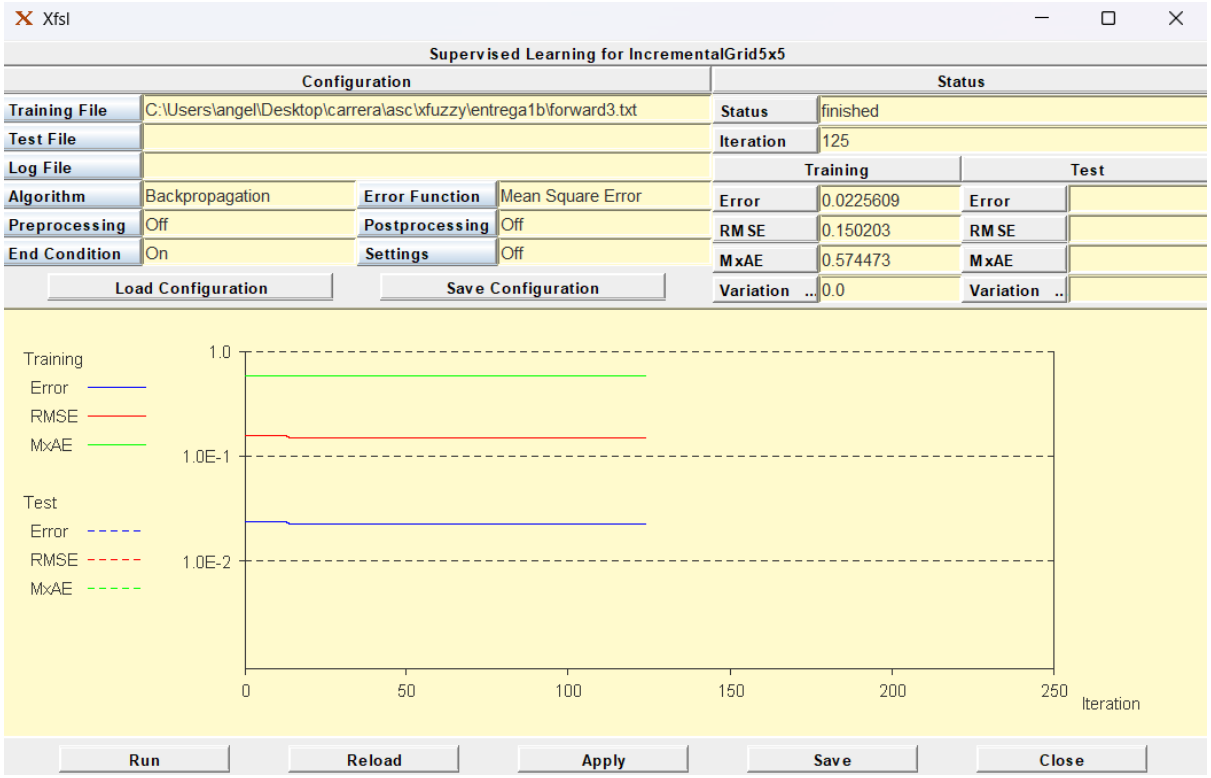
Además, los singleton de la salida (curvatura) se han repartido esta vez más hacia los extremos del rango en lugar de concentrarse en el centro, lo que indica que el sistema tiende a reforzar los ajustes más fuertes de curvatura cuando la posición u orientación del vehículo se alejan más de la trayectoria deseada.



En la superficie de control generada con el sistema Incremental Grid, se puede apreciar que en la zona izquierda se producen cambios más bruscos en la curvatura.

Esto puede deberse a que, en esta ocasión, los singleton de la salida se han distribuido más hacia los extremos, lo que provoca que el sistema reaccione con ajustes más intensos cuando las variables de entrada toman valores más alejados del centro.

Además, se observa que la superficie no es completamente simétrica, lo cual es coherente ya que el algoritmo no ha generado las 25 reglas posibles, sino únicamente 20, al detenerse una vez alcanzado el límite de error establecido.



Como ya podíamos esperar, el sistema Incremental Grid no muestra una gran mejora durante el entrenamiento con backpropagation, ya que parte de una base muy bien ajustada. Alcanzando en la iteración 125 un Mean Square Error (MSE) de 0.1502, mejorando ligeramente el rendimiento respecto a los sistemas anteriores. Esto confirma que el algoritmo ya genera desde el inicio una estructura muy precisa, necesitando pocas iteraciones para estabilizarse.

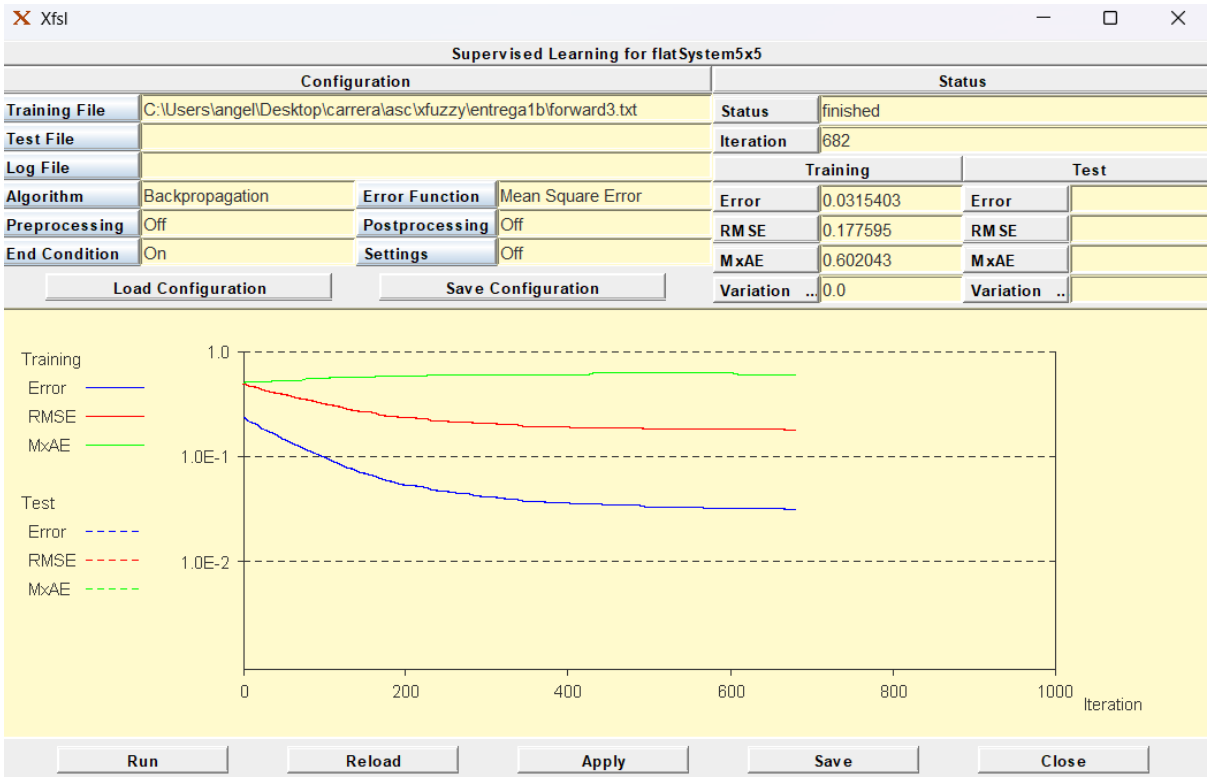
1.3 Comparativa de algoritmos:

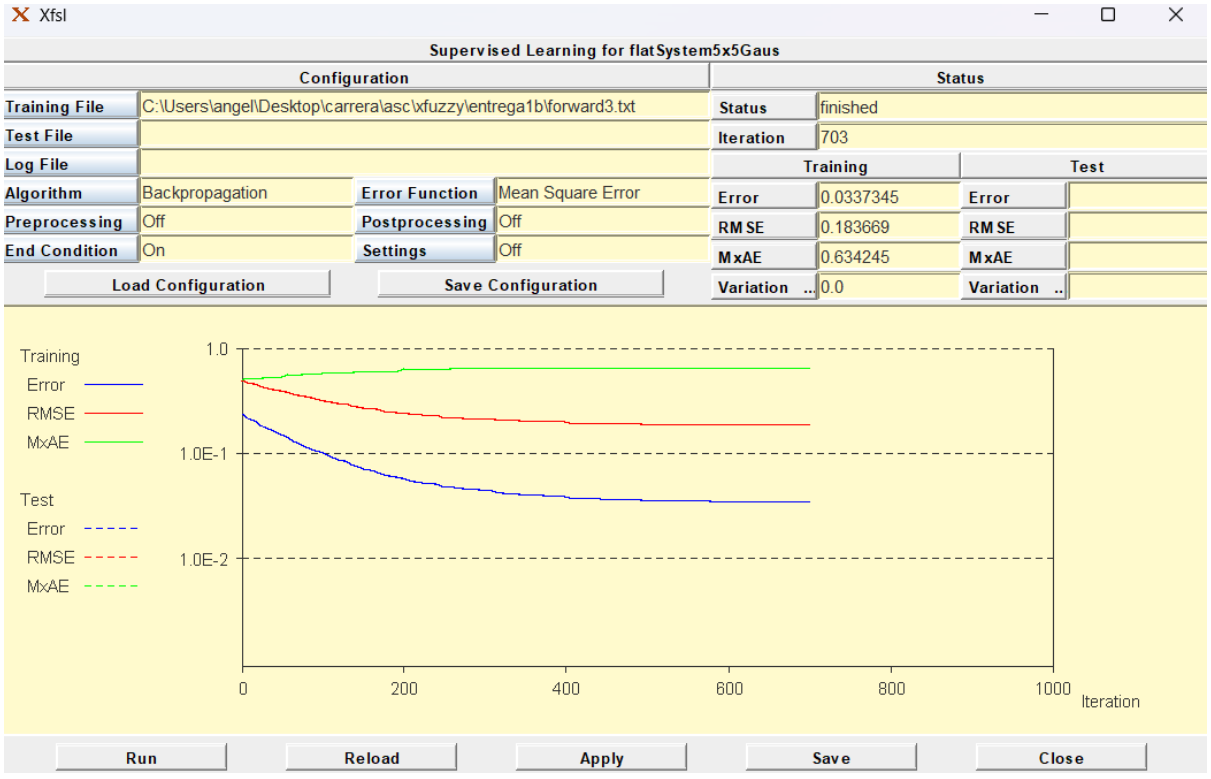
El sistema que presenta un aprendizaje más rápido es el obtenido mediante el algoritmo Incremental Grid, ya que parte de una estructura inicial muy bien adaptada a los datos, alcanzando un MSE de 0.1502 en tan solo 125 iteraciones. Esto demuestra su eficiencia, a pesar de que lo hemos limitado en su creación para que se detuviera cuando igualase aproximadamente el error de los otros sistemas, lo que hace aún más destacable su rendimiento.

Por otro lado, el sistema generado con el algoritmo de Wang & Mendel también muestra un aprendizaje más rápido que el flat system, ya que parte con una base de reglas ya construida y apenas necesita ajustes adicionales.

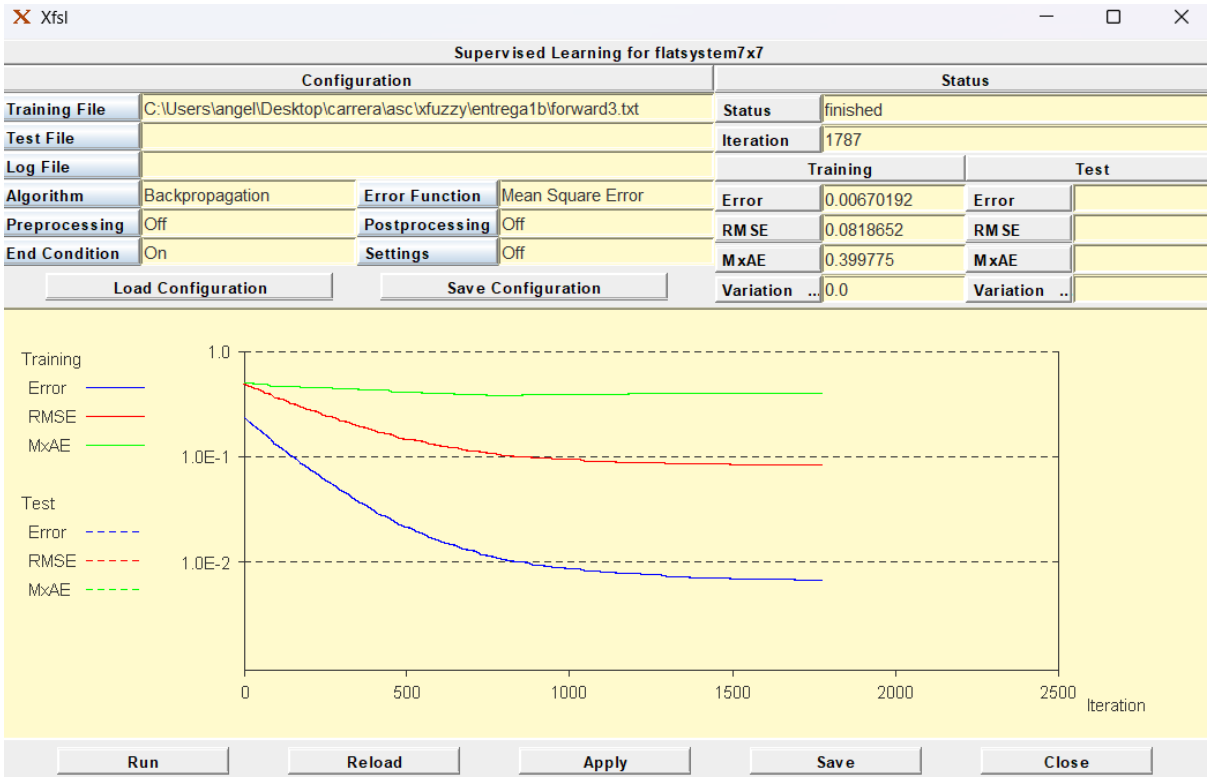
Finalmente, el flat system resulta ser el más lento, puesto que parte completamente desde cero y requiere un número mucho mayor de iteraciones para alcanzar un error similar.

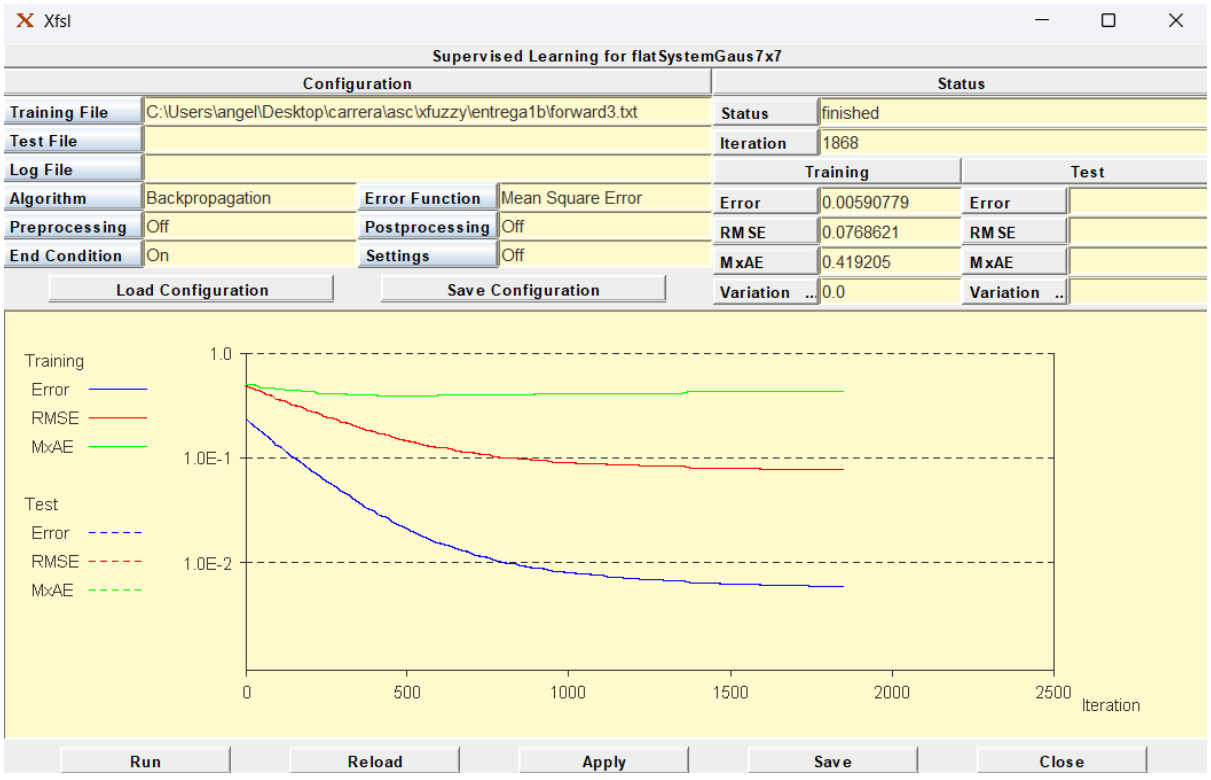
1.4 Comparativa de funciones triangulares vs gaussianas:





Las triangulares 5×5 funcionan mejor en este caso porque las transiciones abruptas de las triangulares se adaptan mejor a la resolución pequeña de la malla. Las gaussianas, aunque suaves, no capturan bien los detalles cuando las funciones son tan pocas.





Las gaussianas 7×7 ofrecen un mejor rendimiento porque su suavidad y mayor capacidad de adaptación permiten un ajuste más preciso. Las triangulares 7×7 todavía muestran un buen rendimiento, pero su comportamiento más rígido limita su capacidad para manejar transiciones suaves comparadas con las gaussianas.

7×7 mejora el ajuste en ambos tipos de funciones. Las triangulares 7×7 tienen más funciones de pertenencia, lo que les permite representar mejor los detalles y cambios, aunque sus transiciones

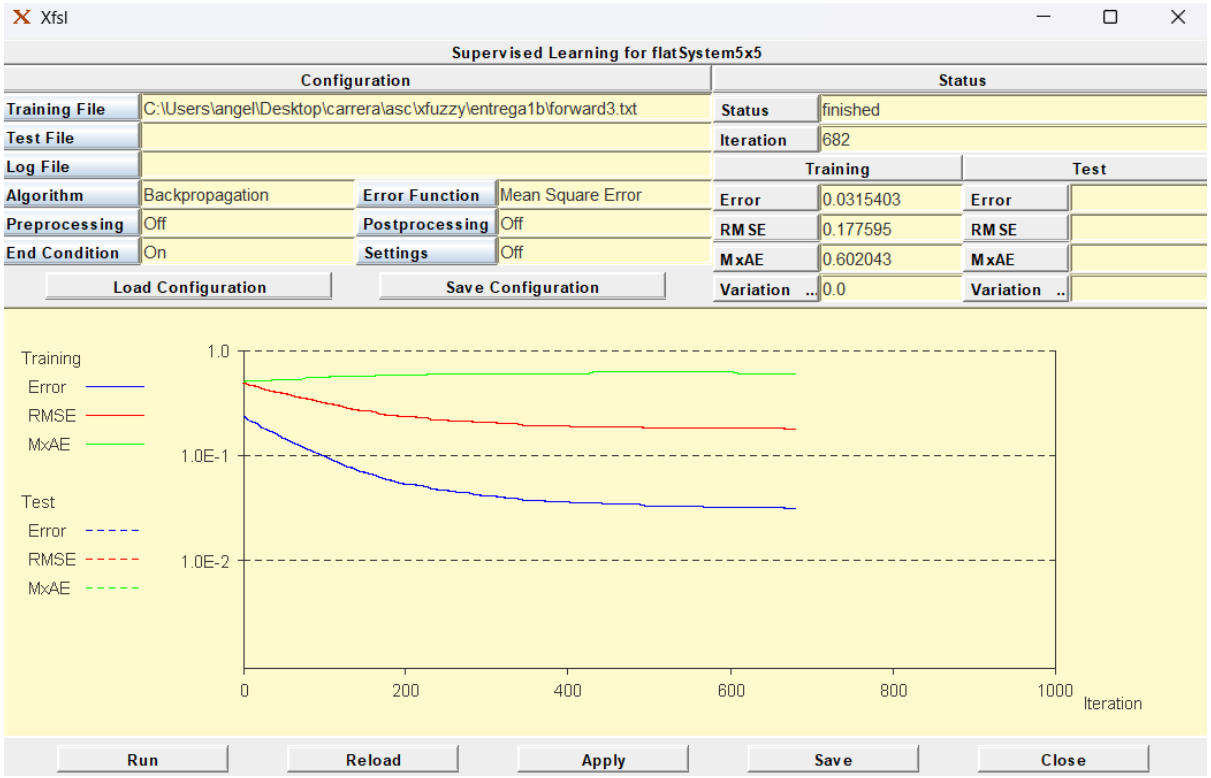
siguen siendo abruptas. Las gaussianas 7×7, al ser más suaves, modelan mejor las transiciones, lo que les da una ventaja con más puntos de control.

El aumento de la resolución permite una mejor cobertura del espacio de entrada y mejora la generalización. Para las gaussianas, esto suaviza las fronteras, mientras que, para las triangulares, mejora la representación de las transiciones.

Apartado 3:

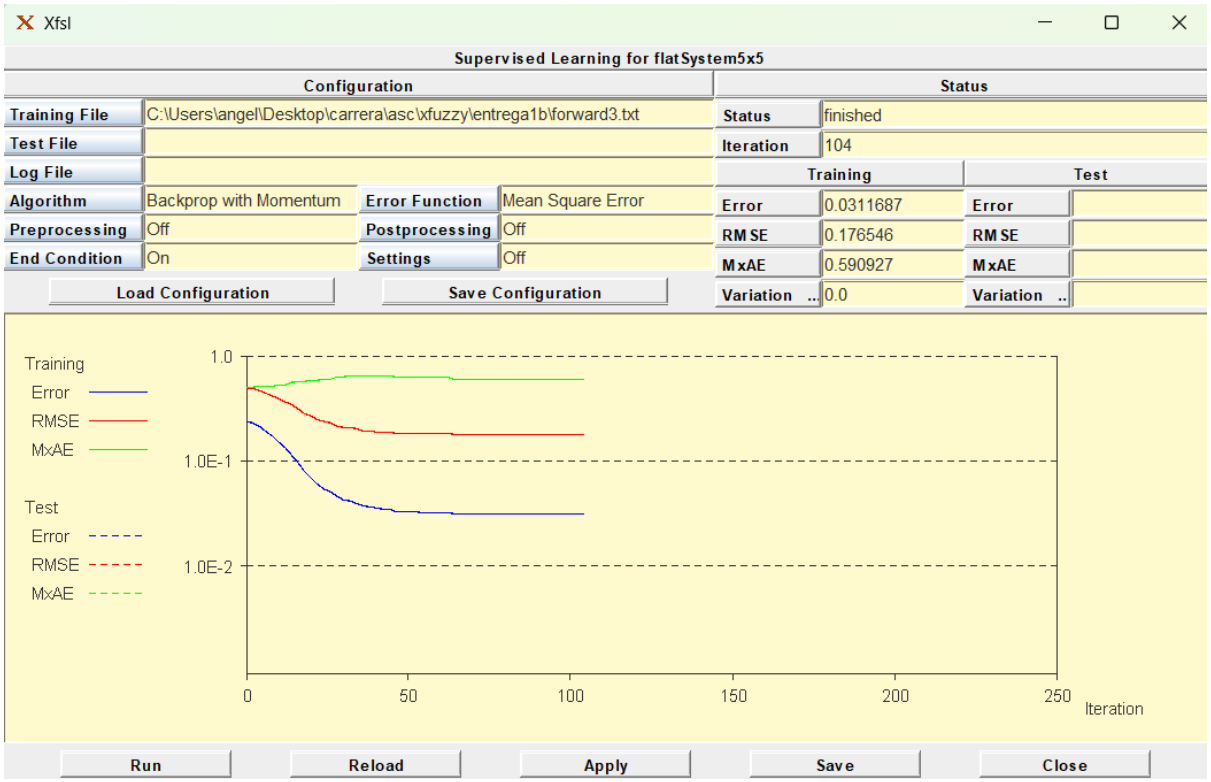
Para comparar los diferentes algoritmos de aprendizaje, realizaremos los aprendizajes sobre el sistema flatsystem5x5 y evaluaremos el comportamiento de los distintos algoritmos aplicando una tasa de aprendizaje de 0.05.

1.1 Aprendizaje con Backpropagation:



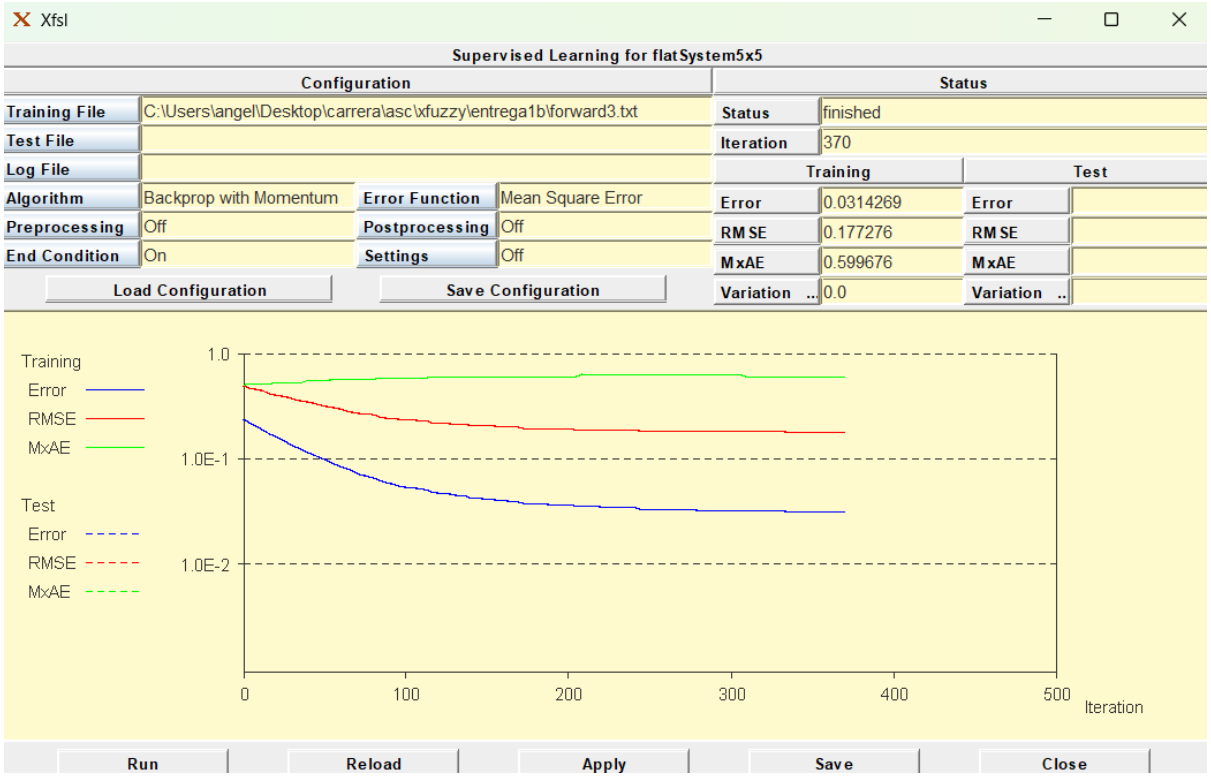
Con el algoritmo de *backpropagation*, detenemos la iteración en el paso 682, ya que al variar menos del 0.0001, nuestra condición de fin indica que el aprendizaje ha llegado a su límite. En ese punto, conseguimos un RMSE de 0.177595.

1.2 Aprendizaje con Backpropagation con Momentum:



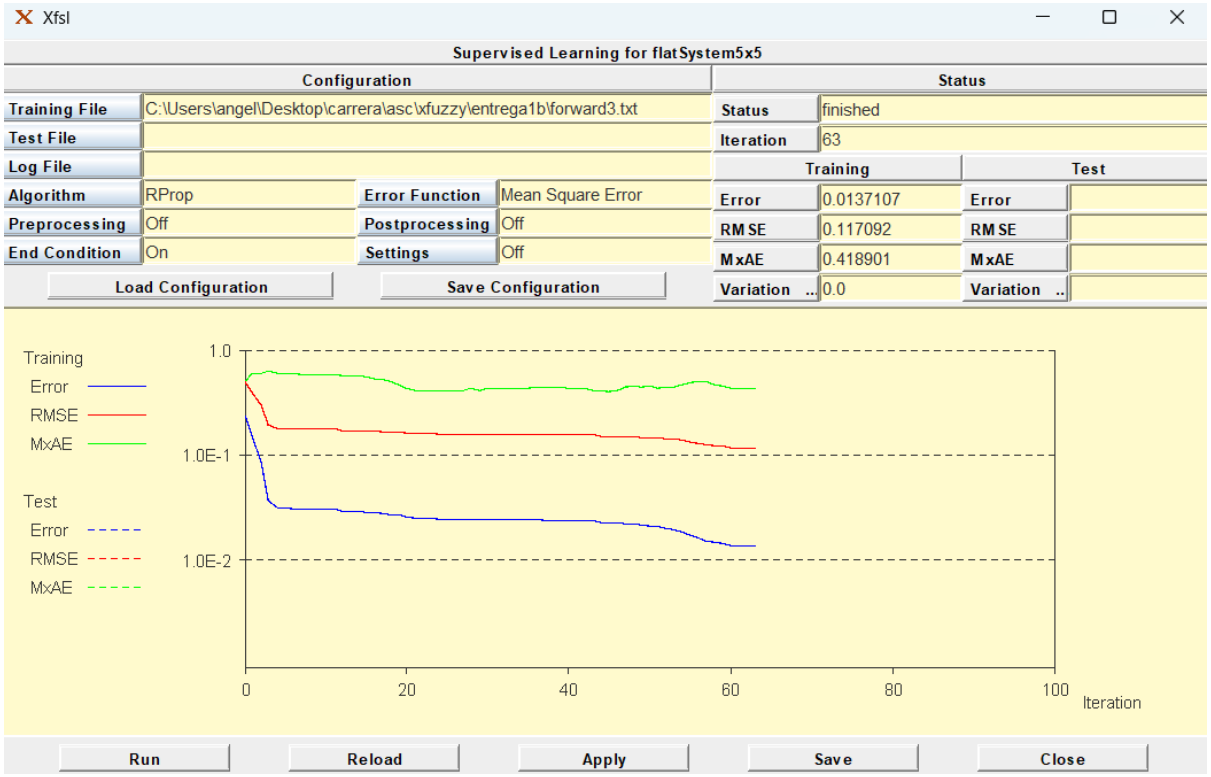
Primero, hemos probado con un momentum de 0.9, buscando una aceleración fuerte. El momentum se utiliza para acelerar el proceso de optimización, acumulando gradientes previos y ayudando al algoritmo a superar mínimos locales, lo que mejora la convergencia.

El RMSE obtenido con esta configuración es muy parecido al de Backpropagation normal, pero con muchas menos iteraciones, 104. Esto se debe a que el momentum acelera el proceso de aprendizaje al ajustar los parámetros más rápidamente, lo que permite alcanzar una solución óptima en menos pasos.



Con un momentum de 0.5, se obtiene un RMSE de 0.177, muy similar al caso anterior, pero requiere más iteraciones 370 para alcanzar ese valor. Esto se debe a que un momentum menor suaviza las actualizaciones, haciendo el aprendizaje más estable pero también más lento.

1.3 Aprendizaje con RProp:

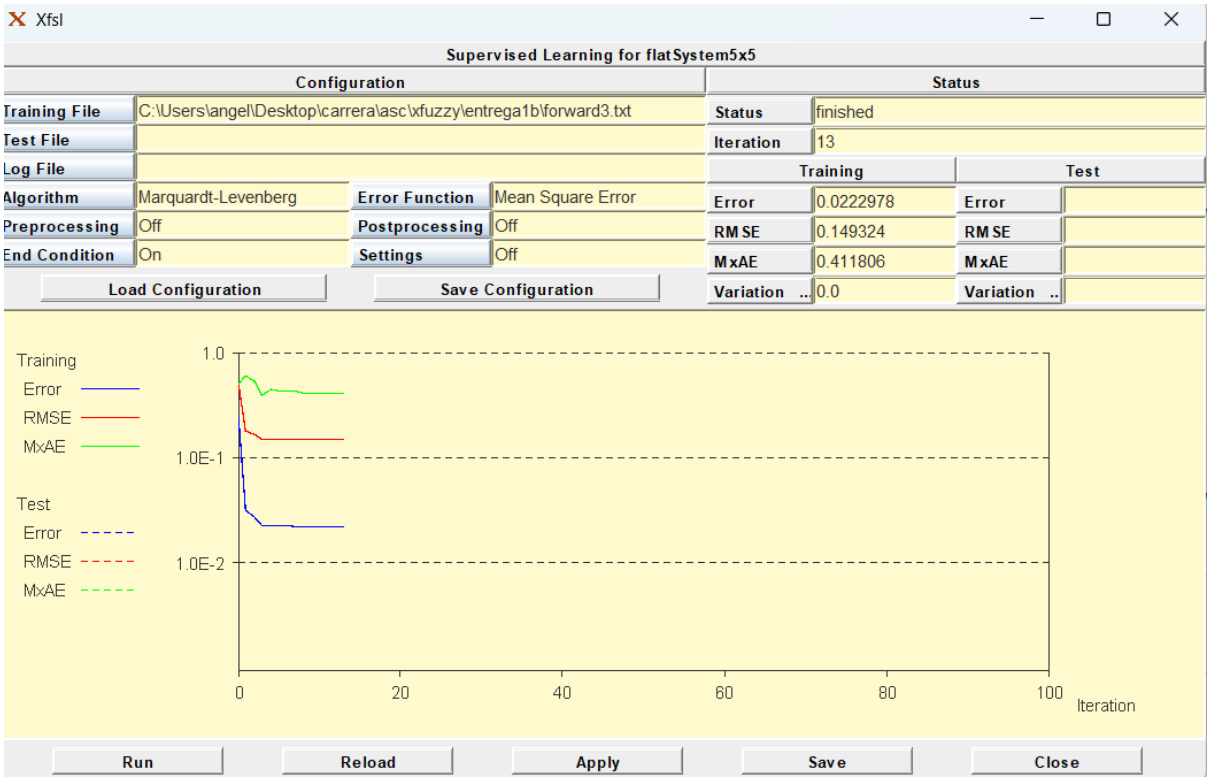


Para el algoritmo RProp, hemos utilizado los siguientes valores Initial Update 0.1, Increase Factor 1.2, Decrease Factor 0.5.

Con esta configuración, el entrenamiento finaliza en 63 iteraciones, alcanzando un RMSE de 0.117, lo que supone una mejora notable respecto a los algoritmos de Backpropagation y Backpropagation con Momentum, que necesitaban más iteraciones (682 y 104 respectivamente) para obtener errores similares o superiores.

Esto muestra que RProp converge mucho más rápido, ya que ajusta automáticamente el tamaño de los pasos en función del signo del gradiente, evitando los problemas de oscilación y lentitud asociados al uso de una tasa de aprendizaje fija. En conjunto, logra una convergencia más estable y eficiente, con un error más bajo en menos tiempo.

1.4 Aprendizaje con Marquardt–Levenberg:

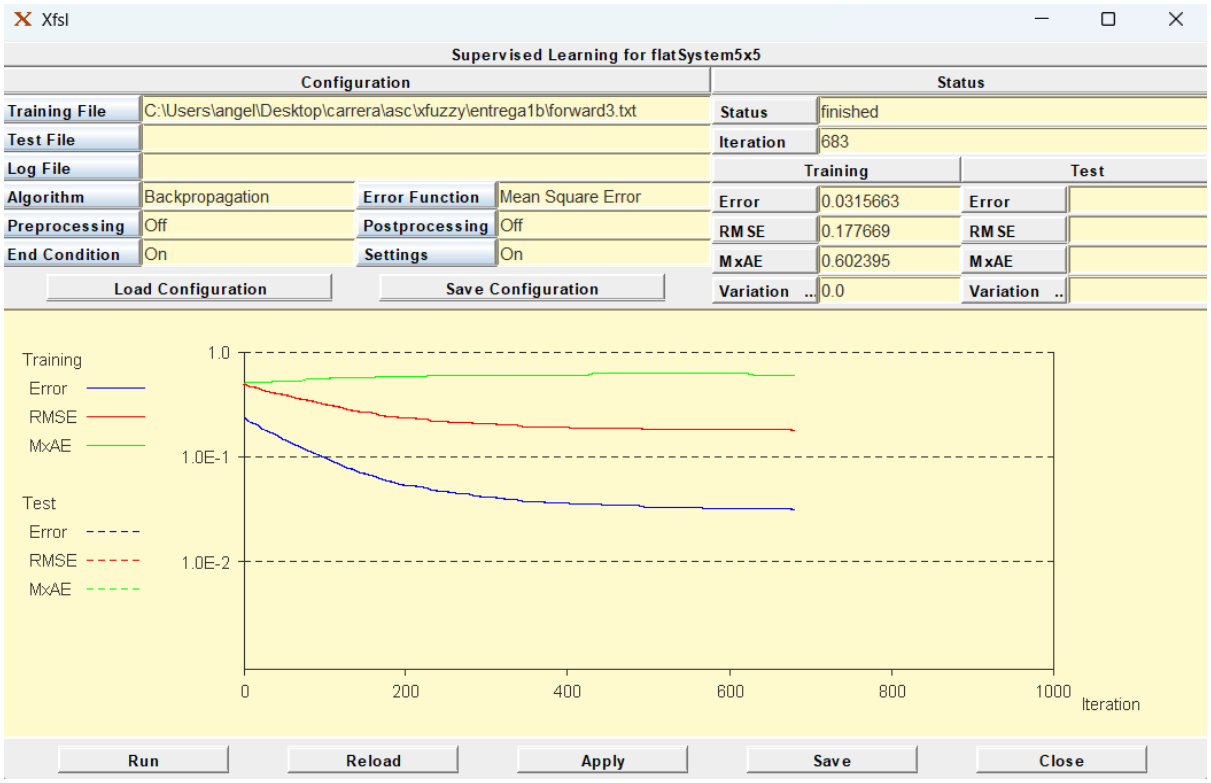


Parámetros usados Initial Hessian Addition 0.01, Increase Factor 10, Decrease Factor 0.1.

Con esta configuración, converge en 13 iteraciones, mucho más rápido que Momentum, Backpropagation normal y también más rápido que RProp. El RMSE es 0.1493, mejor que Backpropagation y Backpropagation +Momentum, aunque peor que RProp.

Marquardt–Levenberg utiliza información de segundo orden (una aproximación del Hessiano) y ajusta el parámetro de damping (λ) para adaptarse durante el aprendizaje. De esta forma, se comporta como un Gradient Descent cuando necesita más estabilidad y como un método de Newton cuando puede acelerar la convergencia. Gracias a esto, consigue una convergencia muy rápida, aunque con un RMSE intermedio en comparación con el obtenido por RProp.

1.5 Aprendizaje con Backpropagation solo consecuentes:



Solo consecuentes: RMSE 0.1777 en 683 iteraciones.

Todos los parámetros: RMSE 0.1776 en 682 iteraciones.

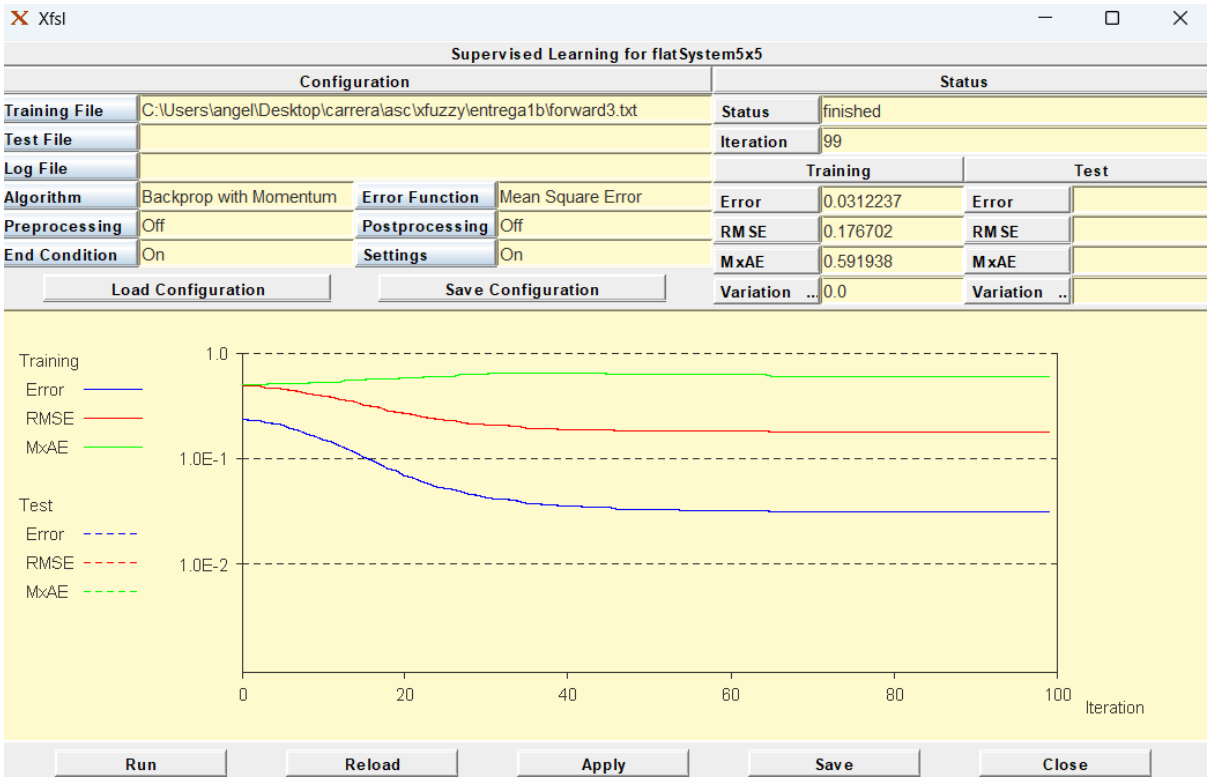
La diferencia de error es prácticamente nula. En este problema, afinar los antecedentes no aporta mejora.

1.6 Aprendizaje con Backpropagation con Momentum solo consecuentes:

Solo consecuentes: RMSE 0.1767 en 99 iteraciones.

Todos los parámetros: RMSE 0.1765 en 104 iteraciones.

El error de aproximación es muy similar en ambos casos, pero al aprender solo los consecuentes el algoritmo converge en prácticamente el mismo número de iteraciones, afinar los antecedentes no aporta mejora.

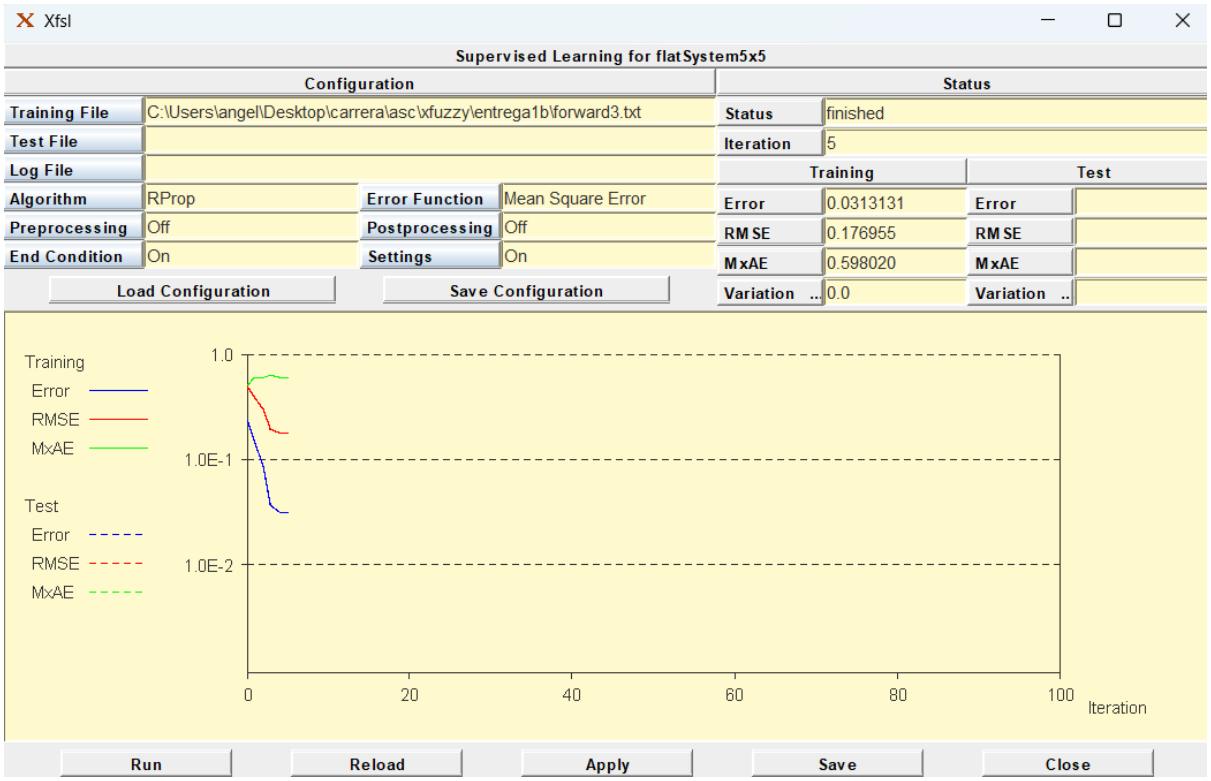


1.7 Aprendizaje con RProp solo consecuentes:

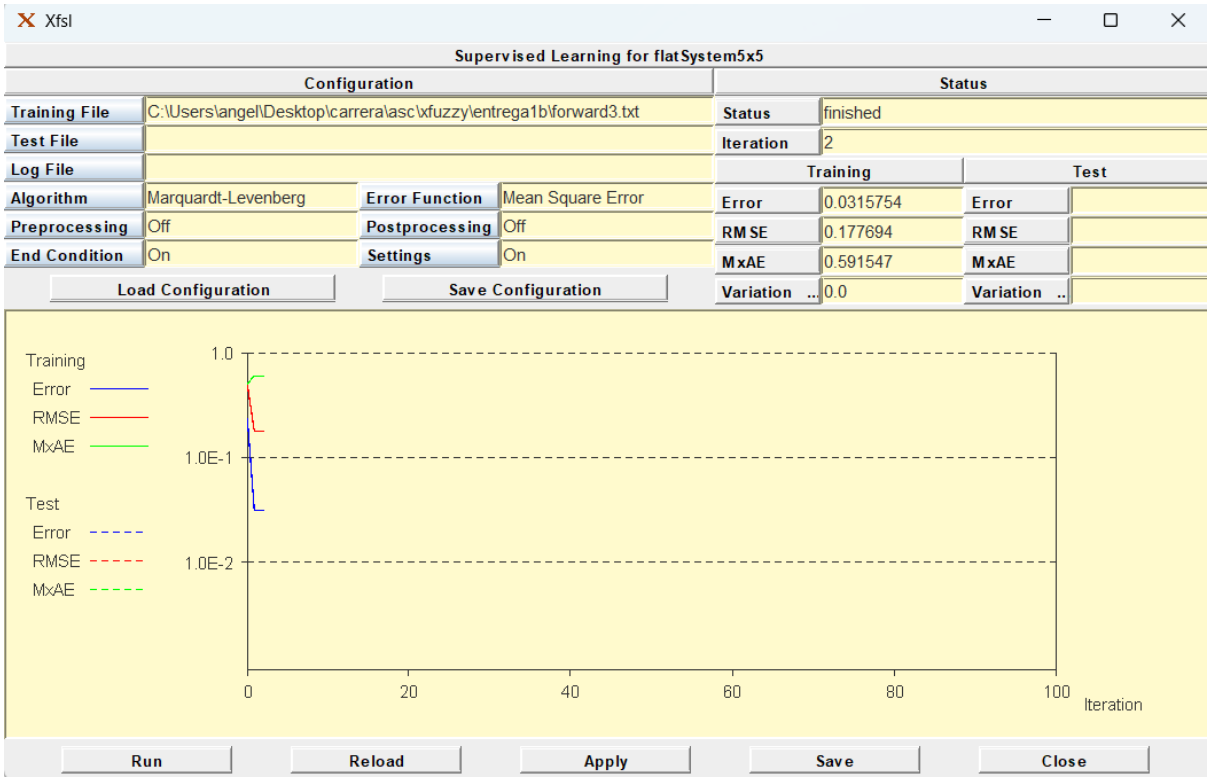
Solo consecuentes: RMSE 0.1769 en 5 iteraciones.

Todos los parámetros: RMSE 0.1171 en 63 iteraciones.

En este caso, la diferencia es más notable. Cuando el algoritmo RProp aprende todos los parámetros, el error de aproximación disminuye considerablemente, aunque requiere más iteraciones. Esto se debe a que RProp ajusta de forma independiente los pesos de cada parámetro según el signo del gradiente, aprovechando mejor la información adicional al optimizar también los antecedentes. Por tanto, RProp se beneficia claramente del aprendizaje completo, logrando un error menor con una convergencia aún rápida.



1.8 Aprendizaje con RProp solo consecuentes:



Solo consecuentes: RMSE 0.1777 en 2 iteraciones.

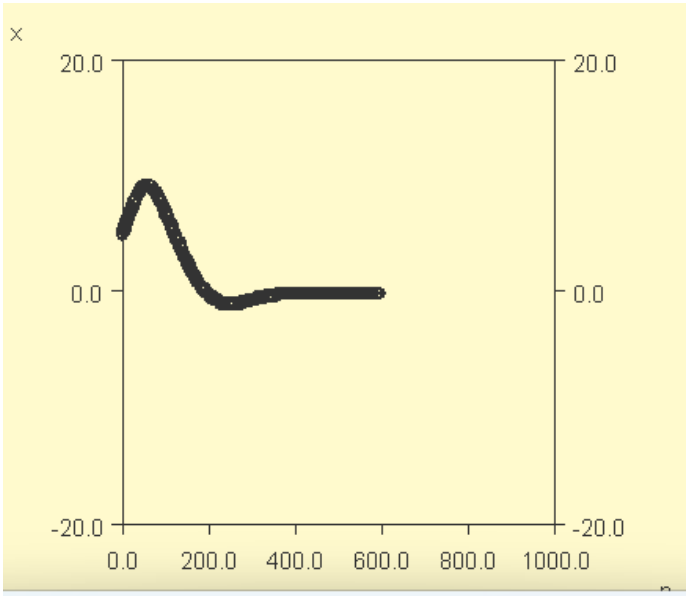
Todos los parámetros: RMSE 0.1493 en 13 iteraciones.

Al aprender solo los consecuentes, el algoritmo tiene un error de aproximación similar al de otros algoritmos, pero la convergencia es mucho más rápida. Sin embargo, cuando se aprenden todos los parámetros, el RMSE mejora significativamente, aunque con un ligero aumento en el número de iteraciones. Esto muestra que el ajuste de todos los parámetros mejora la precisión, pero no cambia drásticamente la rapidez del algoritmo.

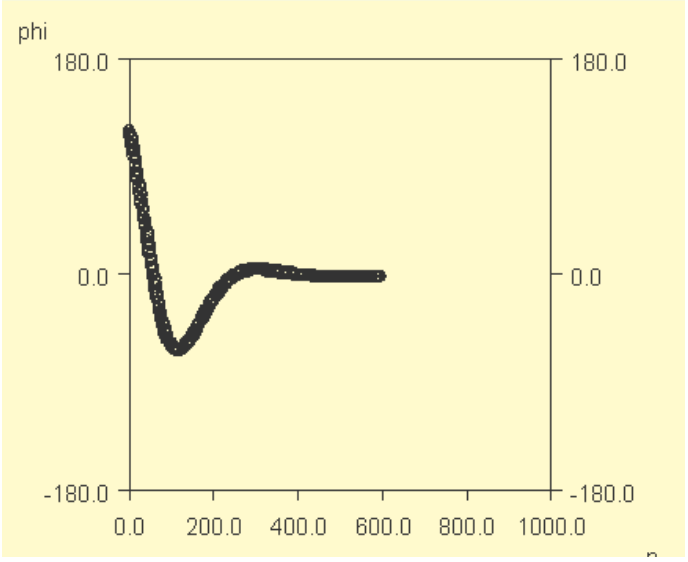
1.9 Simulación de los controladores:

Backpropagation normal:

En la iteración 300 ya está en x=0 y phi = 0.

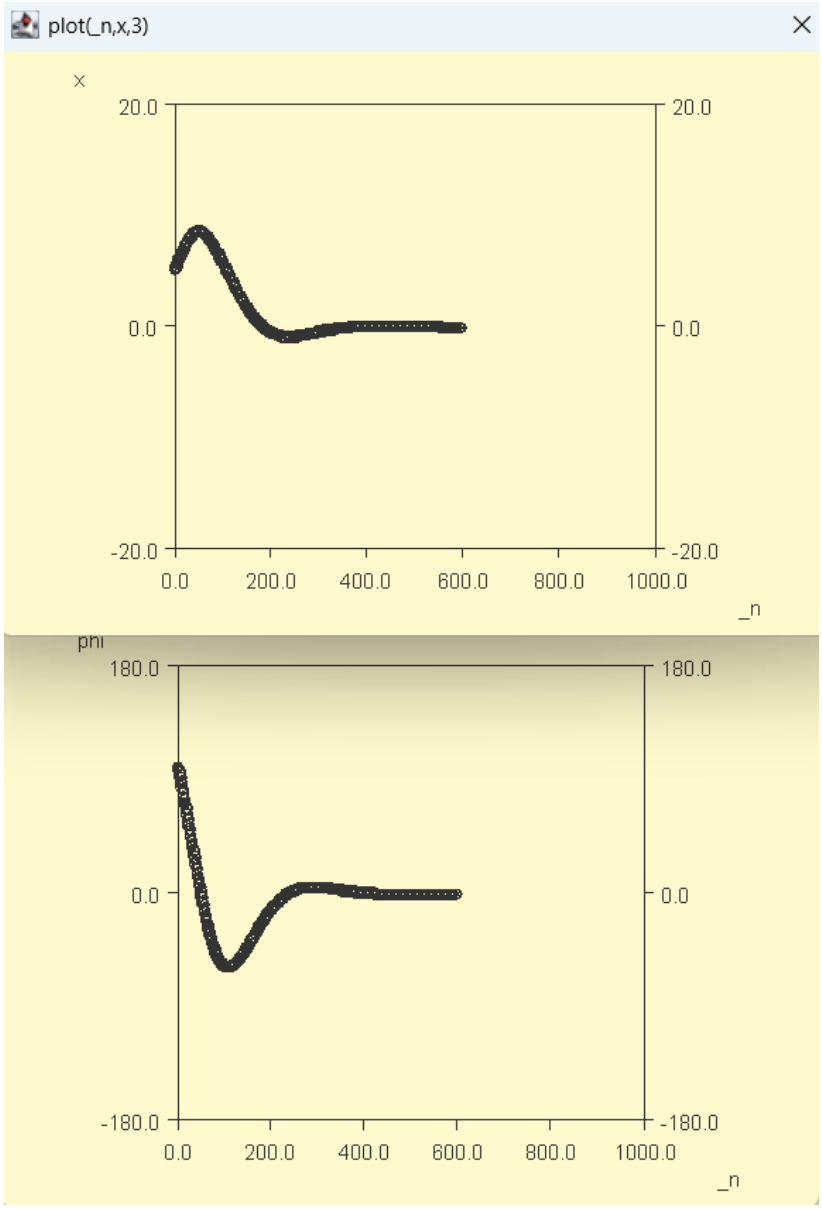


lot(_n,phi,3)



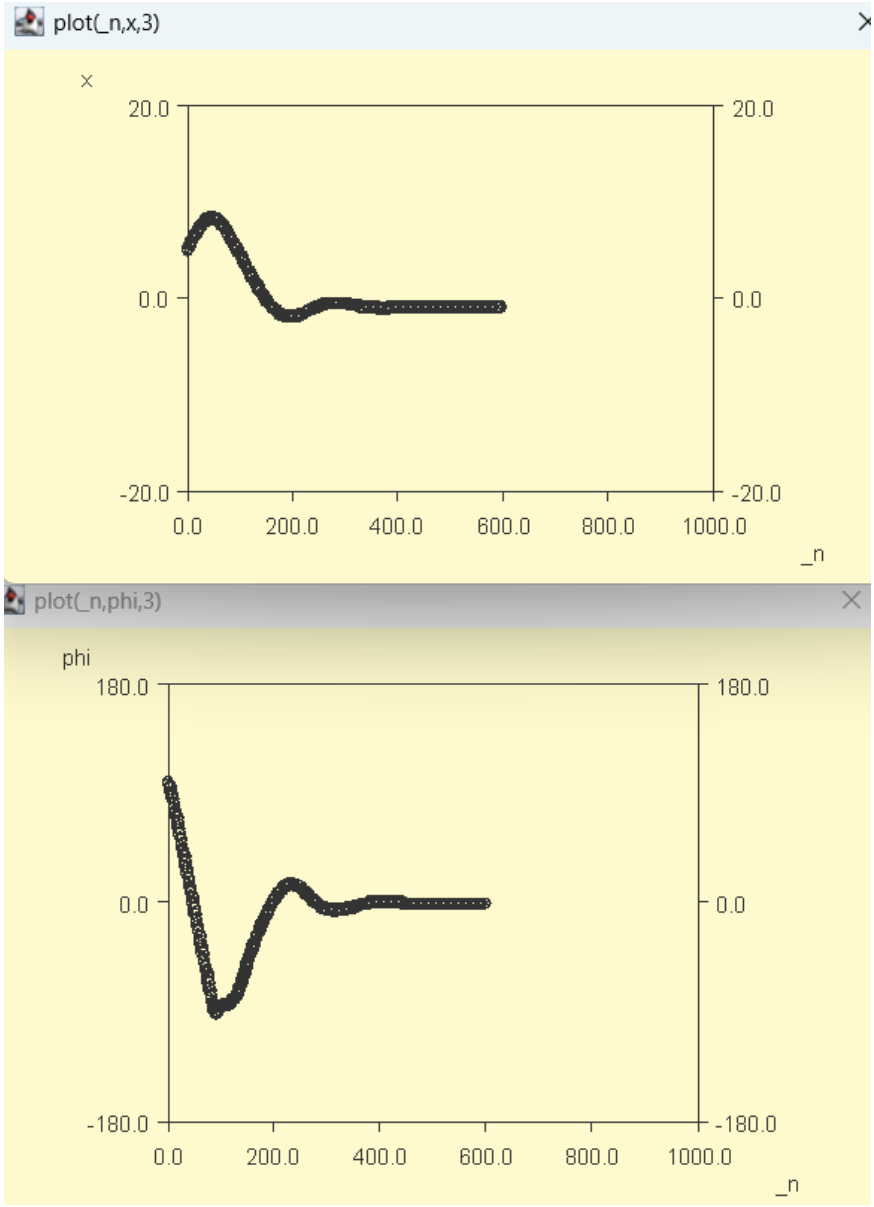
Backpropagation con momentum:

En la iteración 300 ya está en x=0 y phi = 0.



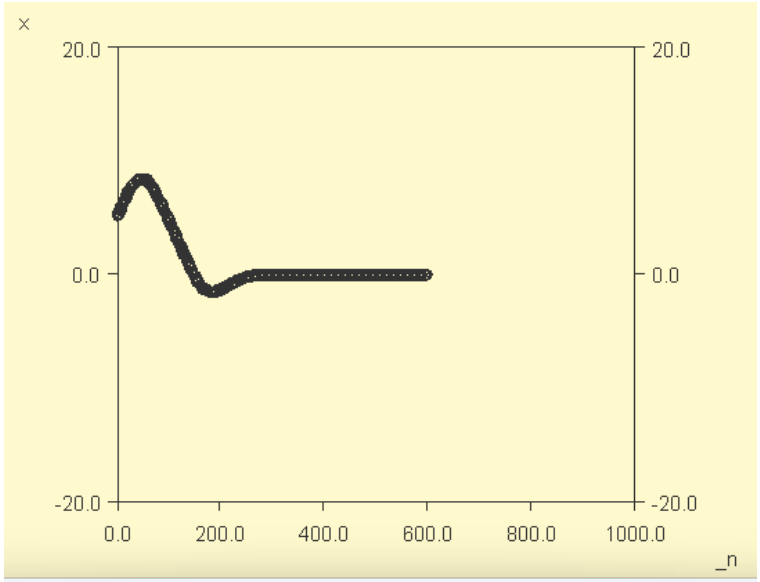
RProp:

RProp a pesar de tener un RMSE más bajo que los anteriores, como vemos no se mantiene estable en x = 0 y phi = 0 hasta la iteración 450 más o menos.

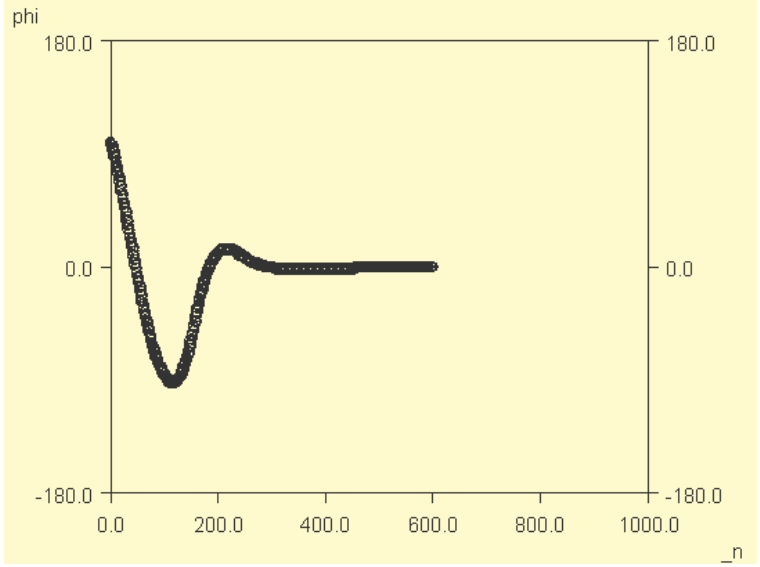


Marquardt Levenberg:

En la iteración 300 ya está en x=0 y phi = 0.



ot(_n,phi,3)

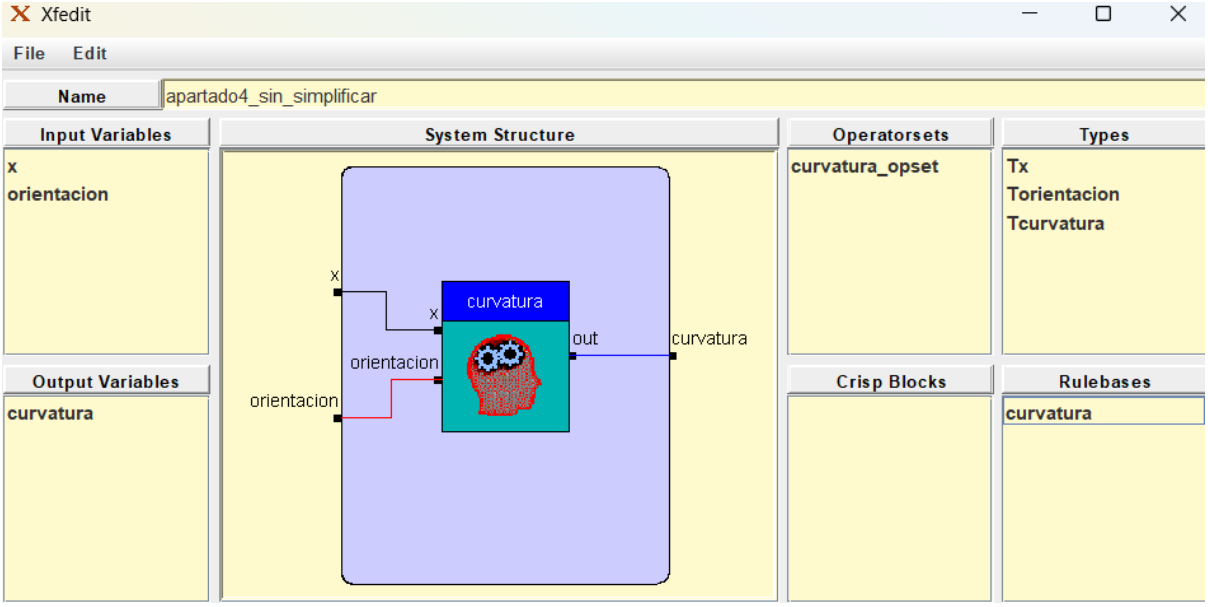


Aunque un error de aproximación pequeño es deseable, no siempre se traduce en una mayor eficiencia del controlador. Los algoritmos como Backpropagation y Backpropagation con Momentum tienen un error moderado, pero tardan más en estabilizarse. RProp, aunque logra un RMSE más bajo, muestra inestabilidad durante más iteraciones. En cambio, Marquardt-Levenberg, con un error intermedio, alcanza una estabilización más rápida y eficiente. Así, la velocidad y estabilidad de la convergencia son clave para un control eficiente, más allá del simple tamaño del error.

Apartado 4:

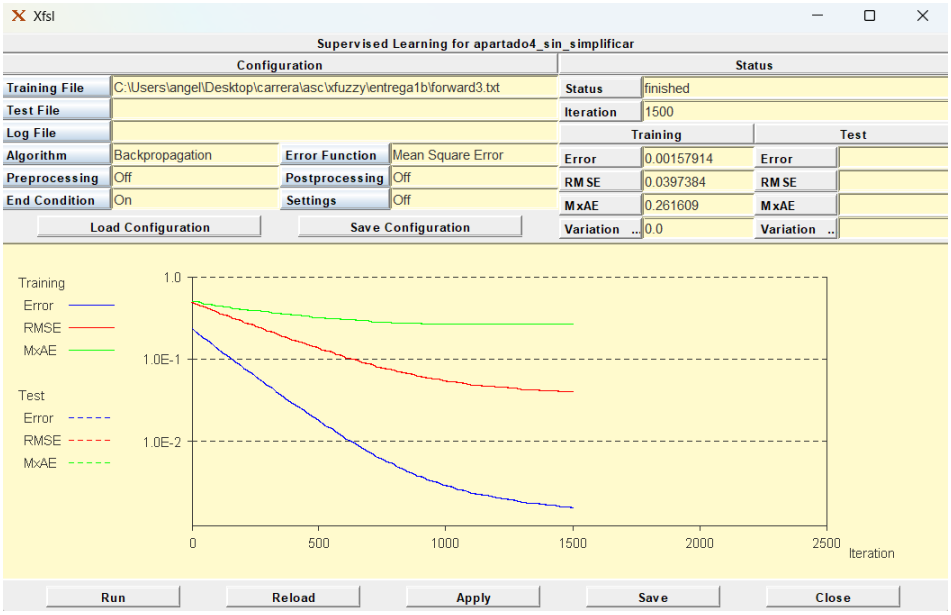
1.1 Creación del controlador base:

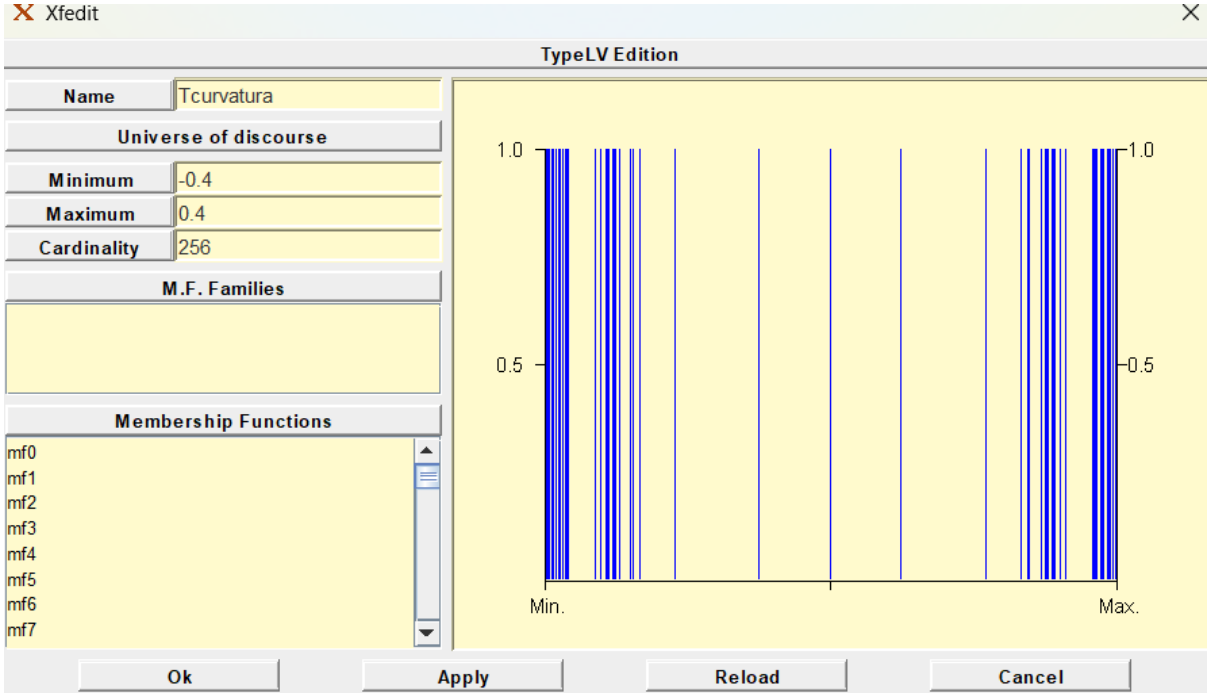
Como controlador base tenemos un flat system con 15 funciones para la x y para la orientacion, generando 225 reglas.



1.2 Entrenamiento del controlador:

Vamos a entrenarlo con Backpropagation



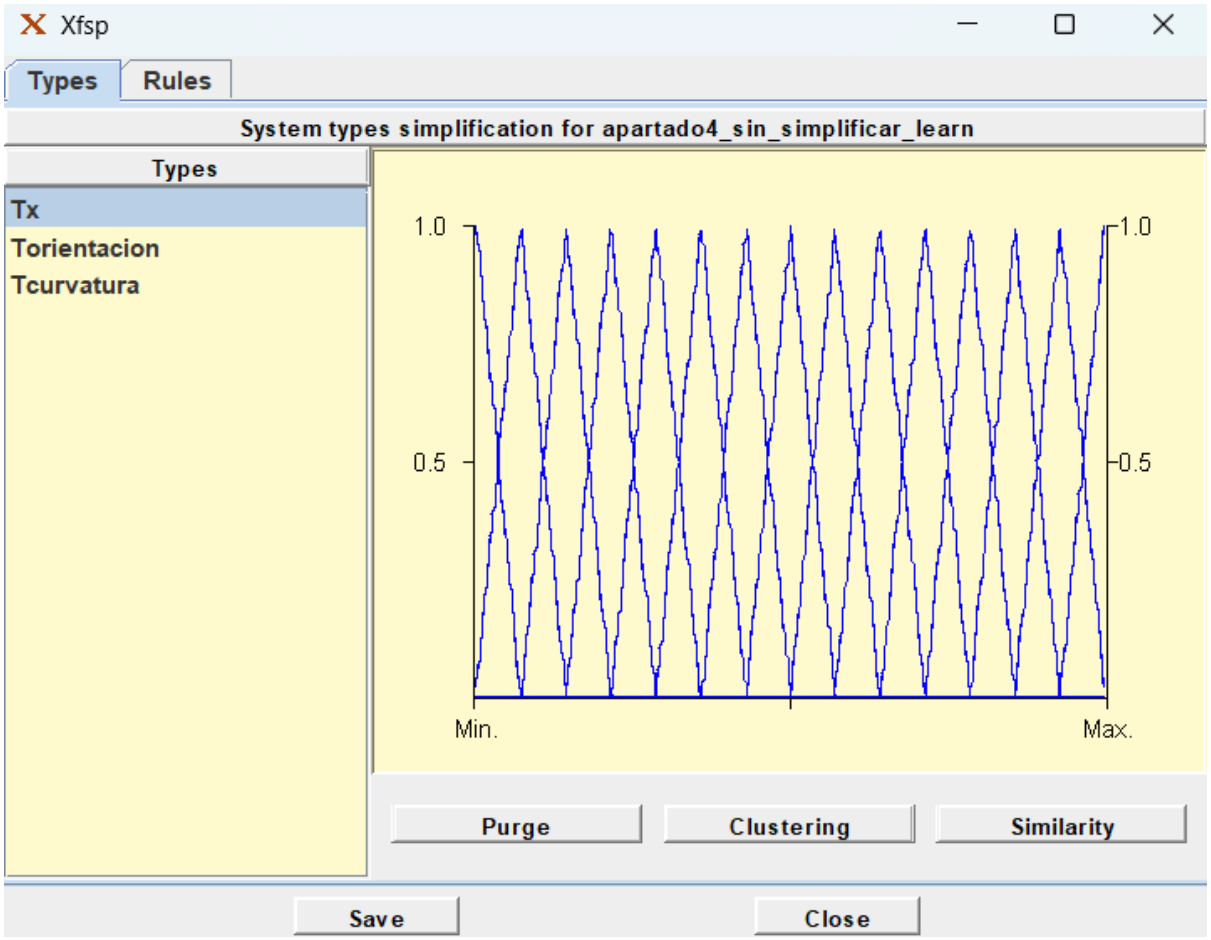


Como vemos nos ha modificado la salida distribuyendo las funciones.

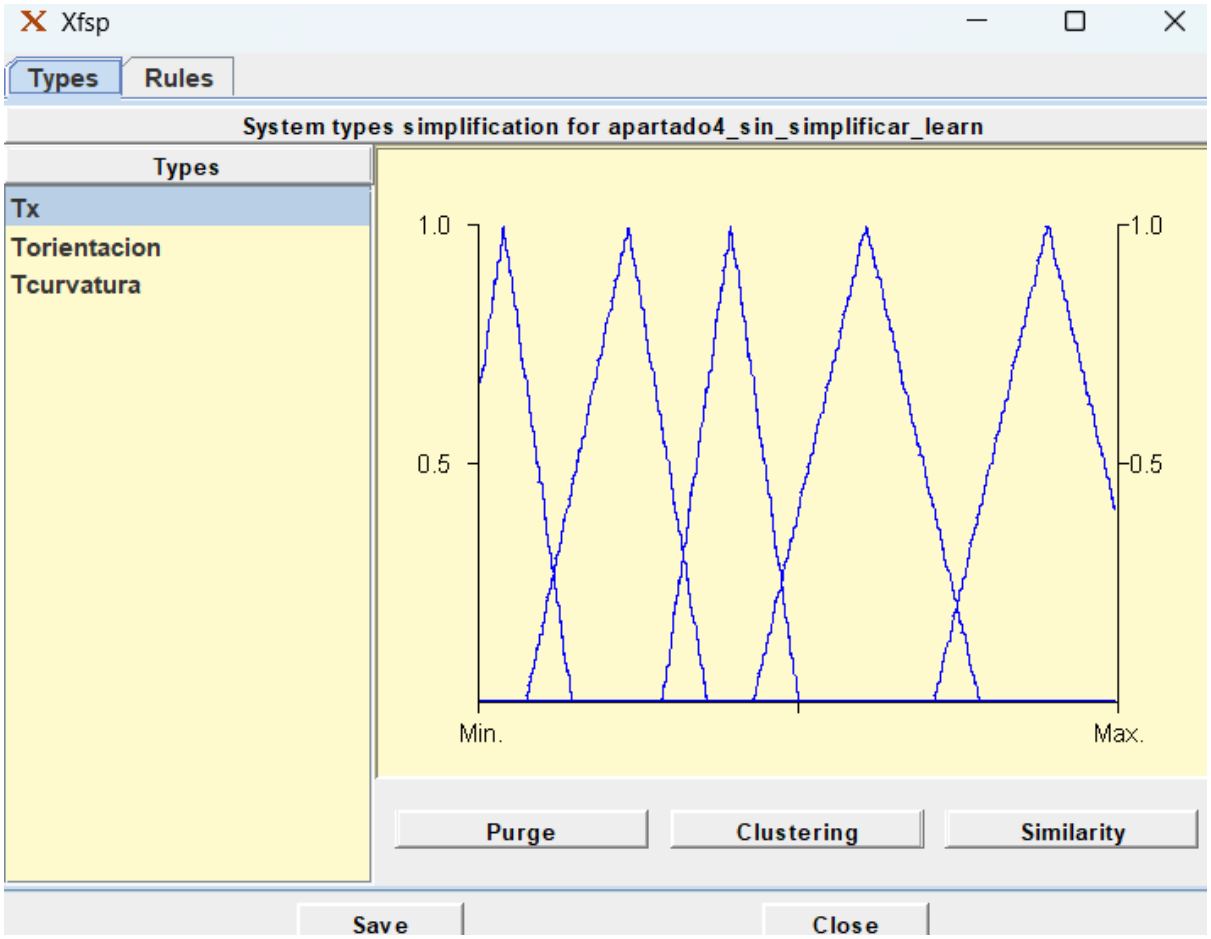
1.3 Simplificación del controlador:

Como vemos hay muchas funciones que hacen que el contralador tenga demasiada complejidad, asique vamos a simplificarlo.

Primero empezamos con Tx:

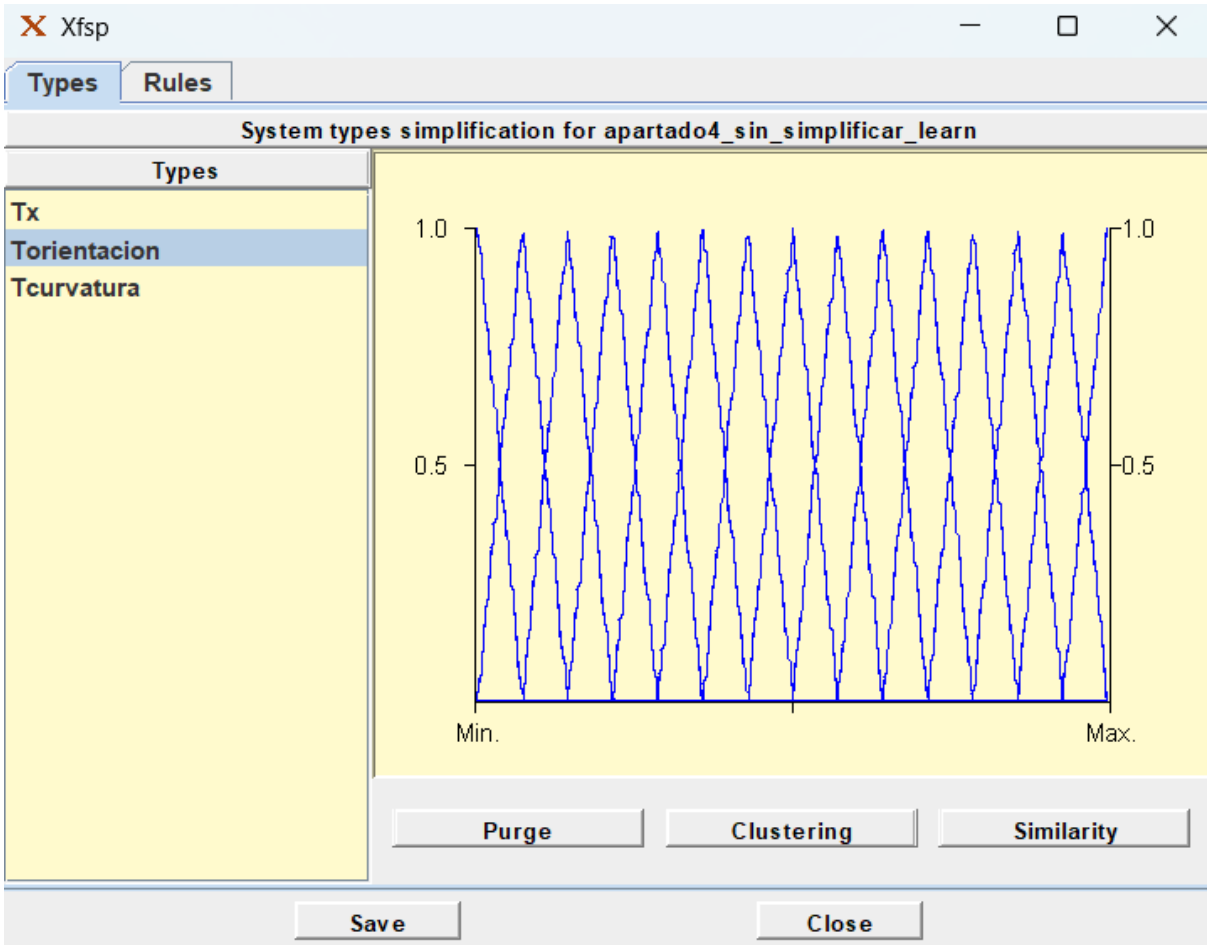


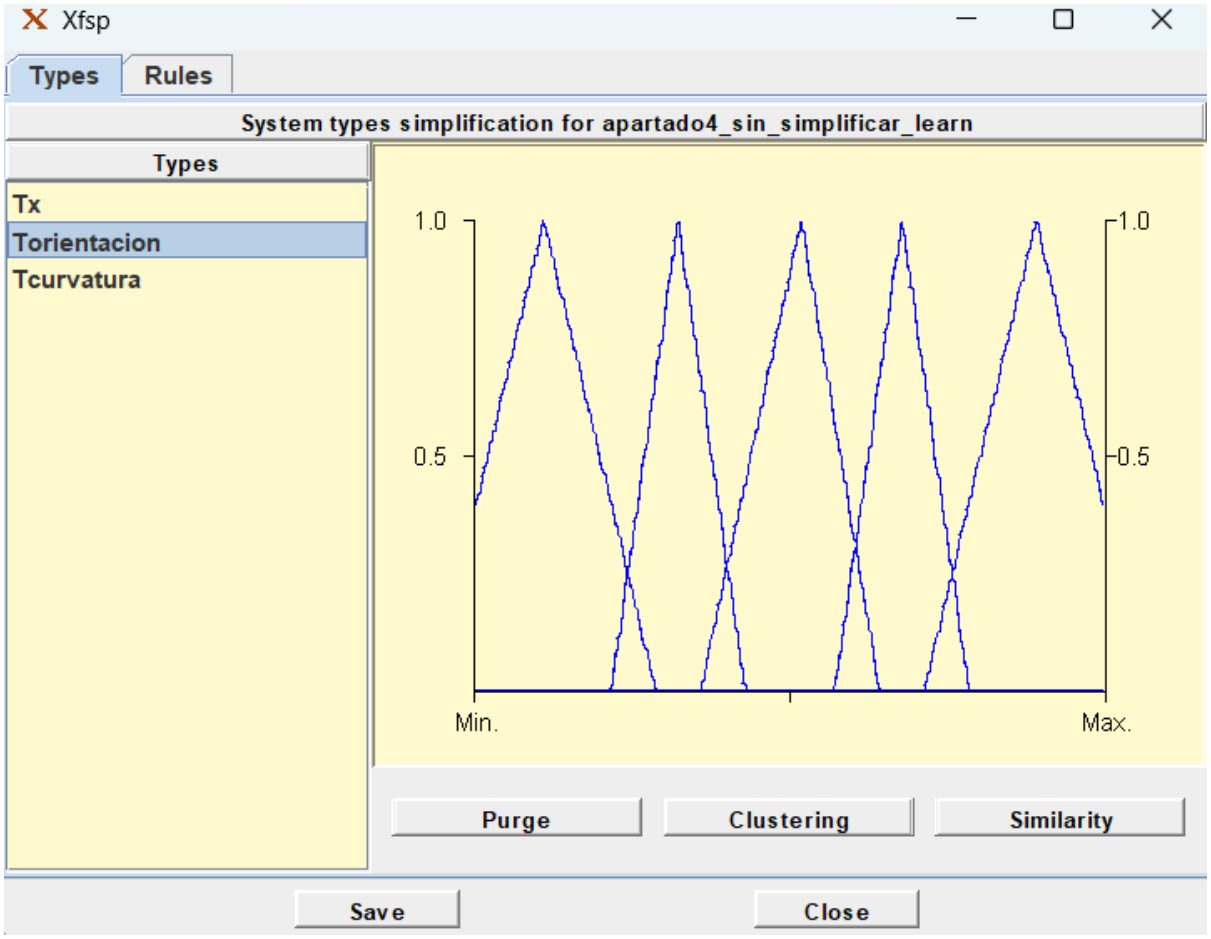
Aplicaremos similarity con factor 0.05, este proceso permite agrupar funciones fuzzy similares.



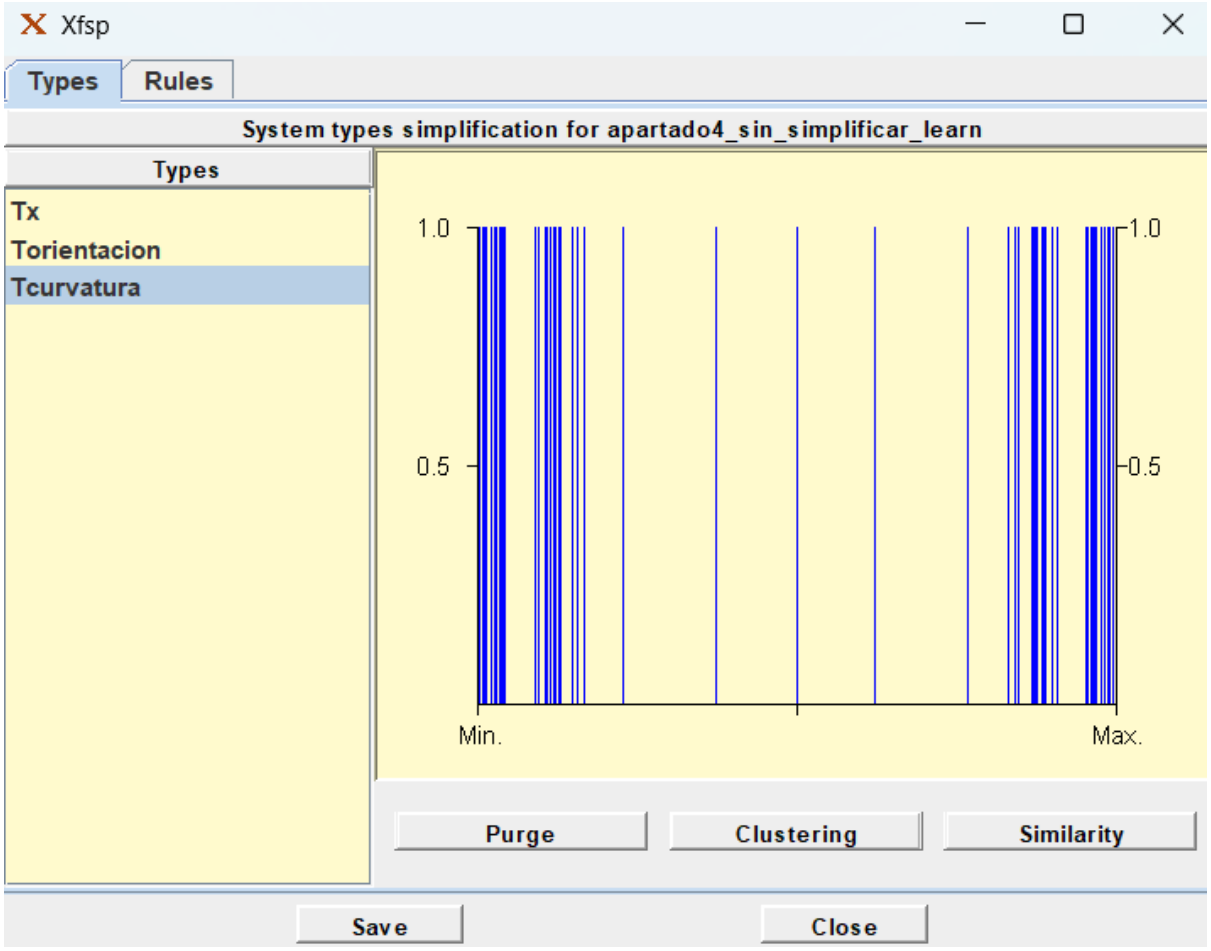
Después de aplicar la simplificación, el número de funciones se redujo a 5. El gráfico refleja las 5 funciones resultantes, que muestran un comportamiento similar al conjunto original, pero con menos complejidad.

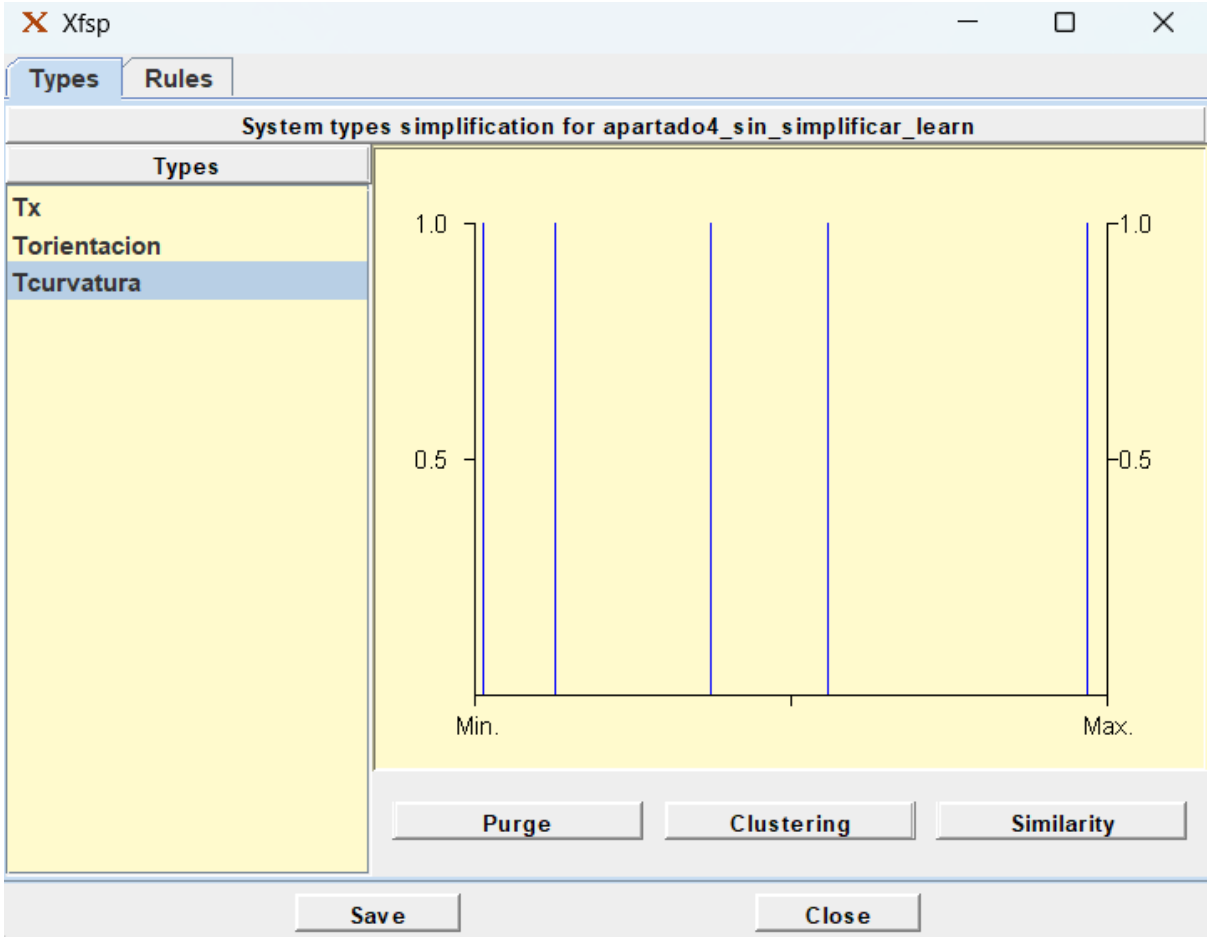
Para la orientación vamos a hacer lo mismo:


Xfsp
Types
Rules
System types simplification for apartado4_sin_simplificar_learn
Types
Tx
Torientacion
Tcurvatura
1.0
0.5
1.0
0.5
Min.
Max.
Purge
Clustering
Similarity
Save
Close



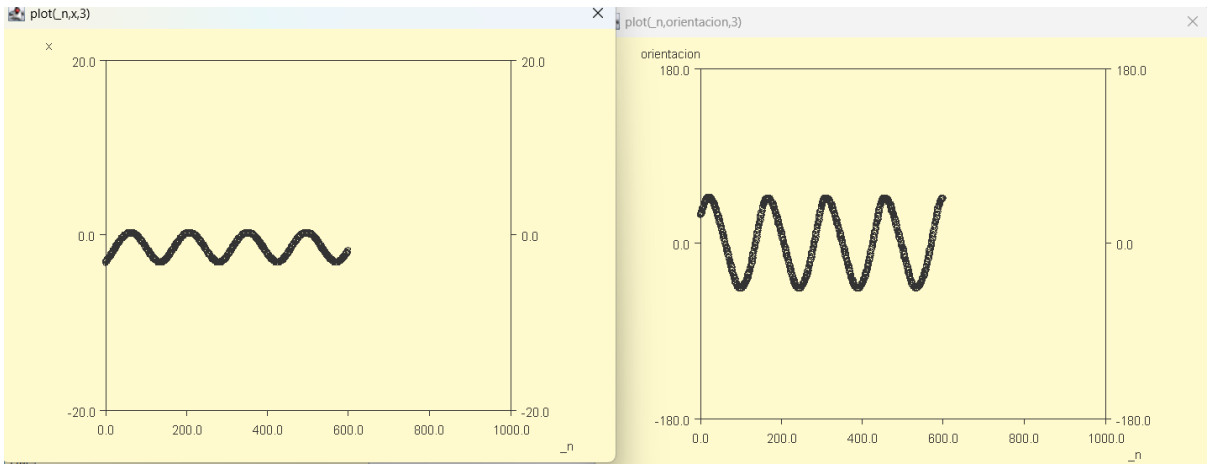
Y por último simplificamos la curvatura se aplicó el algoritmo de clustering con un número de 5 clusters. Esta técnica agrupa las funciones fuzzy similares, permitiendo que se conserven solo las representativas, mientras que las funciones redundantes o muy similares se combinan en un solo grupo.


Xfsp
Types
Rules
System types simplification for apartado4_sin_simplificar_learn
Types
Tx
Torientacion
Tcurvatura
1.0
0.5
1.0
0.5
Min.
Max.
Purge
Clustering
Similarity
Save
Close



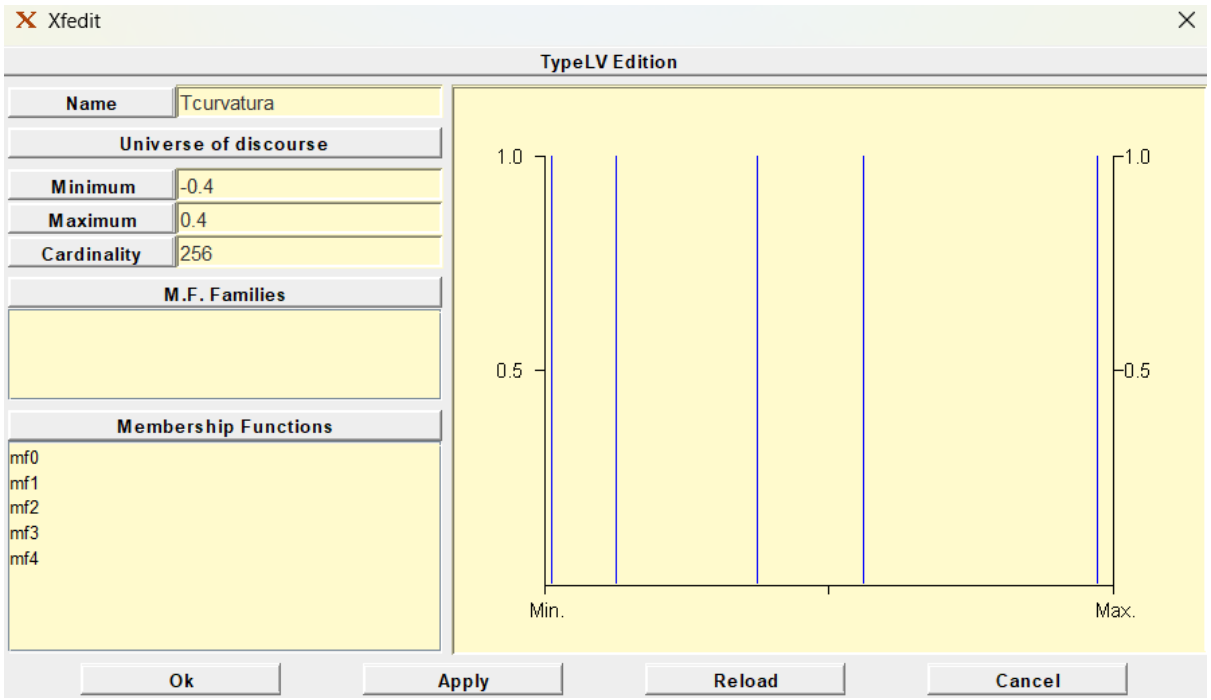
La aplicación de clustering ha simplificado las funciones de curvatura, reduciendo el número de funciones de forma significativa.

1.3 funcionamiento del controlador despues de simplificar:



El mal comportamiento observado podría ser consecuencia de una simplificación excesivamente agresiva. Al reducir el número de funciones de pertenencia, es posible que hayamos eliminado funciones clave que eran fundamentales para el comportamiento correcto del sistema. Es

recomendable evaluar más cuidadosamente el proceso de simplificación para equilibrar entre reducción de complejidad y preservación de las características esenciales del modelo.



Si nos fijamos en la curvatura, ahora no tenemos ninguna función que permita tener curvatura 0, lo cual cuadra con que no consiga quedarse sin oscilar.

Apartado 5:

1.1 Creación del modelo:

Cambios realizados en el modelo Modelforwarddin para crear el nuevo controlador Mimodelo2:

- Incorporación de la aceleración a como entrada: Se ha añadido la aceleración a como una variable que se actualiza en cada iteración de la simulación para afectar la velocidad y el movimiento del vehículo.
- Cálculo de la velocidad: El cálculo de la velocidad ahora se hace teniendo en cuenta la aceleración.
- Inicialización completa: El método init() ahora inicializa todas las variables del modelo, incluyendo la velocidad v, asegurando que el sistema comience con un estado válido.
- Estado completo: El estado ahora incluye no solo las variables de posición y orientación, sino también la velocidad v.

Quedando así nuestro modelo:

```
import xfuzzy.PlantModel;
```

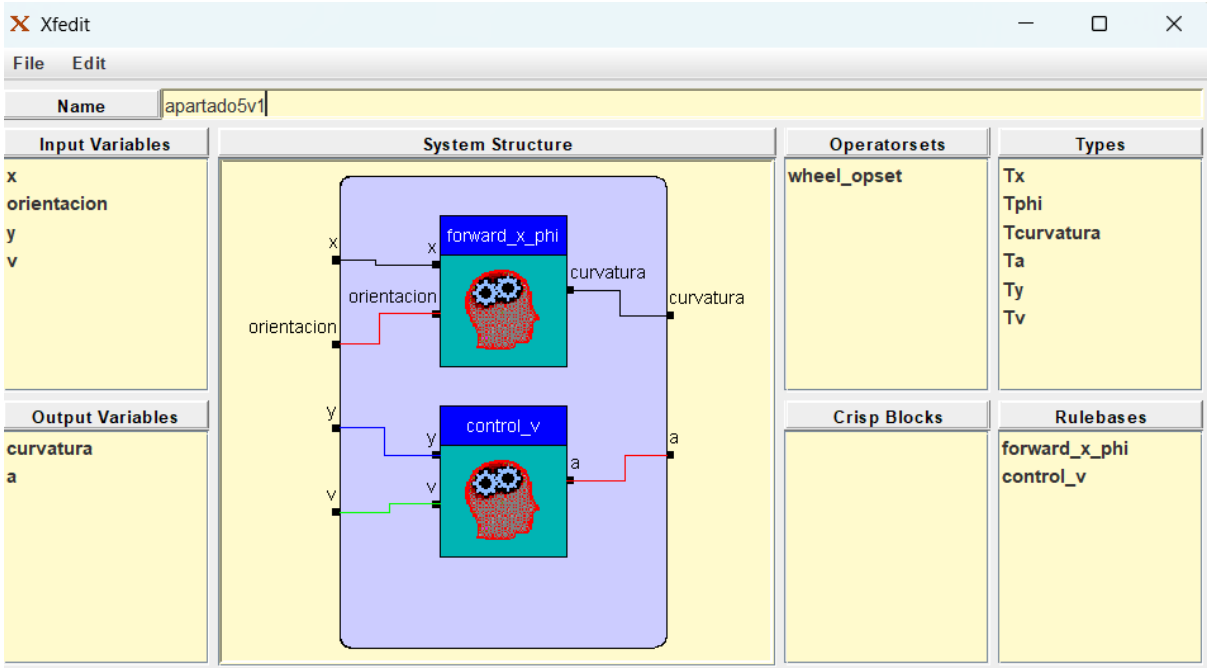
```
public class Mimodelo2 implements PlantModel {
    private double x;
    private double y;
    private double phi;
    private double v = 0.0;
    private double oldgamma;
    private double gamma;
    private double olda;
    private double a;

    public Mimodelo2() {
    }

    public void init() {
        x = 0;
        phi = 0;
        y = 0;
        v = 0;
    }

    public void init(double val[]) {
        x = val[0];
        phi = val[1] * Math.PI / 180;
        y = val[2];
        v = val[3];
    }

    public double[] state() {
        double state[] = new double[4];
        state[0] = x;
        state[1] = phi * 180 / Math.PI;
```

```
    state[2] = y;
    state[3] = v;
    return state;
}

public double[] compute(double val[]) {
    double LAPSE = 0.1;
    double P_TAU = 0.5; // 1.8
    double t = 0.0;
    double gref = 1.0 * val[0];
    olda = a;
    oldgamma = gamma;

    for (t = 0.0; t <= LAPSE; t += 0.001) {
        x += v * Math.sin(phi) * 0.001;
        y += v * Math.cos(phi) * 0.001;
        phi += v * gamma * 0.001;

        if (phi > Math.PI) phi -= 2 * Math.PI;
        if (phi < -Math.PI) phi += 2 * Math.PI;

        gamma = gref + (oldgamma - gref) * Math.exp(-t / P_TAU);
        if (gamma > 0.4) gamma = 0.4;
        if (gamma < -0.4) gamma = -0.4;

        a = olda + (val[1] - olda) * Math.exp(-t / P_TAU);
        if (a > 5) a = 5;
        if (a < -5) a = -5;

        v += a * 0.001;

    }
```

```
        return state();
    }
}
```

1.2 Creación del controlador:

Controlador apartado5v1:



El sistema tiene cuatro variables de entrada:

- x (Tipo Tx):

  Rango: [-10.0, 10.0]

  Términos lingüísticos (5): LB, LS, ZE, RS, RB (Funciones triangulares).

- orientación (Tipo Tphi):

  Rango: [-180.0, 180.0]

  Términos lingüísticos (5): LB, LM, ZE, RM, RB (Funciones triangulares y trapezoidales).

- y (Tipo Ty):

  Rango: [-2.0, 64.2]

Términos lingüísticos (3): L, C, O (Funciones triangulares).

- v (Tipo Tv):

  Rango: [0.0, 5.0]

  Térmimos lingüísticos (3): C, L, R (Funciones triangulares).

El sistema tiene dos variables de salida. Ambas usan singletons para sus conjuntos difusos:

- curvatura (Tipo Tcurvatura):

  Rango: [-0.4, 0.4]

  Términos lingüísticos (5): NB, NM, ZE, PM, PB (Valores singleton).

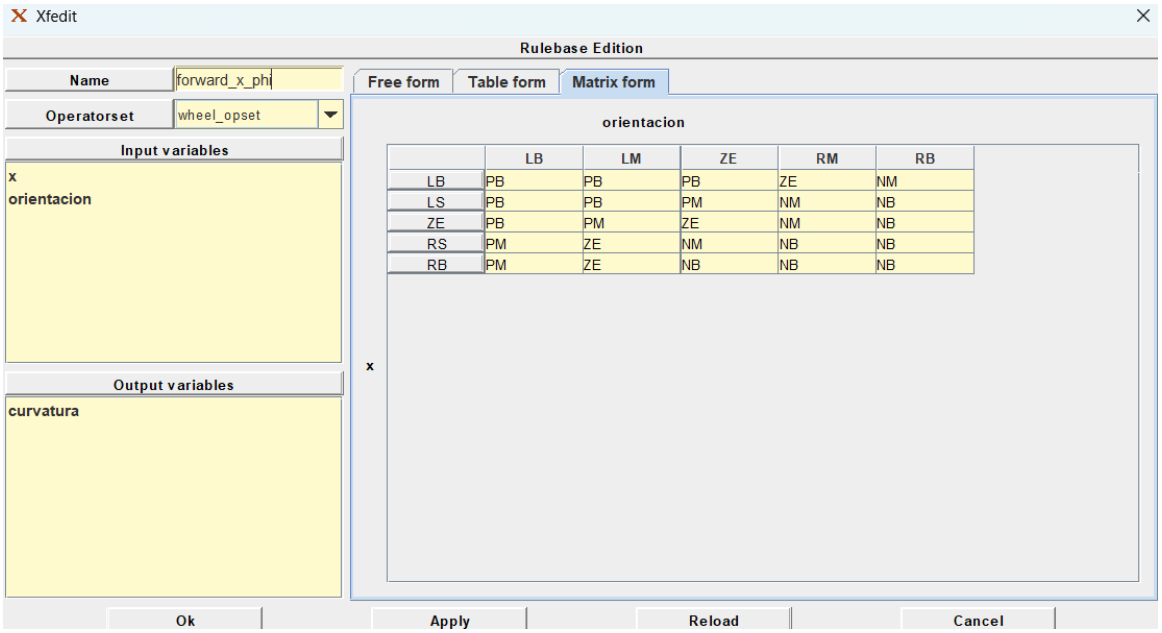
- a (Tipo Ta):

  Rango: [-5.0, 5.0]

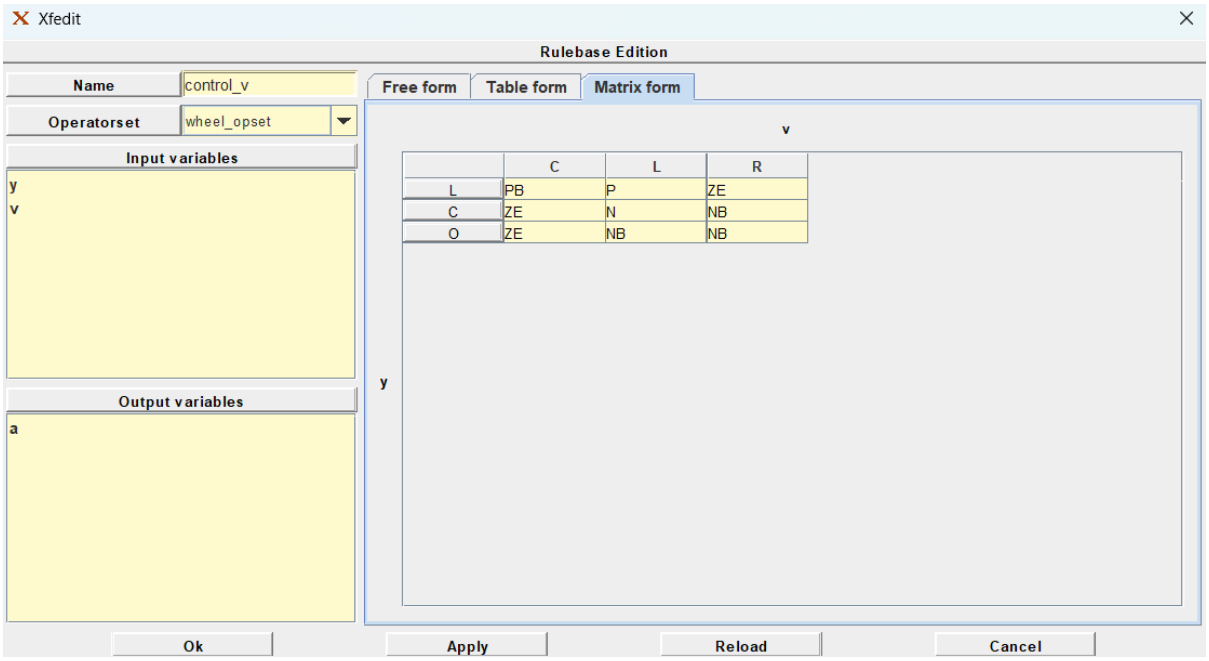
  Términos lingüísticos (5): NB, N, ZE, P, PB (Valores singleton).

El controlador difuso consta de dos bases de reglas:

- forward_x_phi: Controla la navegación, utilizando la posición lateral x y la orientación phi para determinar la curvatura de salida.

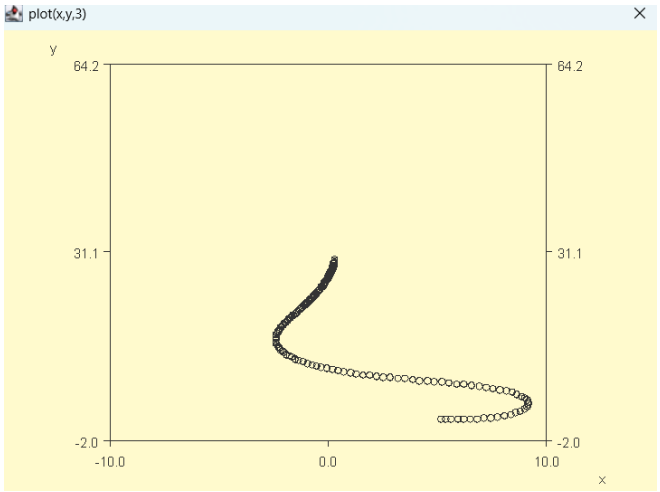


- control_v: Gestiona la velocidad y el frenado, usando la velocidad v y la posición longitudinal y para calcular la aceleración a. Su objetivo es conseguir la detención en y = 30.



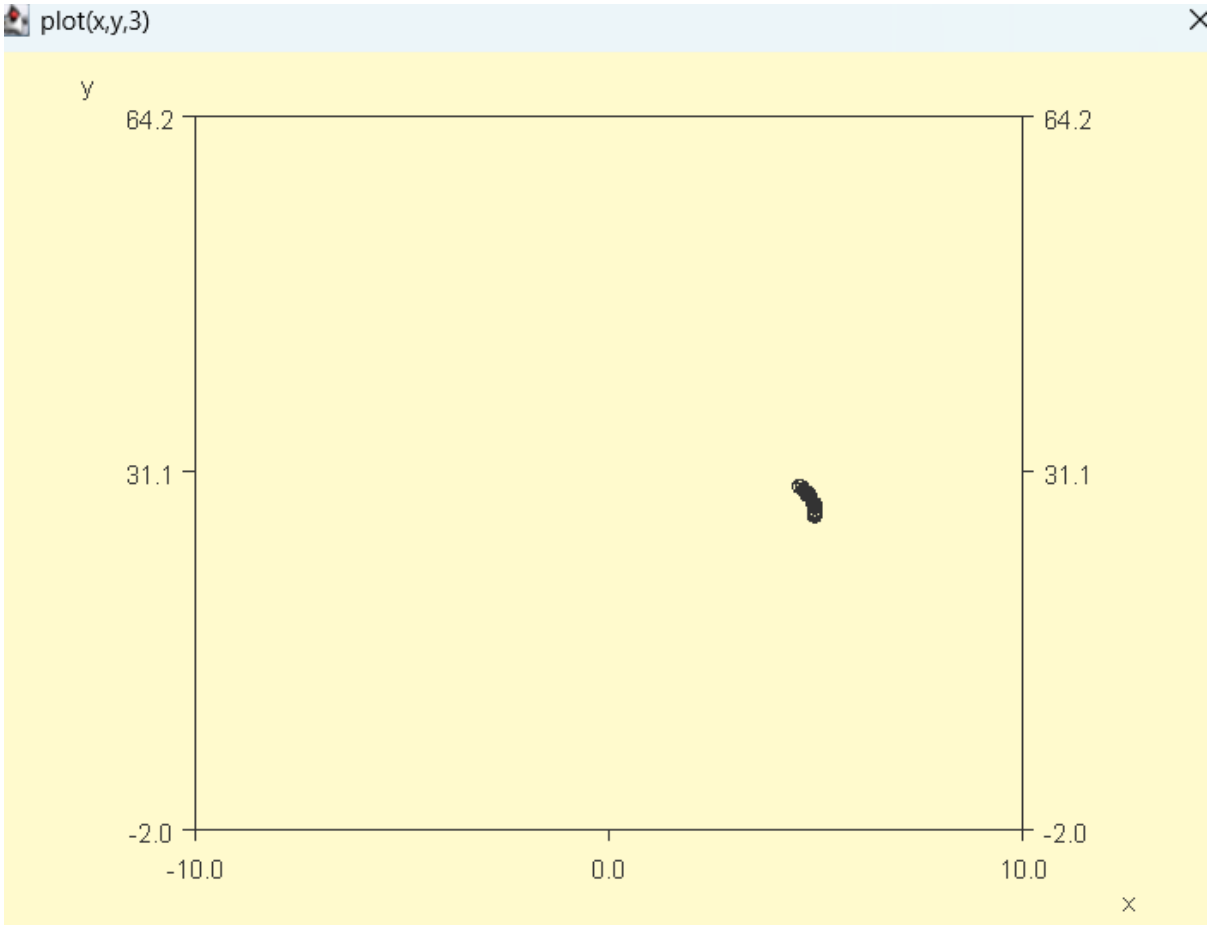
Originalmente, la variable de entrada y se definió en el rango [-2.0, 30.0]. Esto causaba un frenado prematuro, con la detención ocurriendo cerca de y = 14.

Para corregir esto, se amplió el rango de la variable y de forma proporcional hasta [-2.0, 64.2]. Este ajuste reubicó el punto de frenado dentro de la lógica del sistema, logrando que la detención ocurriera muy cerca del objetivo de y = 30.



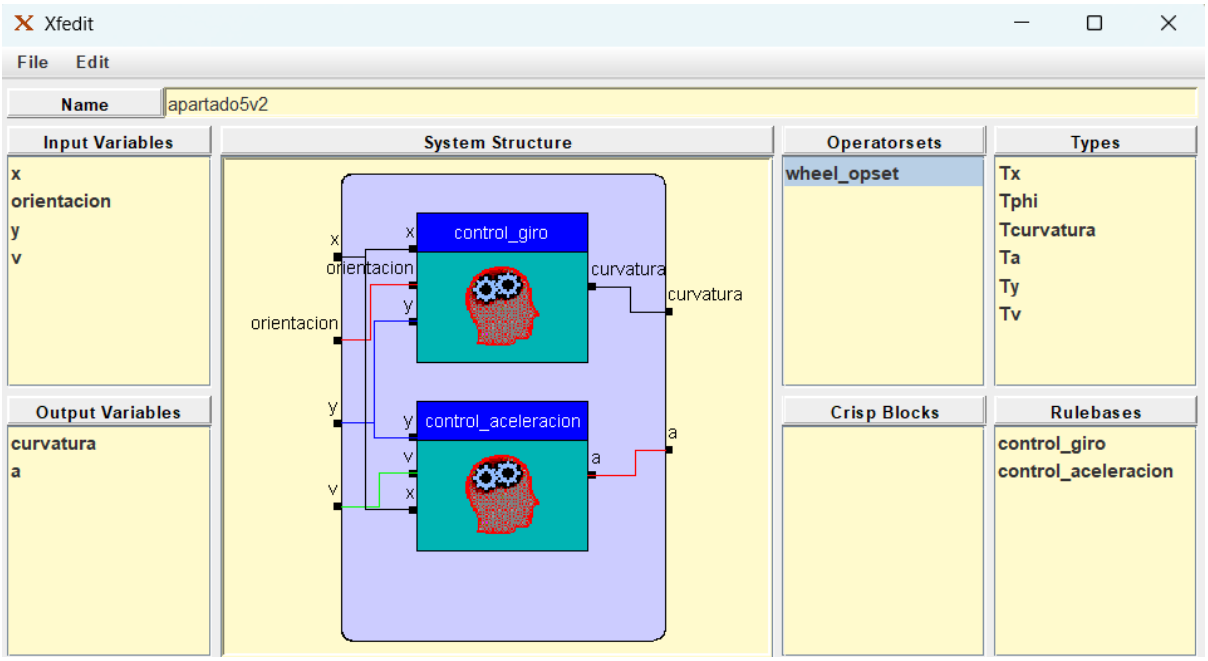
x	0.357143
orientacion	3.3785
y	29.919
v	1.0126E-4
Fuzzy system output	
curvatura	-0.0255475
a	-1.9756E-5

Como vemos funciona bastante bien, pero nos encontramos con el siguiente problema ¿Si empiezo cerca de la y=30 y estoy mal posicionado en y llego a mi objetivo?



Nuestro controlador no consigue llegar a nuestro objetivo porque según nuestra base de reglas si estamos cerca de y=30 frenamos sin conseguir llegar a nuestro objetivo, esto nos lleva al siguiente intento.

Controlador apartado5v2:



El objetivo es controlar la trayectoria del vehículo utilizando dos bases de reglas:

- Control de giro (control_giro):

  Entradas: y, x, y orientación.

  Salida: Curvatura.

- Control de aceleración (control_aceleración):

  Entradas: y, v (velocidad) y x.

  Salida: Aceleración (a).

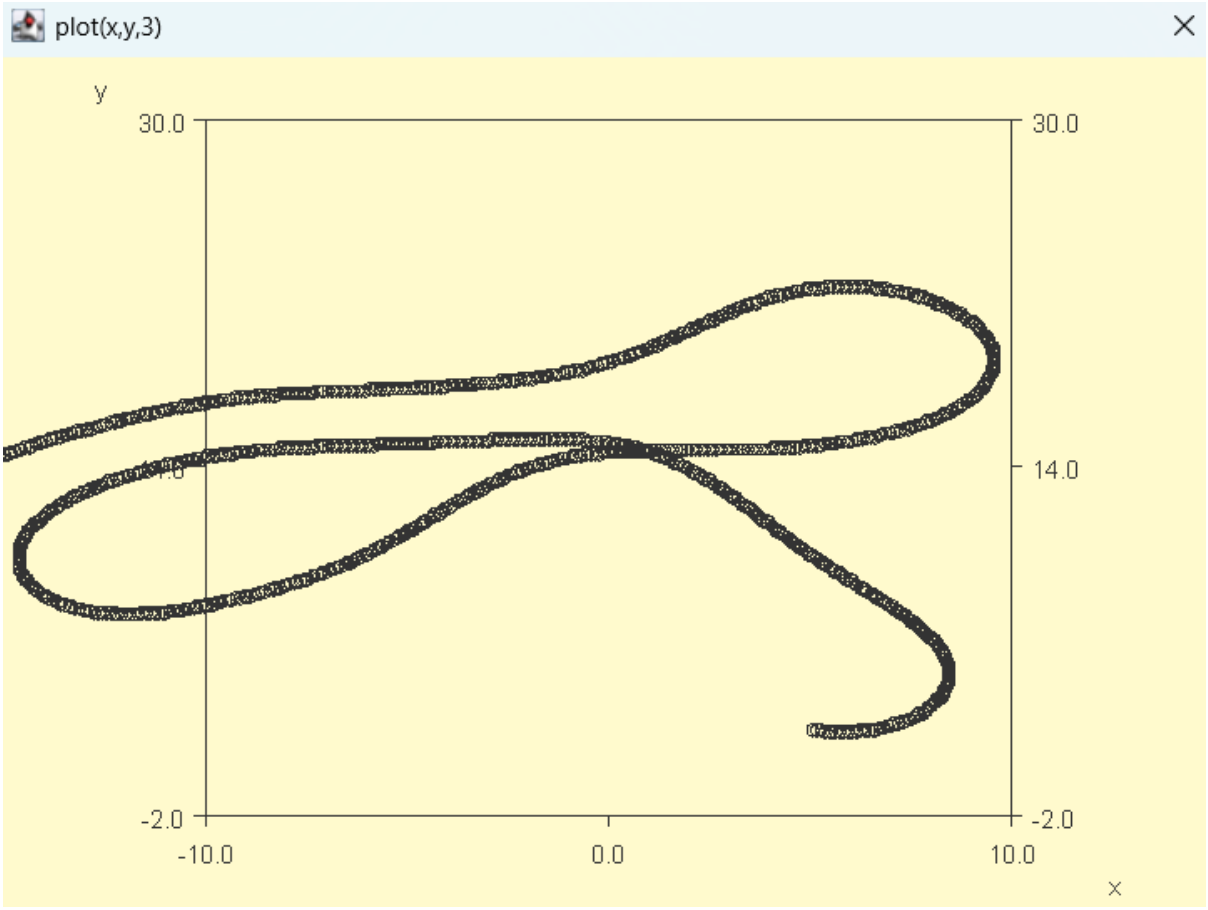
El propósito es que, mediante la variable y, el vehículo ajuste su trayectoria de forma que, si no está correctamente alineado con el objetivo, busque regresar a la zona más lejana del objetivo (en este caso, y=30). Desde allí, ajustará la posición de x y la orientación, y finalmente se dirigirá hacia el objetivo de forma precisa.

```
rulebase control_giro (Tx x, Tphi orientacion, Ty y : Tcurvatura curvatura) using
wheel_opset {

if(y  ==  Lejos  &  x  ==  LB  &  orientacion  ==  LB)  ->  curvatura  =  NB;
if(y  ==  Lejos  &  x  ==  LB  &  orientacion  ==  LM)  ->  curvatura  =  PB;
if(y  ==  Lejos  &  x  ==  LB  &  orientacion  ==  ZE)  ->  curvatura  =  PM;
if(y  ==  Lejos  &  x  ==  LB  &  orientacion  ==  RM)  ->  curvatura  =  ZE;
if(y  ==  Lejos  &  x  ==  LB  &  orientacion  ==  RB)  ->  curvatura  =  NB;

if(y  ==  Lejos  &  x  ==  LS  &  orientacion  ==  LB)  ->  curvatura  =  NB;
if(y  ==  Lejos  &  x  ==  LS  &  orientacion  ==  LM)  ->  curvatura  =  PB;
if(y  ==  Lejos  &  x  ==  LS  &  orientacion  ==  ZE)  ->  curvatura  =  PM;
if(y  ==  Lejos  &  x  ==  LS  &  orientacion  ==  RM)  ->  curvatura  =  NM;
if(y  ==  Lejos  &  x  ==  LS  &  orientacion  ==  RB)  ->  curvatura  =  NB;
```

```
if(y == Lejos & x == ZE & orientacion == LB) -> curvatura = PB;
if(y == Lejos & x == ZE & orientacion == LM) -> curvatura = PM;
if(y == Lejos & x == ZE & orientacion == ZE) -> curvatura = ZE;
if(y == Lejos & x == ZE & orientacion == RM) -> curvatura = NM;
if(y == Lejos & x == ZE & orientacion == RB) -> curvatura = NB;

if(y == Lejos & x == RS & orientacion == LB) -> curvatura = PB;
if(y == Lejos & x == RS & orientacion == LM) -> curvatura = PM;
if(y == Lejos & x == RS & orientacion == ZE) -> curvatura = NM;
if(y == Lejos & x == RS & orientacion == RM) -> curvatura = NB;
if(y == Lejos & x == RS & orientacion == RB) -> curvatura = NB;

if(y == Lejos & x == RB & orientacion == LB) -> curvatura = NB;
if(y == Lejos & x == RB & orientacion == LM) -> curvatura = ZE;
if(y == Lejos & x == RB & orientacion == ZE) -> curvatura = NB;
if(y == Lejos & x == RB & orientacion == RM) -> curvatura = NB;
if(y == Lejos & x == RB & orientacion == RB) -> curvatura = NB;


if(y == Cerca & x == RS & orientacion == LB) -> curvatura = NM;
if(y == Cerca & x == RS & orientacion == LM) -> curvatura = NB;
if(y == Cerca & x == RS & orientacion == ZE) -> curvatura = PM;
if(y == Cerca & x == RS & orientacion == RM) -> curvatura = ZE;
if(y == Cerca & x == RS & orientacion == RB) -> curvatura = NM;

if(y == Cerca & x == RB & orientacion == LB) -> curvatura = ZE;
if(y == Cerca & x == RB & orientacion == LM) -> curvatura = NM;
if(y == Cerca & x == RB & orientacion == ZE) -> curvatura = NB;
if(y == Cerca & x == RB & orientacion == RM) -> curvatura = PM;
if(y == Cerca & x == RB & orientacion == RB) -> curvatura = ZE;



if(y == Cerca & x == LS & orientacion == LB) -> curvatura = PM;
if(y == Cerca & x == LS & orientacion == LM) -> curvatura = ZE;
if(y == Cerca & x == LS & orientacion == ZE) -> curvatura = NM;
if(y == Cerca & x == LS & orientacion == RM) -> curvatura = PB;
if(y == Cerca & x == LS & orientacion == RB) -> curvatura = PM;

if(y == Cerca & x == LB & orientacion == LB) -> curvatura = ZE;
if(y == Cerca & x == LB & orientacion == LM) -> curvatura = NB;
if(y == Cerca & x == LB & orientacion == ZE) -> curvatura = PB;
if(y == Cerca & x == LB & orientacion == RM) -> curvatura = PB;
if(y == Cerca & x == LB & orientacion == RB) -> curvatura = ZE;


if(y == Medio & x == RS & orientacion == LB) -> curvatura = NM;
```

```
if(y == Medio & x == RS & orientacion == LM) -> curvatura = NB;
if(y == Medio & x == RS & orientacion == ZE) -> curvatura = PM;
if(y == Medio & x == RS & orientacion == RM) -> curvatura = ZE;
if(y == Medio & x == RS & orientacion == RB) -> curvatura = NM;

if(y == Medio & x == RB & orientacion == LB) -> curvatura = ZE;
if(y == Medio & x == RB & orientacion == LM) -> curvatura = NM;
if(y == Medio & x == RB & orientacion == ZE) -> curvatura = NB;
if(y == Medio & x == RB & orientacion == RM) -> curvatura = NB;
if(y == Medio & x == RB & orientacion == RB) -> curvatura = ZE;



if(y == Medio & x == LS & orientacion == LB) -> curvatura = PM;
if(y == Medio & x == LS & orientacion == LM) -> curvatura = NM;
if(y == Medio & x == LS & orientacion == ZE) -> curvatura = NM;
if(y == Medio & x == LS & orientacion == RM) -> curvatura = PB;
if(y == Medio & x == LS & orientacion == RB) -> curvatura = PM;

if(y == Medio & x == LB & orientacion == LB) -> curvatura = ZE;
if(y == Medio & x == LB & orientacion == LM) -> curvatura = NB;
if(y == Medio & x == LB & orientacion == ZE) -> curvatura = PB;
if(y == Medio & x == LB & orientacion == RM) -> curvatura = PM;
if(y == Medio & x == LB & orientacion == RB) -> curvatura = ZE;

if(y == Medio & x == ZE & orientacion == LB) -> curvatura = PM;
if(y == Medio & x == ZE & orientacion == LM) -> curvatura = NB;
if(y == Medio & x == ZE & orientacion == ZE) -> curvatura = ZE;
if(y == Medio & x == ZE & orientacion == RM) -> curvatura = PB;
if(y == Medio & x == ZE & orientacion == RB) -> curvatura = NM;



// "si x es Cero, alinea el coche (phi=0) para la parada final"
if(y == Cerca & x == ZE & orientacion == LB) -> curvatura = PM;
if(y == Cerca & x == ZE & orientacion == LM) -> curvatura = ZE;
if(y == Cerca & x == ZE & orientacion == ZE) -> curvatura = ZE;
if(y == Cerca & x == ZE & orientacion == RM) -> curvatura = ZE;
if(y == Cerca & x == ZE & orientacion == RB) -> curvatura = NM;


}

rulebase control_aceleracion (Ty y, Tv v, Tx x : Ta a) using wheel_opset {

if(y == Lejos & v == C) -> a = P;
if(y == Lejos & v == L) -> a = ZE;
if(y == Lejos & v == R) -> a = N;
```

```
if(y == Cerca & (x == LB | x == LS | x == RS | x == RB) & v == C) -> a = P;
if(y == Cerca & (x == LB | x == LS | x == RS | x == RB) & v == L) -> a = ZE;
if(y == Cerca & (x == LB | x == LS | x == RS | x == RB) & v == R) -> a = N;

if(y == Cerca & x == ZE & v == C) -> a = ZE;
if(y == Cerca & x == ZE & v == L) -> a = N;
if(y == Cerca & x == ZE & v == R) -> a = NB;

}
```



Después de varios intentos fallidos, llegamos a la conclusión de que manejar 84 reglas puede ser tedioso. A pesar de que la idea es buena, ajustar cada regla punto por punto resulta ser muy difícil, lo que nos lleva a optar por un controlador diferente.

Controlador apartado5v3:

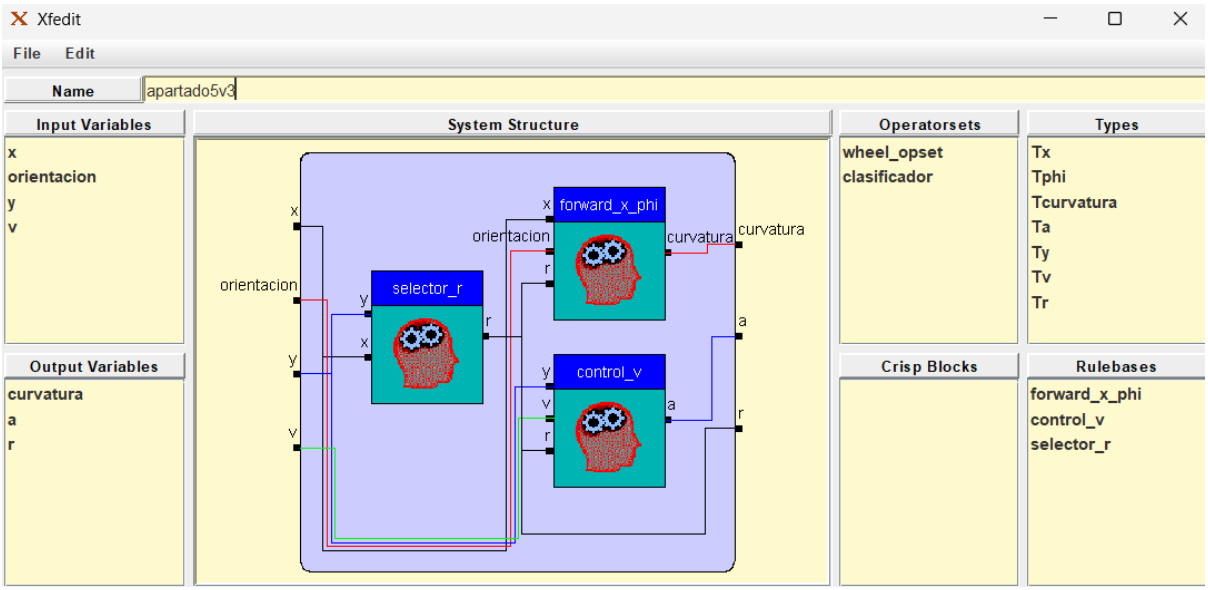
Tras analizar los controladores anteriores, hemos descubierto que manejar un gran número de reglas se vuelve confuso, y que conseguir frenar sin que el vehículo empiece a retroceder también es complicado. Estas cuestiones nos han llevado a considerar nuevas estrategias. Si analizamos el ejercicio, nos damos cuenta de que el coche no debería dar marcha atrás. Entonces, ¿y si tratamos el acelerador negativo como un freno, limitando la velocidad a un mínimo de 0 en el modelo del coche?

De esta forma, no infringimos la lógica real de un coche y solucionamos la dificultad de ajustar la velocidad, pudiendo ahora volver a poner Ty entre –2 y 30.

Para abordar el problema de tener tantas reglas, proponemos incorporar al inicio un clasificador que seleccione qué estrategia aplicar en función de las coordenadas x e y. Así, aplicaremos una regla estándar siempre que nos encontremos en un caso favorable, donde cualquier velocidad y orientación puedan cumplir el objetivo. Si no estamos en una situación favorable, retrocederemos hasta alcanzar una posición que lo sea.

Nuestro nuevo modelo será mimodelo2v2, al cual simplemente le hemos añadido esto en el for dentro del compute:

```
if (v < 0) {
    v = 0;
}
```

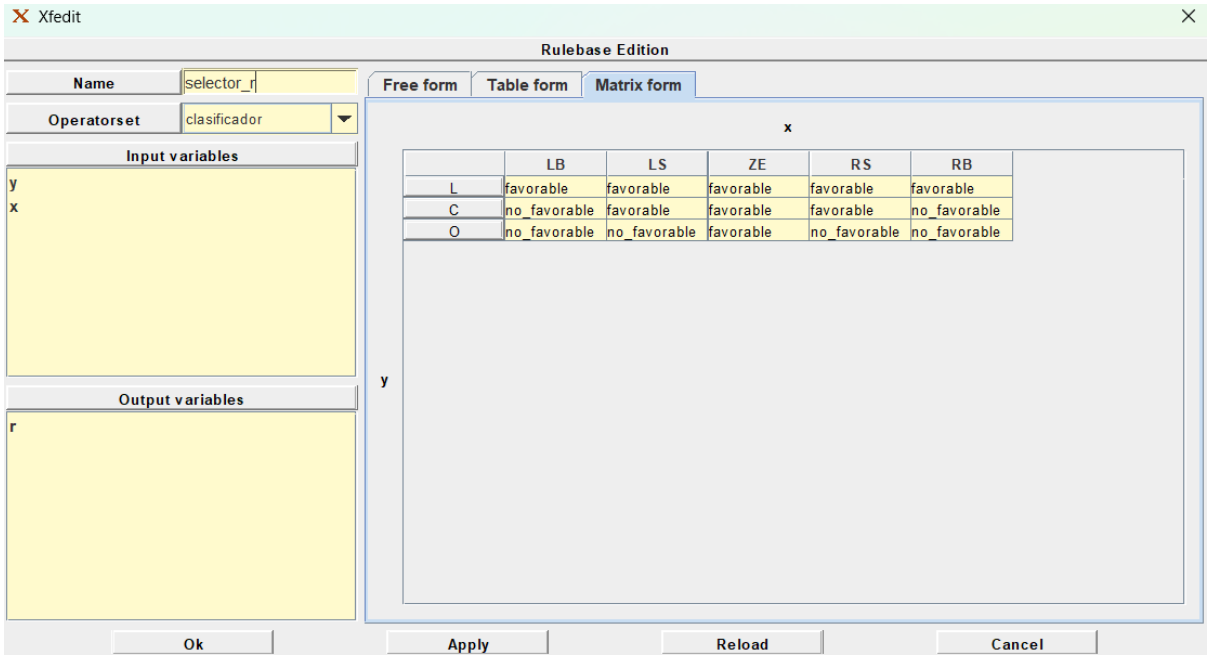


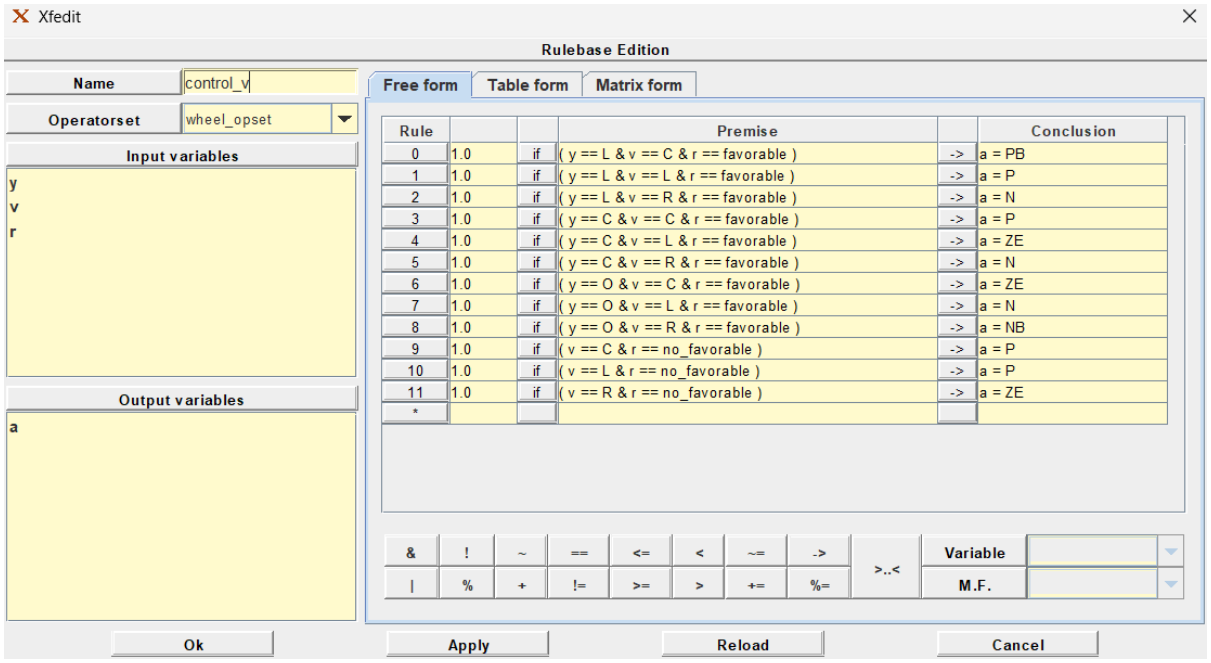
En este controlador hemos definido un nuevo operator set con defuzzificación MaxLabel, que se utiliza en selector_r.

La variable r es un singleton con dos posibles valores: 0 (favorable) o 1 (no favorable).

La salida de r se determina en la base de reglas selector_r, como hemos comentado anteriormente. A continuación, r se pasa al resto de la base de reglas para:

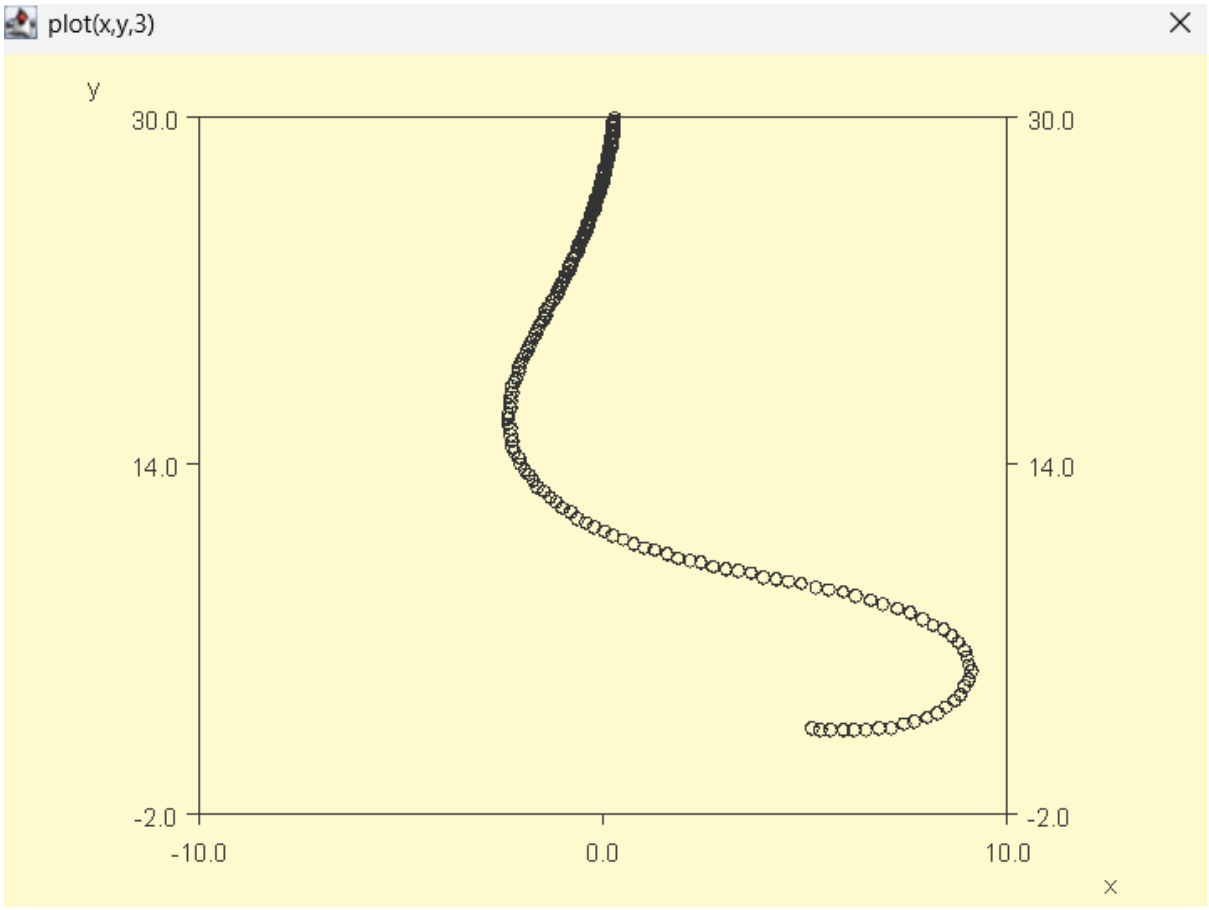
- Aplicar las reglas normales si estamos en un caso favorable.
- Aplicar un conjunto de reglas específicas destinadas a revertir la situación y volver a un estado favorable si estamos en un caso no favorable.


Xfedit
Rulebase Edition
Name
selector_r
Operatorset
clasificador
Input variables
y
x
Output variables
r
Free form
Table form
Matrix form
x
LB
LS
ZE
RS
RB
L
favorable
favorable
favorable
favorable
favorable
C
no_favorable
favorable
favorable
favorable
no_favorable
O
no_favorable
no_favorable
favorable
no_favorable
no_favorable
y
Ok
Apply
Reload
Cancel


Xfedit
Rulebase Edition
Name
control_v
Operatorset
wheel_opset
Input variables
y
v
r
Output variables
a
Free form
Table form
Matrix form
Rule
Premise
Conclusion
0 1.0 if (y == L & v == C & r == favorable) -> a = PB
1 1.0 if (y == L & v == L & r == favorable) -> a = P
2 1.0 if (y == L & v == R & r == favorable) -> a = N
3 1.0 if (y == C & v == C & r == favorable) -> a = P
4 1.0 if (y == C & v == L & r == favorable) -> a = ZE
5 1.0 if (y == C & v == R & r == favorable) -> a = N
6 1.0 if (y == O & v == C & r == favorable) -> a = ZE
7 1.0 if (y == O & v == L & r == favorable) -> a = N
8 1.0 if (y == O & v == R & r == favorable) -> a = NB
9 1.0 if (v == C & r == no_favorable) -> a = P
10 1.0 if (v == L & r == no_favorable) -> a = P
11 1.0 if (v == R & r == no_favorable) -> a = ZE
*
& ! ~ == <= < ~= ->
| % + != >= > += %=
>..<
Variable
M.F.
Ok
Apply
Reload
Cancel

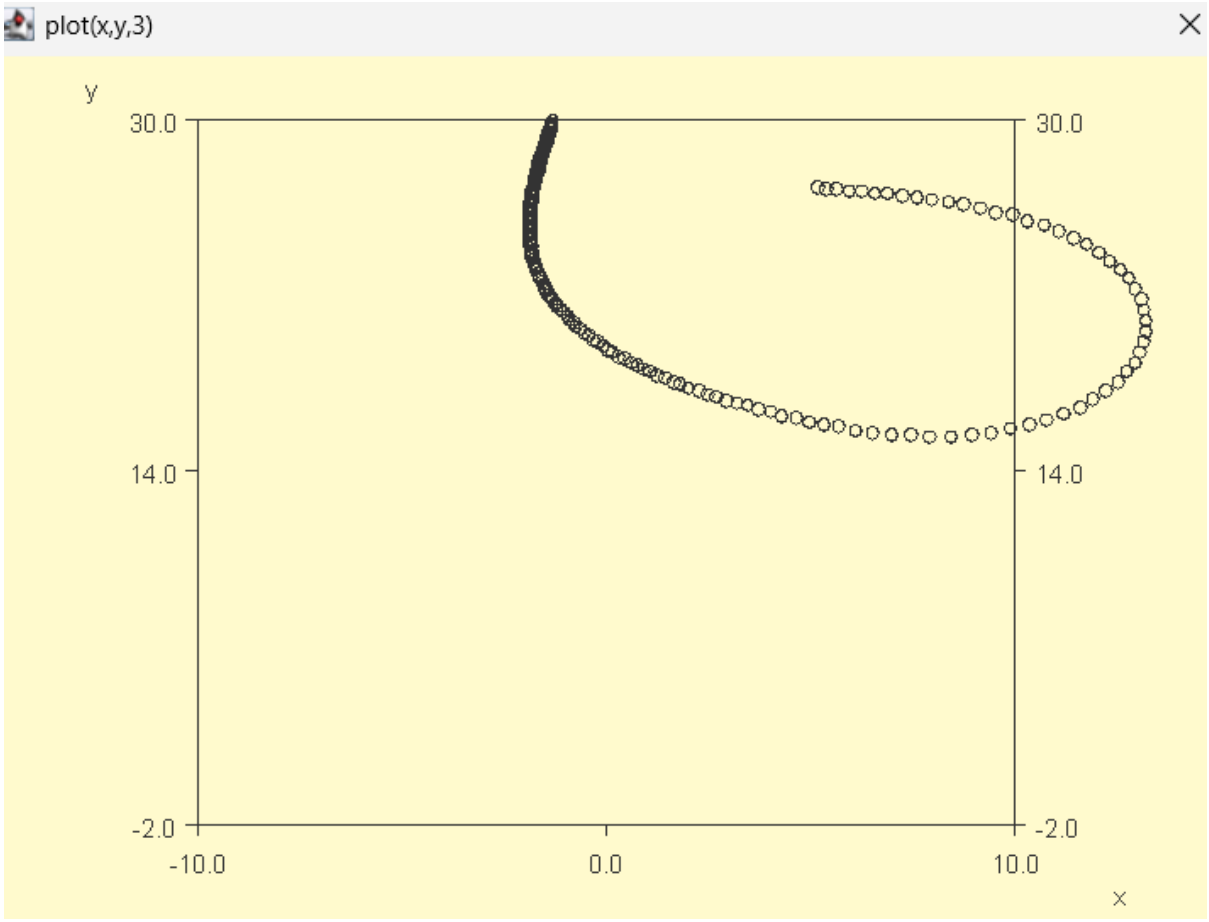
```
rulebase forward_x_phi (Tx x, Tphi orientacion, Tr r : Tcurvatura curvatura) using
wheel_opset {
  if(x == LB & orientacion == LB & r == favorable) -> curvatura = PB;
  if(x == LB & orientacion == LM & r == favorable) -> curvatura = PB;
  if(x == LB & orientacion == ZE & r == favorable) -> curvatura = PB;
  if(x == LB & orientacion == RM & r == favorable) -> curvatura = ZE;
  if(x == LB & orientacion == RB & r == favorable) -> curvatura = NM;
  if(x == LS & orientacion == LB & r == favorable) -> curvatura = PB;
  if(x == LS & orientacion == LM & r == favorable) -> curvatura = PB;
  if(x == LS & orientacion == ZE & r == favorable) -> curvatura = PM;
  if(x == LS & orientacion == RM & r == favorable) -> curvatura = NM;
  if(x == LS & orientacion == RB & r == favorable) -> curvatura = NB;
  if(x == ZE & orientacion == LB & r == favorable) -> curvatura = PB;
  if(x == ZE & orientacion == LM & r == favorable) -> curvatura = PM;
  if(x == ZE & orientacion == ZE & r == favorable) -> curvatura = ZE;
  if(x == ZE & orientacion == RM & r == favorable) -> curvatura = NM;
  if(x == ZE & orientacion == RB & r == favorable) -> curvatura = NB;
  if(x == RS & orientacion == LB & r == favorable) -> curvatura = PM;
  if(x == RS & orientacion == LM & r == favorable) -> curvatura = ZE;
  if(x == RS & orientacion == ZE & r == favorable) -> curvatura = NM;
  if(x == RS & orientacion == RM & r == favorable) -> curvatura = NB;
  if(x == RS & orientacion == RB & r == favorable) -> curvatura = NB;
  if(x == RB & orientacion == LB & r == favorable) -> curvatura = PM;
  if(x == RB & orientacion == LM & r == favorable) -> curvatura = ZE;
  if(x == RB & orientacion == ZE & r == favorable) -> curvatura = NB;
  if(x == RB & orientacion == RM & r == favorable) -> curvatura = NB;
  if(x == RB & orientacion == RB & r == favorable) -> curvatura = NB;
  if(x == LB & orientacion == LB & r == no_favorable) -> curvatura = NM;
  if(x == LB & orientacion == LM & r == no_favorable) -> curvatura = NB;
  if(x == LB & orientacion == ZE & r == no_favorable) -> curvatura = PB;
  if(x == LB & orientacion == RM & r == no_favorable) -> curvatura = PM;
  if(x == LB & orientacion == RB & r == no_favorable) -> curvatura = NM;
  if(x == LS & orientacion == LB & r == no_favorable) -> curvatura = PM;
  if(x == LS & orientacion == LM & r == no_favorable) -> curvatura = NM;
  if(x == LS & orientacion == ZE & r == no_favorable) -> curvatura = PB;
  if(x == LS & orientacion == RM & r == no_favorable) -> curvatura = PB;
  if(x == LS & orientacion == RB & r == no_favorable) -> curvatura = PM;
  if(x == RS & orientacion == LB & r == no_favorable) -> curvatura = NM;
  if(x == RS & orientacion == LM & r == no_favorable) -> curvatura = NM;
  if(x == RS & orientacion == ZE & r == no_favorable) -> curvatura = NB;
  if(x == RS & orientacion == RM & r == no_favorable) -> curvatura = PB;
  if(x == RS & orientacion == RB & r == no_favorable) -> curvatura = NM;
  if(x == RB & orientacion == LB & r == no_favorable) -> curvatura = PM;
  if(x == RB & orientacion == LM & r == no_favorable) -> curvatura = PM;
  if(x == RB & orientacion == ZE & r == no_favorable) -> curvatura = NB;
  if(x == RB & orientacion == RM & r == no_favorable) -> curvatura = PB;
  if(x == RB & orientacion == RB & r == no_favorable) -> curvatura = PM;
}
```

A pesar de parecer muchas normas separar la lógica con el clasificador hace que sea mucho más ordenado y entendible.

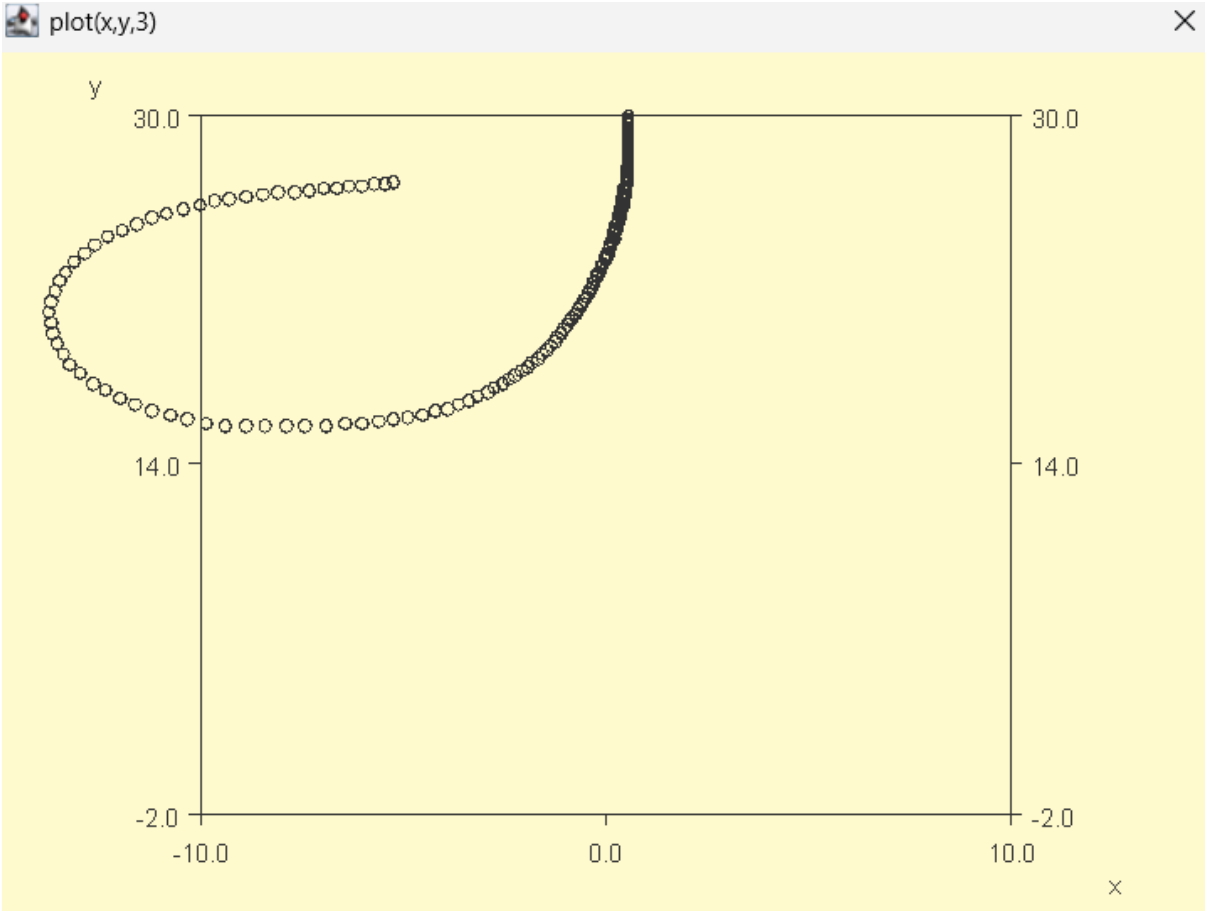


x	0.325265
orientacion	3.7012
y	29.999
v	1.1144E-4
Fuzzy system output	
curvatura	-0.0253482
a	-2.1625E-5
r	0.0

Como vemos desde un caso favorable, como ya habíamos conseguido en controladores anteriores, si funciona, ya que nos alcanza resultados muy próximos a nuestro objetivo.



x	-1.2564
orientacion	10.648
y	29.999
v	1.0205E-4
Fuzzy system output	
curvatura	0.00584237
a	-1.9412E-5
r	0.0



x	0.610113
orientacion	-0.881405
y	29.999
v	7.8162E-5
Fuzzy system output	
curvatura	-0.0214665
a	-1.5206E-5
r	0.0

Se ha logrado el objetivo, como vemos en estas imágenes cuando empezamos en casos no favorables el controlador aplica nuestra lógica y busca volver a un caso favorable para poder conseguir el objetivo. Como vemos los resultados aún tienen algo de error, pero vemos que se aproxima mucho, lo que quiere decir que nuestra estrategia es buena y con un poco de entrenamiento lograra bajar ese error.

Cabe mencionar que estos controladores que se muestran no han sido los únicos construidos para lograr este apartado, se han hecho muchos más intentos para lograr el objetivo, pero estos son los más destacados.

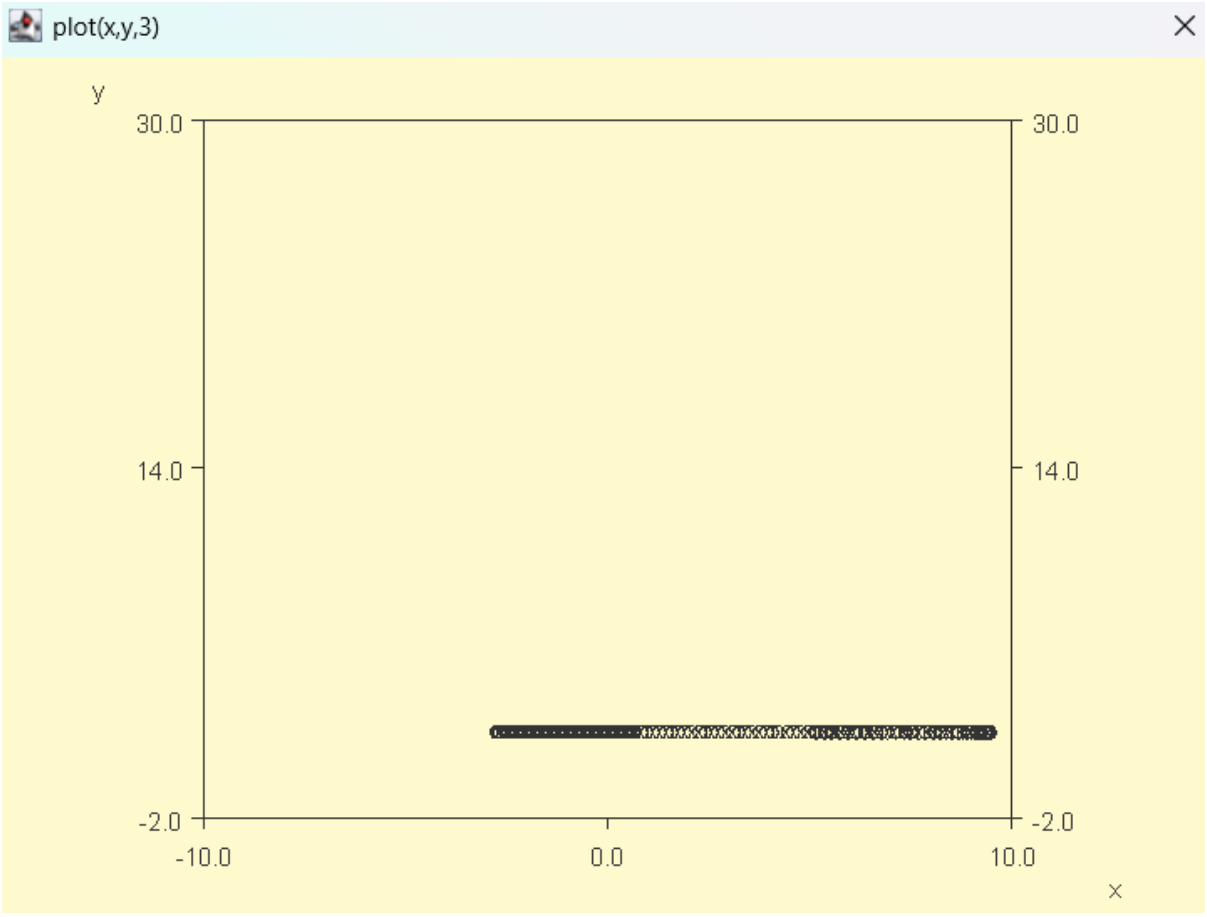
1.3 Modelo finalmente usado:

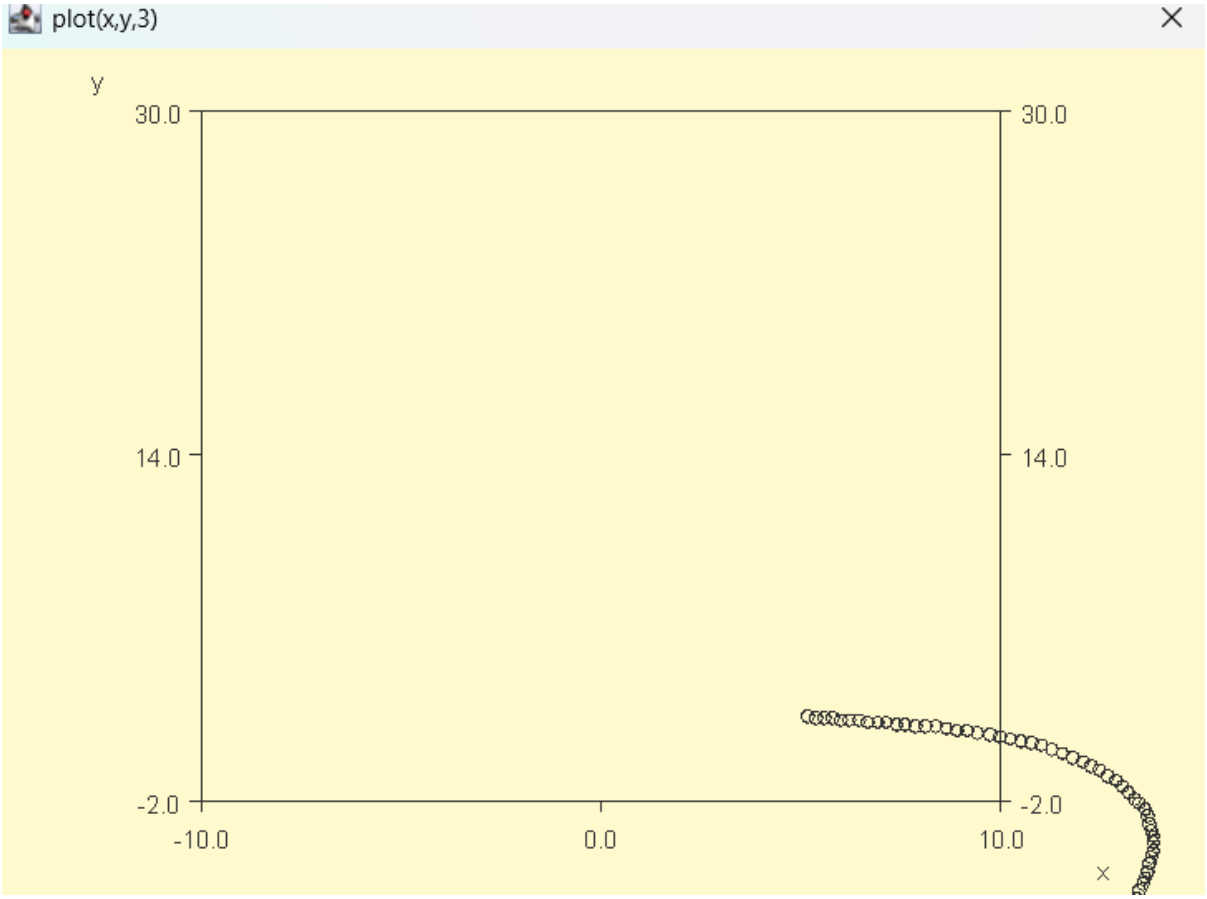
```
import xfuzzy.PlantModel;
public class Mimodelo2v2 implements PlantModel {
    private double x;
    private double y;
    private double phi;
    private double v = 0.0;
    private double oldgamma;
    private double gamma;
    private double olda;
    private double a;
    public Mimodelo2v2() {
    }
    public void init() {
        x = 0;
        phi = 0;
        y = 0;
        v = 0;
    }
    public void init(double val[]) {
        x = val[0];
        phi = val[1] * Math.PI / 180;
        y = val[2];
        v = val[3];
    }
    public double[] state() {
        double state[] = new double[4];
        state[0] = x;
        state[1] = phi * 180 / Math.PI;
        state[2] = y;
        state[3] = v;
        return state;
```

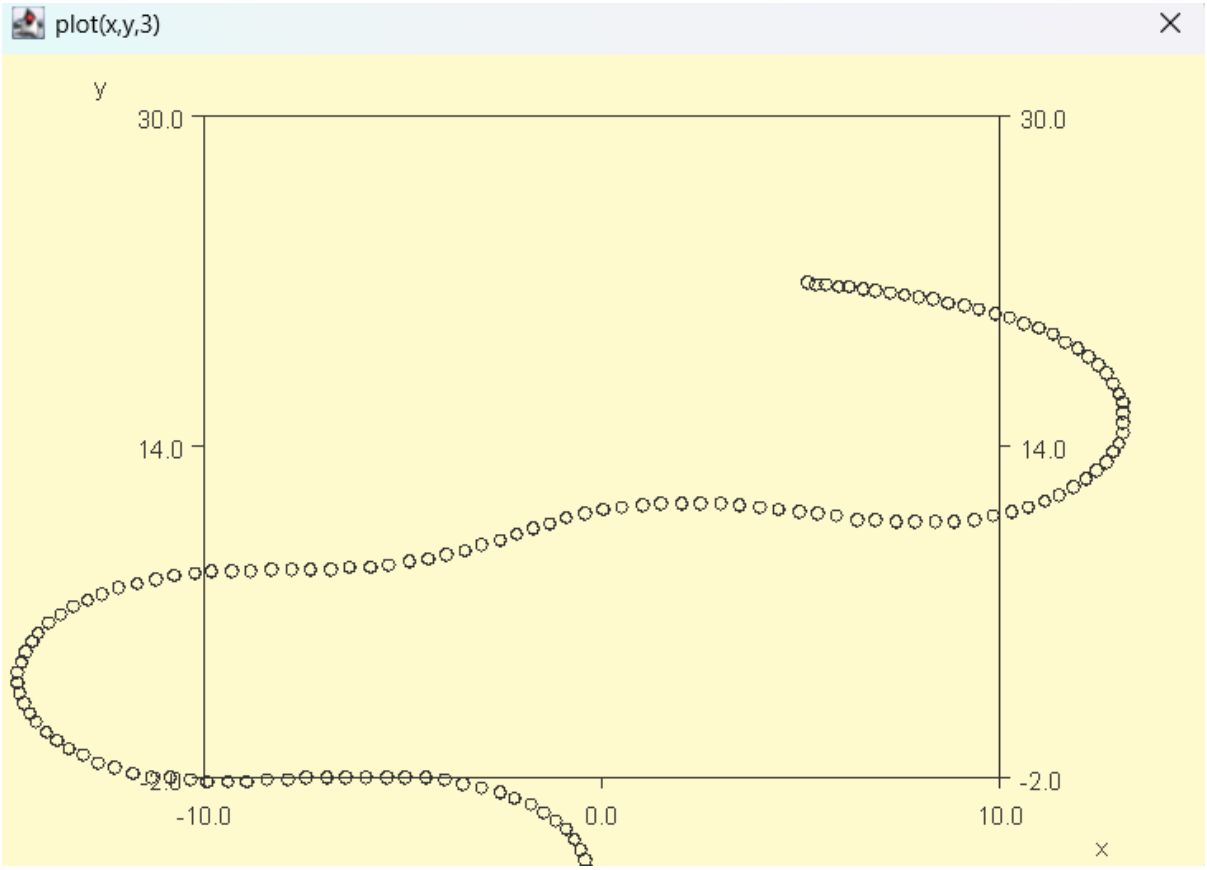
```
    }
    public double[] compute(double val[]) {
        double LAPSE = 0.1;
        double P_TAU = 0.5;
        double t = 0.0;
        double gref = 1.0 * val[0];
        olda = a;
        oldgamma = gamma;
        for (t = 0.0; t <= LAPSE; t += 0.001) {
            x += v * Math.sin(phi) * 0.001;
            y += v * Math.cos(phi) * 0.001;
            phi += v * gamma * 0.001;
            if (phi > Math.PI) phi -= 2 * Math.PI;
            if (phi < -Math.PI) phi += 2 * Math.PI;
            gamma = gref + (oldgamma - gref) * Math.exp(-t / P_TAU);
            if (gamma > 0.4) gamma = 0.4;
            if (gamma < -0.4) gamma = -0.4;
            a = olda + (val[1] - olda) * Math.exp(-t / P_TAU);
            if (a > 5) a = 5;
            if (a < -5) a = -5;
            v += a * 0.001;
            if (v < 0) {
                v = 0;
            }
        }
        return state();
    }
}
```

Apartado 6:

Para el apartado 6, se aplicará la misma técnica utilizada en el caso anterior, pero en esta ocasión el vehículo deberá estacionarse marcha atrás con una posición final en y = -2. Si el coche se encuentra cerca del objetivo y no puede completar la maniobra en esa posición, avanzará hasta una zona más favorable que le permita realizar la maniobra de reversa correctamente y completar el aparcamiento.







(Este último parece bueno, pero está llegando con velocidad positiva no nos sirve).

Después de muchos intentos, no estábamos obteniendo buenos resultados, lo que nos llevó a plantearnos la siguiente pregunta: ¿deberíamos, al igual que en el apartado anterior, cambiar de técnica?

El principal problema que no conseguimos resolver con esta técnica es que no logramos distinguir correctamente cuándo avanzar y cuándo retroceder, lo que provoca que las reglas se confundan.

Lo que nos lleva a la siguiente estrategia, mantener el controlador igual que el del apartado anterior que nos acercaba a y = 30 con x = 0 y orientación 0, una vez en este punto simplemente dar marcha atrás para llegar al aparcamiento.

```
type Tv [-5.0,5.0;256] {
  RB xfl.triangle(-7.5,-5.0,-2.5);
  RS xfl.triangle(-5.0,-2.5,0.0);
  C  xfl.triangle(-1.0,0.0,1.0);
  L  xfl.triangle(0.0,2.5,5.0);
  R  xfl.triangle(2.5,5.0,7.5);
}

type Tr [0.0,1.0;256] {
  favorable xfl.singleton(0.0);
  no_favorable xfl.singleton(0.5);
  back xfl.singleton(1);
}

rulebase selector_r (Ty y, Tx x, Tv v: Tr r) {
  if(y == L & x == LB & v == R) -> r = favorable;
  if(y == L & x == LB & v == L) -> r = favorable;
  if(y == L & x == LB & v == C) -> r = favorable;
  if(y == L & x == LS & v == R) -> r = favorable;
  if(y == L & x == LS & v == L) -> r = favorable;
  if(y == L & x == LS & v == C) -> r = favorable;
  if(y == L & x == ZE & v == R) -> r = favorable;
  if(y == L & x == ZE & v == L) -> r = favorable;
  if(y == L & x == ZE & v == C) -> r = back;
```

```
if(y == L & x == RB & v == R) -> r = favorable;
if(y == L & x == RB & v == L) -> r = favorable;
if(y == L & x == RB & v == C) -> r = favorable;
if(y == L & x == RS & v == R) -> r = favorable;
if(y == L & x == RS & v == L) -> r = favorable;
if(y == L & x == RS & v == C) -> r = favorable;
if(y == C & x == LB & v == R) -> r = no_favorable;
if(y == C & x == LB & v == L) -> r = no_favorable;
if(y == C & x == LB & v == C) -> r = no_favorable;
if(y == C & x == LS & v == R) -> r = favorable;
if(y == C & x == LS & v == L) -> r = favorable;
if(y == C & x == LS & v == C) -> r = favorable;
if(y == C & x == ZE & v == R) -> r = favorable;
if(y == C & x == ZE & v == L) -> r = favorable;
if(y == C & x == ZE & v == C) -> r = favorable;
if(y == C & x == RS & v == R) -> r = favorable;
if(y == C & x == RS & v == L) -> r = favorable;
if(y == C & x == RS & v == C) -> r = favorable;
if(y == C & x == RB & v == R) -> r = no_favorable;
if(y == C & x == RB & v == L) -> r = no_favorable;
if(y == C & x == RB & v == C) -> r = no_favorable;
if(y == O & x == LB & v == R) -> r = no_favorable;
if(y == O & x == LB & v == L) -> r = no_favorable;
if(y == O & x == LB & v == C) -> r = no_favorable;

if(y == O & x == LS & v == R) -> r = no_favorable;
if(y == O & x == LS & v == L) -> r = no_favorable;
if(y == O & x == LS & v == C) -> r = no_favorable;

if(y == O & x == ZE & v == R) -> r = favorable;
if(y == O & x == ZE & v == L) -> r = favorable;
if(y == O & x == ZE & v == C) -> r = back;
```

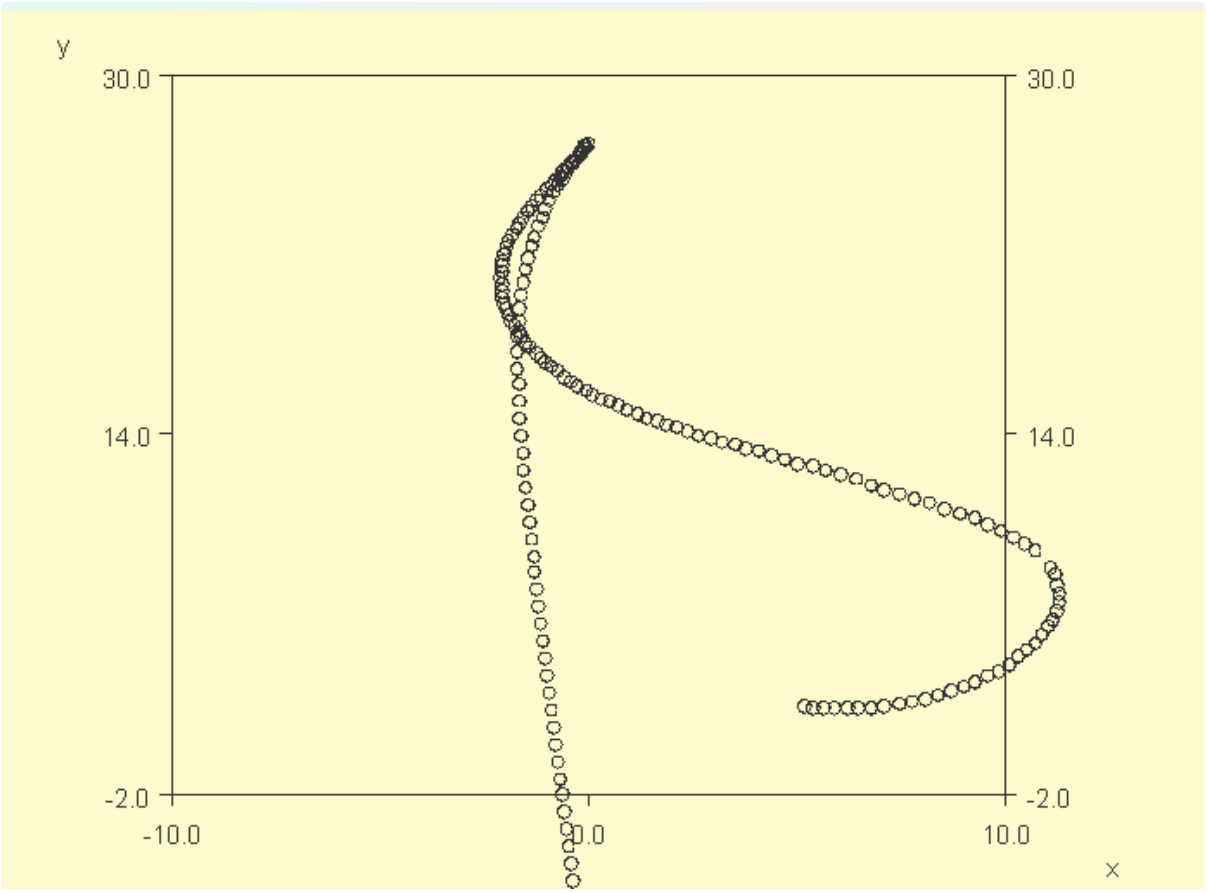
```
  if(y == O & x == RS & v == R) -> r = no_favorable;

  if(y == O & x == RS & v == L) -> r = no_favorable;

  if(y == O & x == RS & v == C) -> r = no_favorable;


  if(y == O & x == RB & v == R) -> r = no_favorable;

  if(y == O & x == RB & v == L) -> r = no_favorable;

  if(y == O & x == RB & v == C) -> r = no_favorable;


  if(v == RB) -> r = back;

  if(v == RS) -> r = back;
}
```

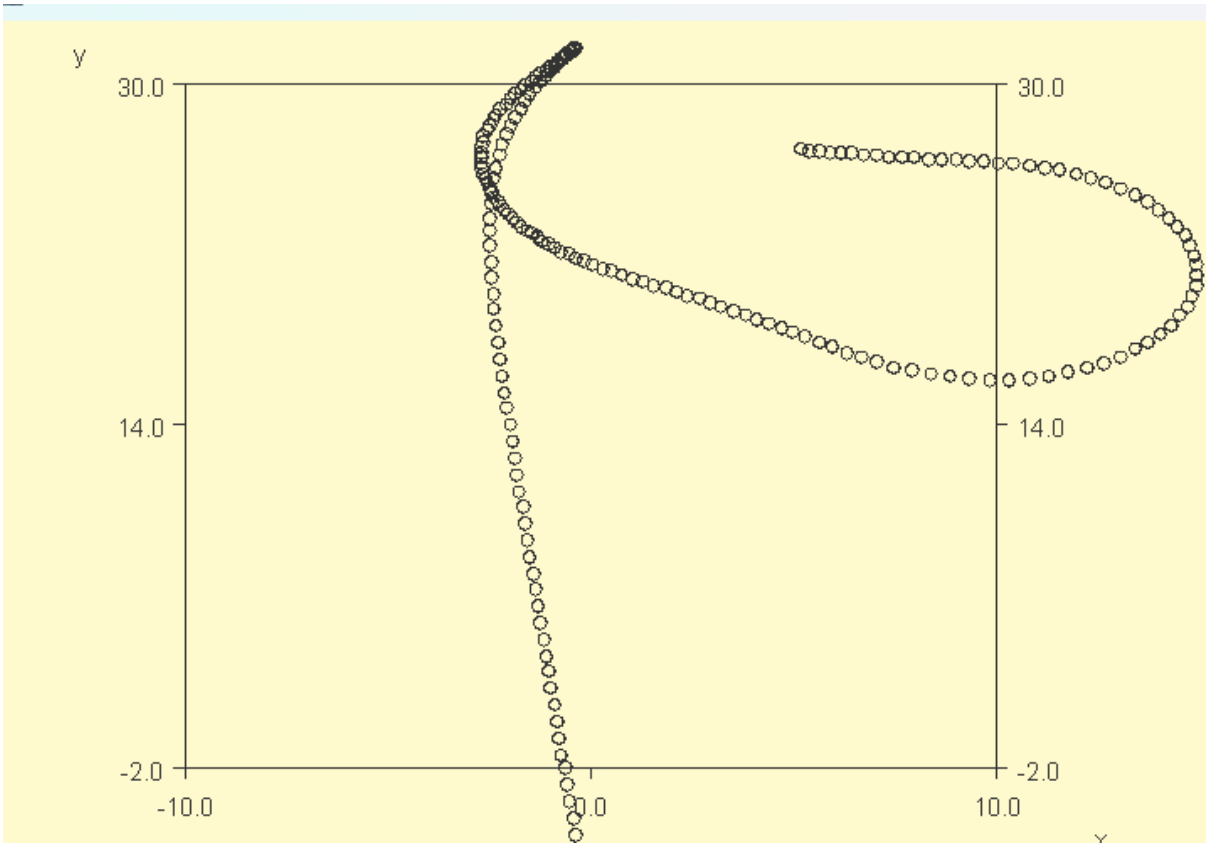
Hemos añadido back, que solo se activa si la velocidad es negativa o si estoy en y = 30, x =0, orientación = 0 y v = 0.

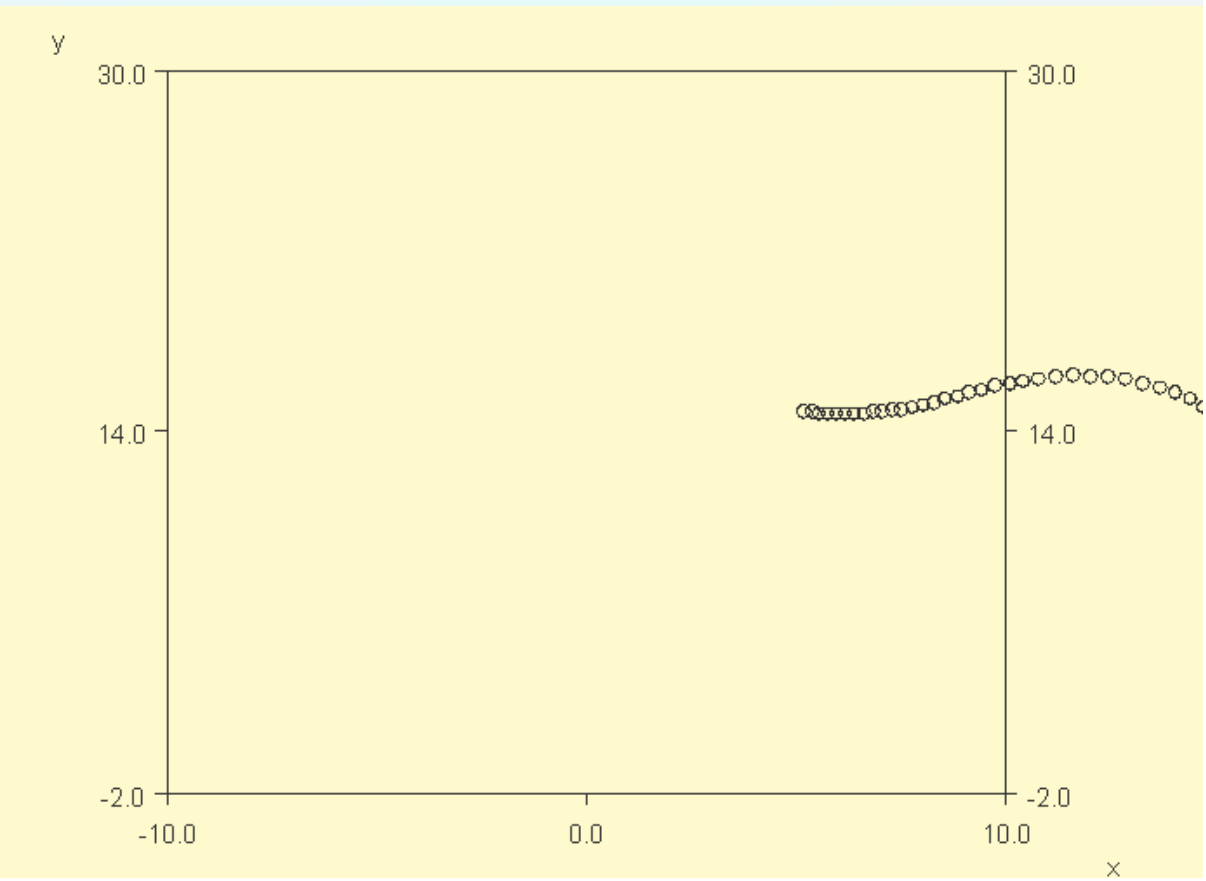
```
rulebase control_v (Ty y, Tv v, Tr r : Ta a) using wheel_opset { if(y == L & v == C & r == favorable) -> a = PB; if(y == L & v == L & r == favorable) -> a = P; if(y == L & v == R & r == favorable) -> a = N;

if(y == C & v == C & r == favorable) -> a = P;
 if(y == C & v == L & r == favorable) -> a = ZE;
 if(y == C & v == R & r == favorable) -> a = N;

if(y == O & v == C & r == favorable) -> a = ZE;
 if(y == O & v == L & r == favorable) -> a = N;
 if(y == O & v == R & r == favorable) -> a = NB;

if(v == C & r == no_favorable) -> a = P;
 if(v == L & r == no_favorable) -> a = P;
 if(v == R & r == no_favorable) -> a = ZE;

if(r == back) -> a = N;
}
```

Y para la curvatura todo igual, pero añadiendo estas reglas:

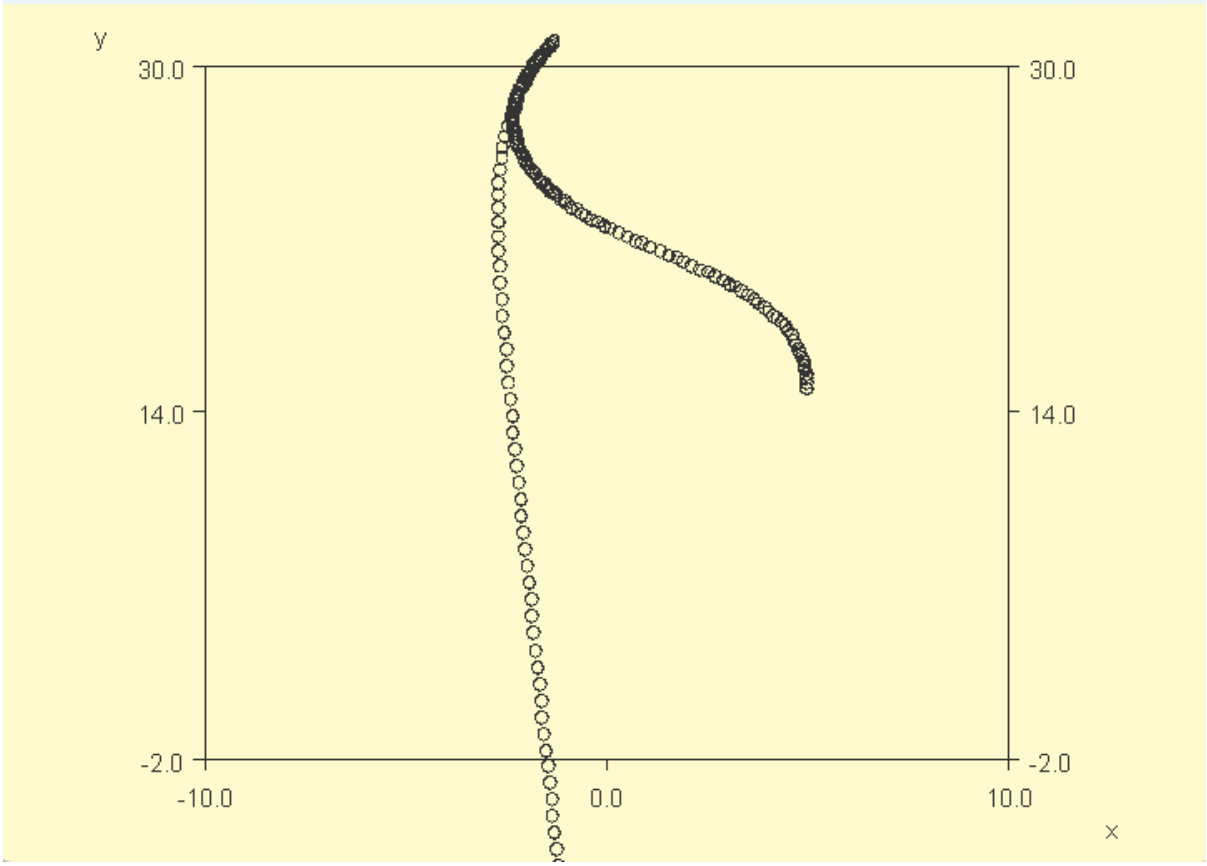
```
  if(r == back & orientacion == RM) -> curvatura = PM;

  if(r == back & orientacion == ZE) -> curvatura = ZE;

  if(r == back & orientacion == LM) -> curvatura = NM;
```


y
30.0
30.0
14.0
14.0
-2.0
-2.0
-10.0
0.0
10.0
x

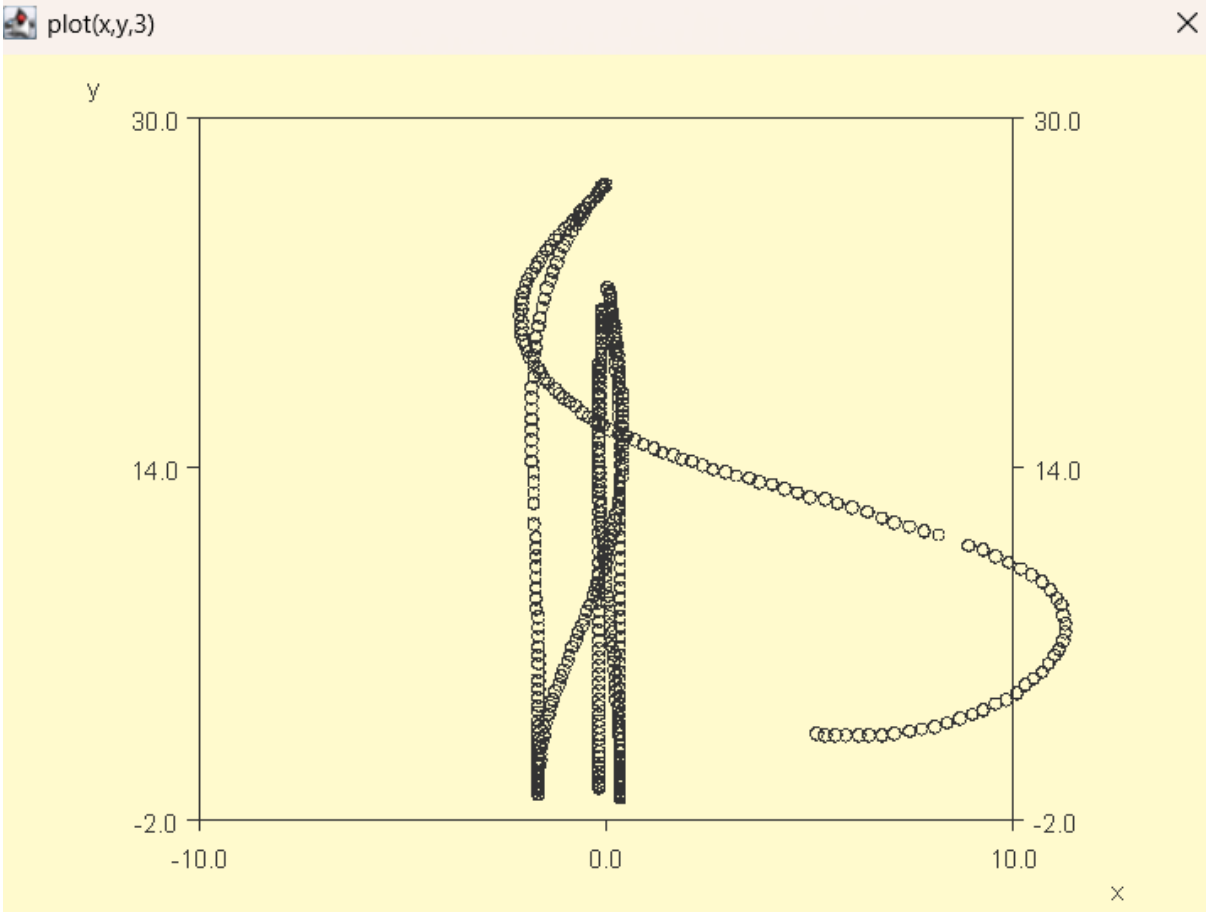

y
30.0
30.0
14.0
14.0
-2.0
-2.0
-10.0
0.0
10.0



Como podemos ver, ha funcionado razonablemente bien, pero aún quedan algunos aspectos por mejorar. En primer lugar, es necesario que el vehículo frene al llegar al aparcamiento. En segundo lugar, en la posición y = 15 no consigue seguir correctamente la trayectoria establecida.



En otra prueba nos damos cuenta de que, con una orientación inicial menos desfavorable (antes usábamos orientación = 100), el sistema funciona correctamente. Vamos a centrarnos primero en solucionar el problema de la velocidad.



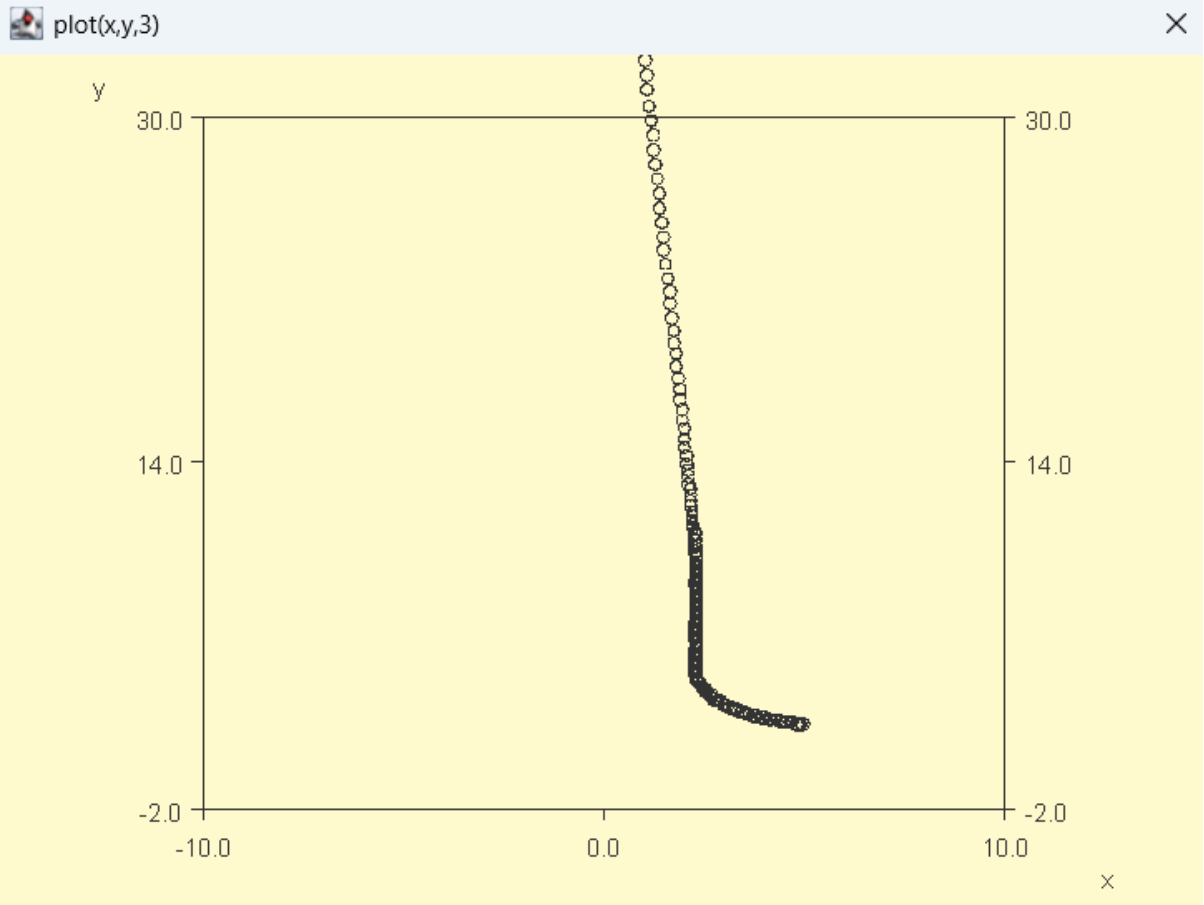
El intento de frenar el coche no está funcionando bien, esto nos lleva al problema que nos pasó en el apartado 5, es muy complejo frenar el coche si este freno que usamos nos permite acabar dirigirnos en sentido contrario, será necesario modificar el modelo para incorporar marchas, de manera que acelerar se traduzca en aplicar aceleración positiva según la marcha seleccionada, y frenar implique aplicar aceleración negativa.

```
type Tm [0.0,1.0;256] {
  adelante xfl.singleton(0.0);
  atras xfl.singleton(1.0);
}


rulebase control_marcha (Tr r : Tm m) using clasificador {
  if(r == favorable) -> m = adelante;
  if(r == no_favorable) -> m = adelante;
  if(r == back) -> m = atras;
}
```

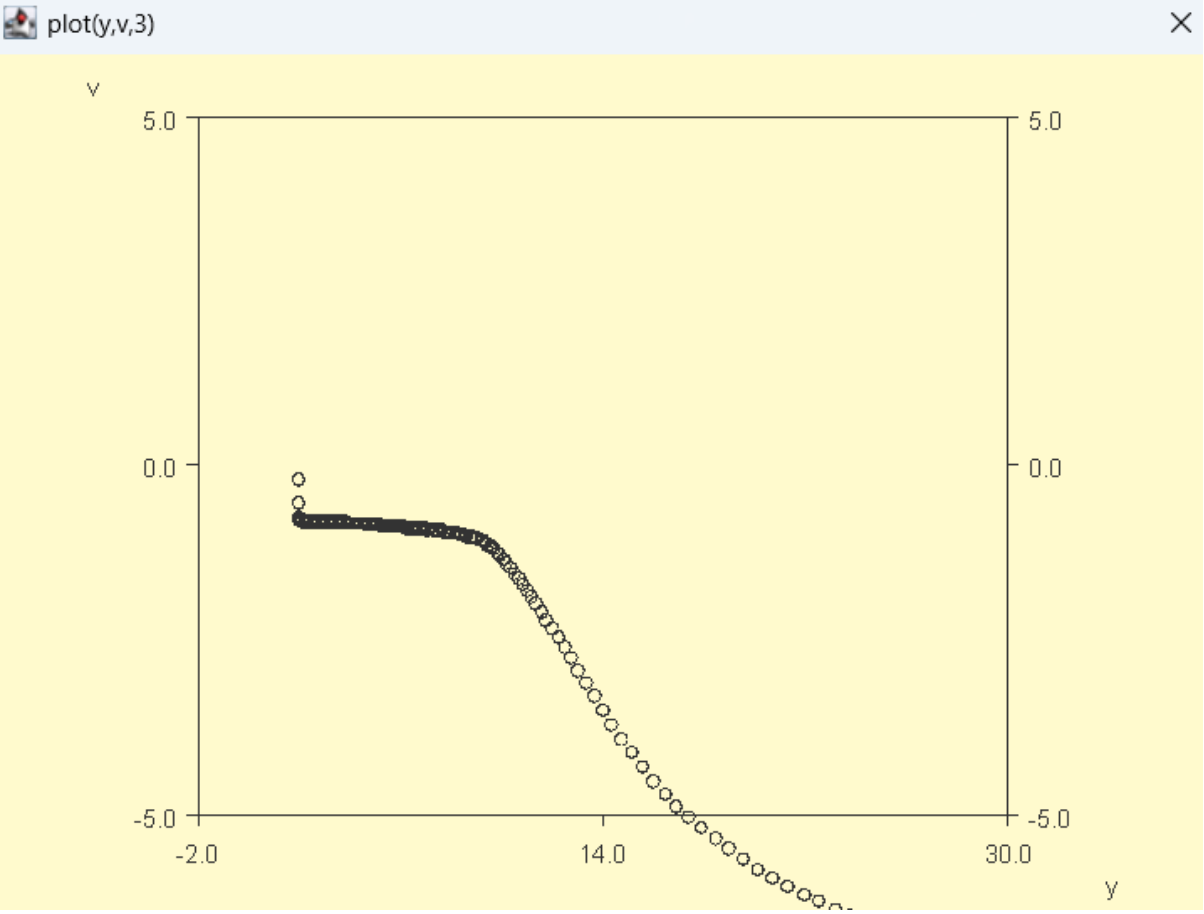
De esta manera seguimos la misma lógica, pero modificando el modelo para que tenga en cuenta esta lógica:

```
public double[] compute(double val[]) {
      double LAPSE = 0.1;
      double P_TAU = 0.5; // 1.8
      double t = 0.0;
      double gref = 1.0 * val[0];
      olda = a;
      oldgamma = gamma;
      int marcha = (int)val[2];
      for (t = 0.0; t <= LAPSE; t += 0.001) {
          x += v * Math.sin(phi) * 0.001;
          y += v * Math.cos(phi) * 0.001;
          phi += v * gamma * 0.001;
          if (phi > Math.PI) phi -= 2 * Math.PI;
          if (phi < -Math.PI) phi += 2 * Math.PI;
          gamma = gref + (oldgamma - gref) * Math.exp(-t / P_TAU);
          if (gamma > 0.4) gamma = 0.4;
          if (gamma < -0.4) gamma = -0.4;
          a = olda + (val[1] - olda) * Math.exp(-t / P_TAU);
          if (a > 5) a = 5;
          if (a < -5) a = -5;
          if(marcha == 0){
              a = a * -1;
          }
          v += a * 0.001;
          if(v>0 && marcha == 0){
              v = 0;
          }
          if(v<0 && marcha == 1){
              v = 0;
          }
      }
      return state();
```

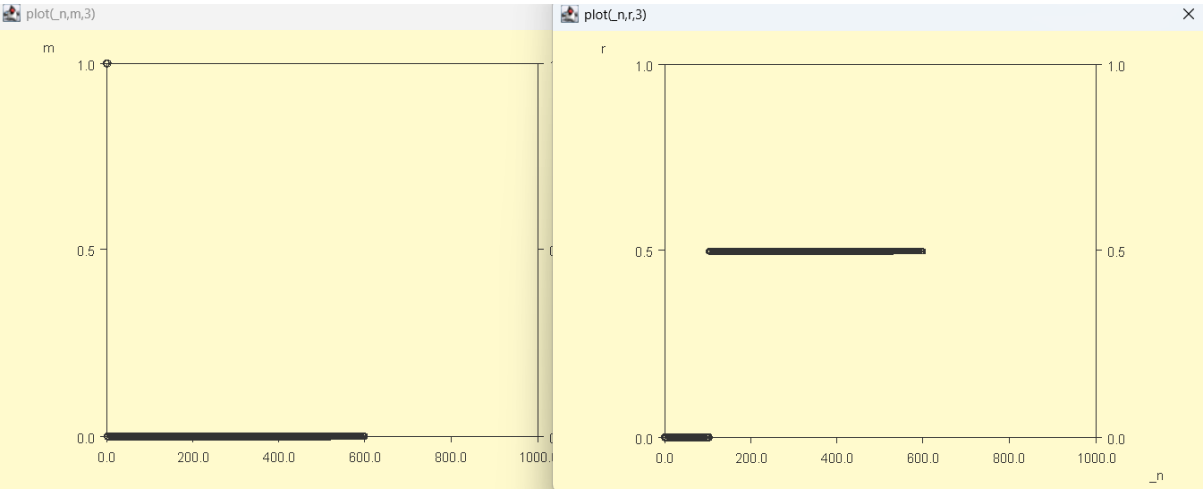
}



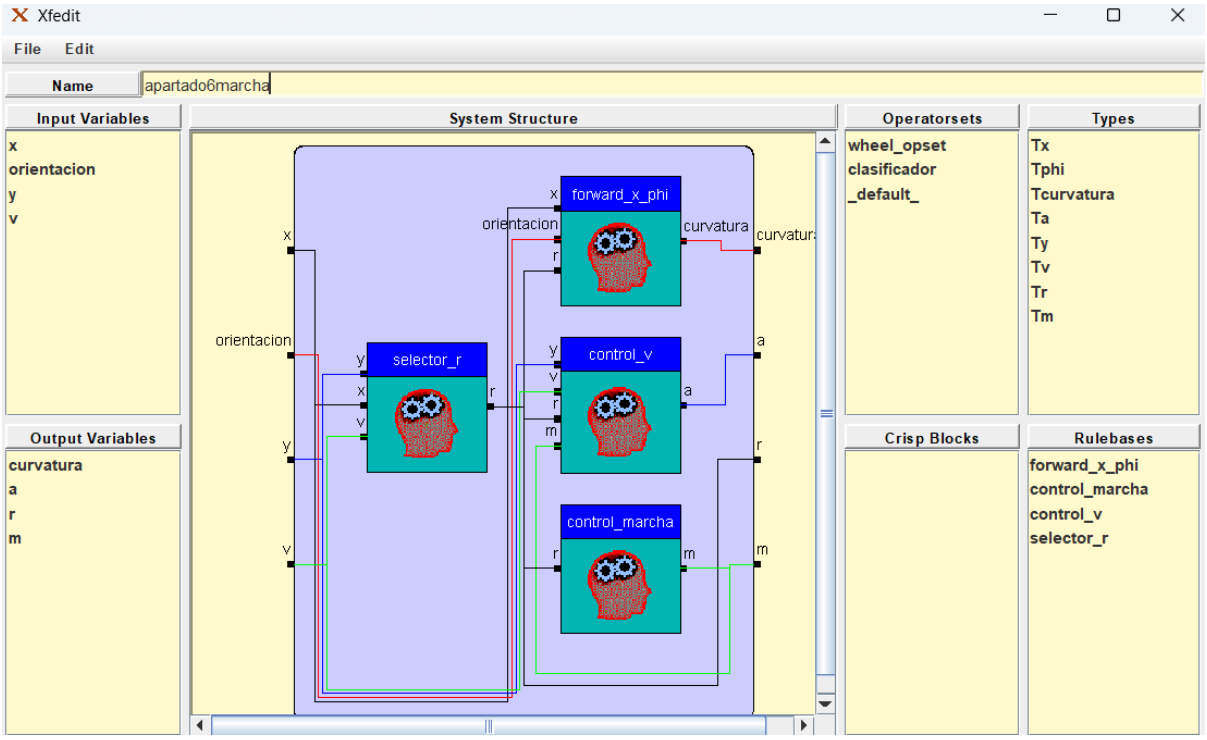
Sin embargo, como vemos sigue sin funcionar, si indagamos un poco y nos fijamos en como varía la velocidad:

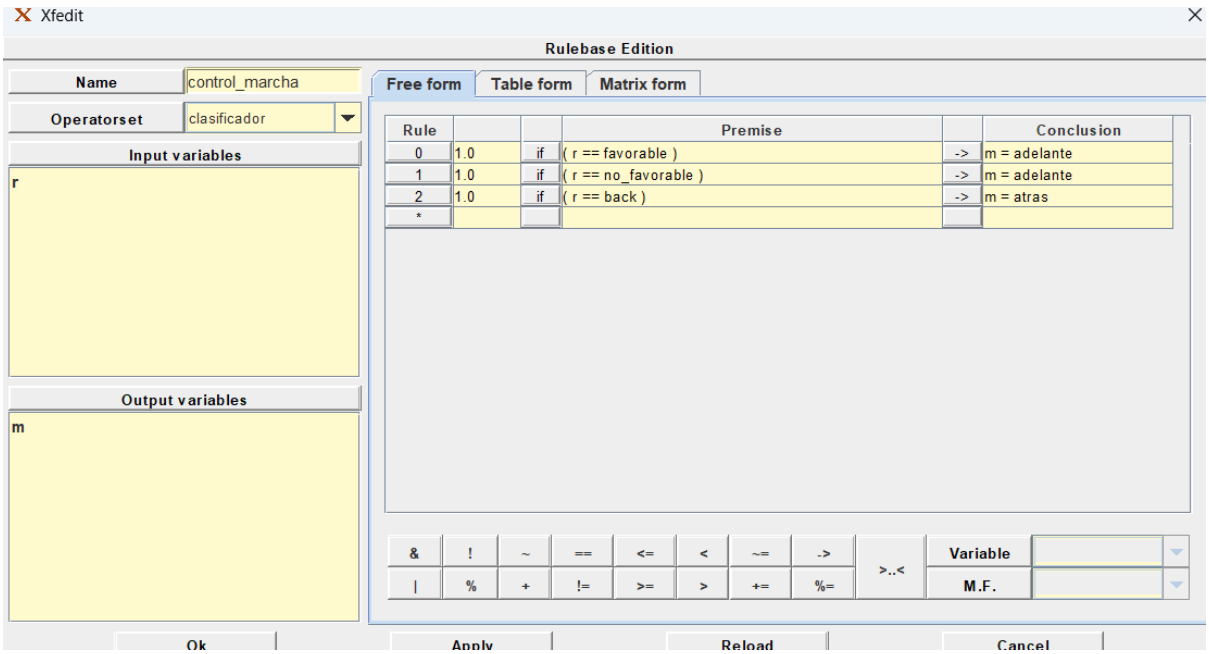


Vemos que la velocidad nos salta directamente a ser negativa cuando debería de ser positiva hasta y=30 y ya ahí cambiar a negativa.

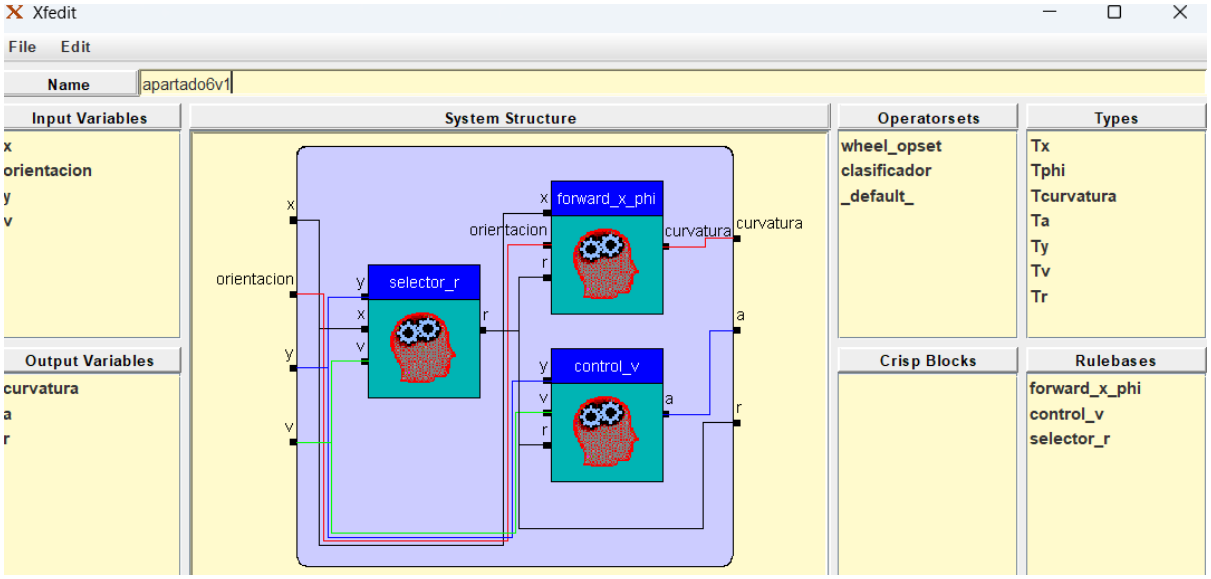


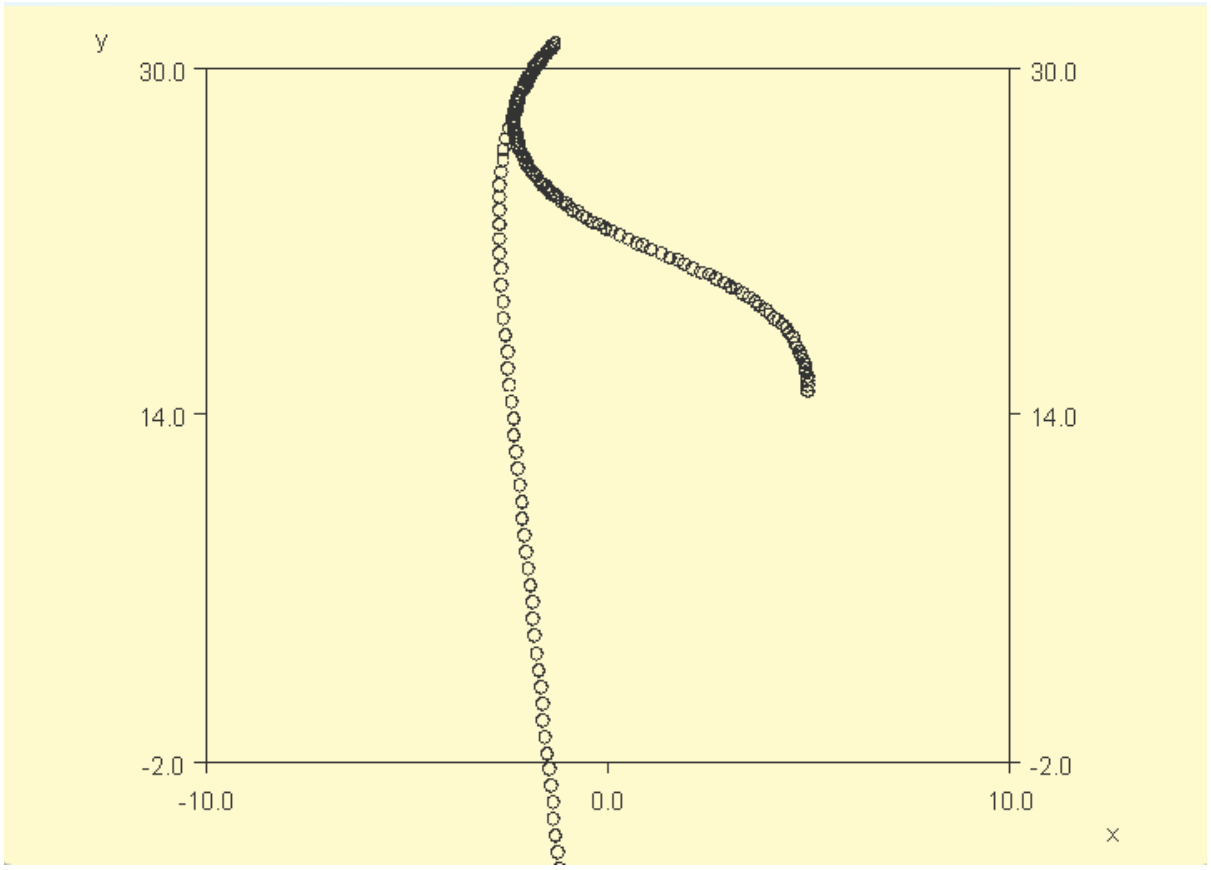
La incoherencia se debe a que, según la regla, cuando la relación (r) es favorable, la marcha debe ser siempre adelante. Sin embargo, al observar que la marcha atrás comienza a activarse a pesar de que r es favorable, se puede inferir que hay un fallo en la implementación de esa lógica o una condición no prevista que permite el cambio de marcha en circunstancias que no deberían ser posibles. Esto podría indicar que el sistema no está manejando correctamente todas las condiciones que determinan el comportamiento de la marcha.





Por lo que el controlador que más se ha acercado a nuestro objetivo ha sido apartado6v1.xfl, como ya vimos (este seguía usando Mimodelo2).





Apartado 7:

1.1 Mimodelo2:

```
import xfuzzy.PlantModel;


public class Mimodelo2 implements PlantModel {
```

```
private double x;
private double y;
private double phi;
private double v = 0.0;
private double oldgamma;
private double gamma;
private double olda;
private double a;

public Mimodelo2() {
}

public void init() {
    x = 0;
    phi = 0;
    y = 0;
    v = 0;
}

public void init(double val[]) {
    x = val[0];
    phi = val[1] * Math.PI / 180;
    y = val[2];
    v = val[3];
}

public double[] state() {
    double state[] = new double[4];
    state[0] = x;
    state[1] = phi * 180 / Math.PI;
    state[2] = y;
    state[3] = v;
```

```
    return state;
}

public double[] compute(double val[]) {
    double LAPSE = 0.1;
    double P_TAU = 0.5; // 1.8
    double t = 0.0;
    double gref = 1.0 * val[0];
    olda = a;
    oldgamma = gamma;

    for (t = 0.0; t <= LAPSE; t += 0.001) {
        x += v * Math.sin(phi) * 0.001;
        y += v * Math.cos(phi) * 0.001;
        phi += v * gamma * 0.001;

        if (phi > Math.PI) phi -= 2 * Math.PI;
        if (phi < -Math.PI) phi += 2 * Math.PI;

        gamma = gref + (oldgamma - gref) * Math.exp(-t / P_TAU);
        if (gamma > 0.4) gamma = 0.4;
        if (gamma < -0.4) gamma = -0.4;

        a = olda + (val[1] - olda) * Math.exp(-t / P_TAU);
        if (a > 5) a = 5;
        if (a < -5) a = -5;

        v += a * 0.001;

    }

    return state();
```

```
    }
}
```

Modelo utilizado para nuestro mejor controlador (apartado6v1.xfl).

1.2 Mimodelo3:

```
import xfuzzy.PlantModel;

public class Mimodelo3 implements PlantModel {
    private double x;
    private double y;
    private double phi;
    private double v = 0.0;
    private double oldgamma;
    private double gamma;
    private double olda;
    private double a;

    public Mimodelo3() {
    }

    public void init() {
        x = 0;
        phi = 0;
        y = 0;
        v = 0;
    }

    public void init(double val[]) {
        x = val[0];
        phi = val[1] * Math.PI / 180;
        y = val[2];
        v = val[3];
```

```
}

public double[] state() {
    double state[] = new double[4];
    state[0] = x;
    state[1] = phi * 180 / Math.PI;
    state[2] = y;
    state[3] = v;
    return state;
}

public double[] compute(double val[]) {
    double LAPSE = 0.1;
    double P_TAU = 0.5; // 1.8
    double t = 0.0;
    double gref = 1.0 * val[0];
    olda = a;
    oldgamma = gamma;
    int marcha = (int)val[2];

    for (t = 0.0; t <= LAPSE; t += 0.001) {
        x += v * Math.sin(phi) * 0.001;
        y += v * Math.cos(phi) * 0.001;
        phi += v * gamma * 0.001;
        if (phi > Math.PI) phi -= 2 * Math.PI;
        if (phi < -Math.PI) phi += 2 * Math.PI;

        gamma = gref + (oldgamma - gref) * Math.exp(-t / P_TAU);
        if (gamma > 0.4) gamma = 0.4;
        if (gamma < -0.4) gamma = -0.4;

        a = olda + (val[1] - olda) * Math.exp(-t / P_TAU);
```

```
            if (a > 5) a = 5;
            if (a < -5) a = -5;

            if(marcha == 0){
                a = a * -1;
            }
            v += a * 0.001;
            if(v>0 && marcha == 0){
                v = 0;
            }
            if(v<0 && marcha == 1){
                v = 0;
            }

        }

        return state();
    }
}
```

Modelo utilizado para el último intento apartado6marcha.xfl.

Ambos modelos han sido explicados ya en los apartados anteriores, al igual que las simulaciones.