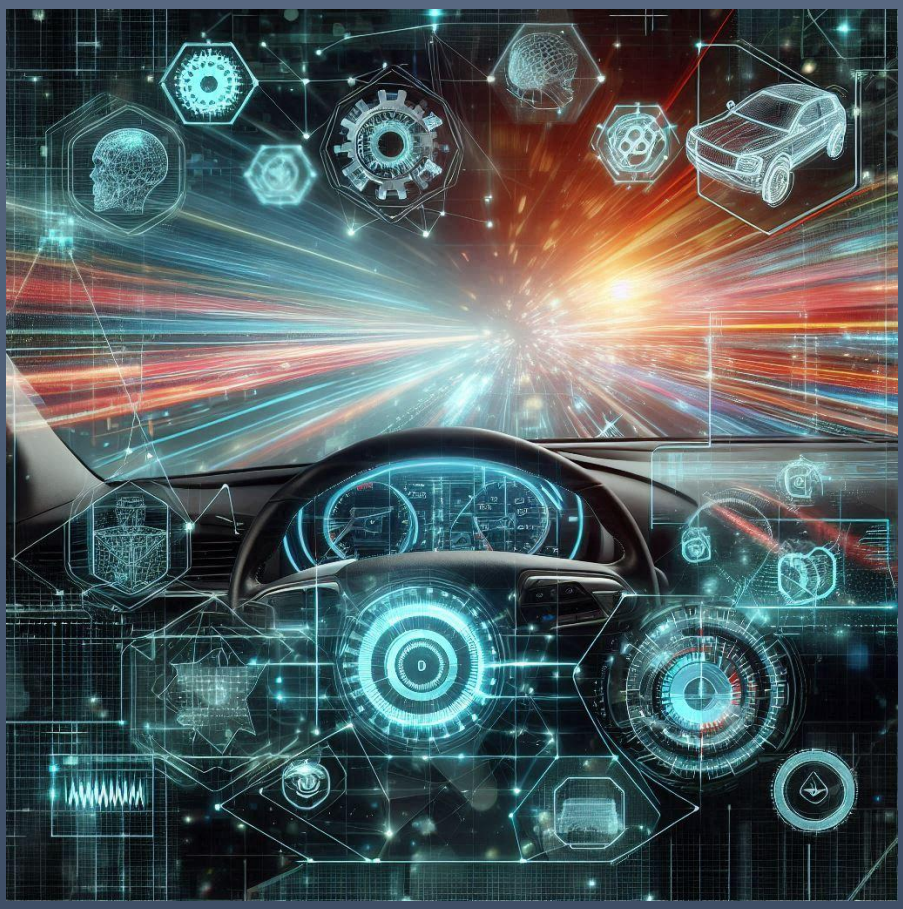
2025



Modelos de Inteligencia Artificial

Profesor/a: Águeda María López Moreno

PRÁCTICA 2.4.- LÓGICA DIFUSA

17/04/2025

PEDRO MANUEL GARCÍA ÁLVAREZ | int.marisma@gmail.com | CEIA | I.E.S. La Marisma

Tabla de contenido

1.- Introducción

1.1.- Contexto del problema

El control de frenado en vehículos es una parte esencial de los sistemas de seguridad activa que buscan prevenir accidentes de tráfico. En situaciones de emergencia, como la proximidad a un obstáculo o la necesidad de reducir la velocidad rápidamente, el sistema de frenado debe reaccionar de manera eficiente y adaptativa. Sin embargo, la toma de decisiones en estos contextos no siempre es un proceso simple ni completamente predecible, ya que depende de variables como la velocidad del vehículo, la distancia a un obstáculo y las condiciones de la carretera. A menudo, los sistemas tradicionales de control se basan en reglas estrictas o modelos lineales que no pueden manejar eficazmente la incertidumbre y la variabilidad inherente a estos escenarios.

La **lógica difusa** emerge como una herramienta poderosa para modelar este tipo de problemas. A diferencia de los sistemas clásicos, que se basan en un enfoque determinista, la lógica difusa permite manejar incertidumbres y tomar decisiones basadas en grados de certeza, en lugar de valores absolutos. Esta aproximación resulta especialmente útil en sistemas de control que deben lidiar con conceptos vagos o difusos, como "velocidad alta", "distancia corta" o "frenada fuerte".

En este contexto, el objetivo es desarrollar un sistema difuso que pueda simular el comportamiento de un vehículo al frenar en función de la velocidad y la distancia a un obstáculo. Esto se logrará a través de un sistema basado en variables lingüísticas, reglas de inferencia difusa y una interfaz gráfica que permita al usuario interactuar y visualizar el comportamiento del sistema.

1.2.- Objetivos del sistema

El principal objetivo de este trabajo es **desarrollar un sistema de control difuso** para simular y calcular la intensidad del frenado de un vehículo en función de dos variables de entrada principales: la velocidad y la distancia a un obstáculo. A través de este sistema, se busca proporcionar una solución más flexible y realista frente a los

sistemas tradicionales de control, permitiendo una mejor interpretación y respuesta ante situaciones complejas.

Los objetivos específicos del sistema son los siguientes:

1. **Implementar un controlador difuso para el cálculo de frenada**: Utilizar lógica difusa para modelar las variables de entrada (velocidad y distancia) y determinar el coeficiente de frenada adecuado. Este cálculo se basará en reglas de inferencia difusa que permitan una respuesta más adaptativa y gradual.
2. **Desarrollar una interfaz gráfica interactiva**: Crear una interfaz de usuario que permita introducir los valores de velocidad y distancia de manera sencilla. La interfaz también deberá mostrar gráficas interactivas de los conjuntos difusos y una simulación visual del comportamiento del vehículo al frenar.
3. **Visualizar el comportamiento del sistema**: Generar gráficas que muestren los conjuntos difusos para cada variable, así como una representación 3D de la superficie de respuesta del sistema. Además, la simulación gráfica deberá ilustrar el proceso de frenado del vehículo y cómo varía la distancia a medida que el coche frena.
4. **Validar el sistema de inferencia**: Asegurarse de que el sistema funcione correctamente en distintos escenarios, como velocidades bajas y altas, y distancias cortas o largas, comprobando que el coeficiente de frenada calculado sea realista y útil para la simulación.

Con estos objetivos, se pretende crear un sistema de control difuso que no solo sea útil para la simulación de frenado en vehículos, sino que también sirva como base para futuros desarrollos en el área de seguridad vehicular, permitiendo incorporar más variables y escenarios complejos en el control difuso.

2.- Fundamentos Teóricos

2.1.- Lógica difusa

La **lógica difusa** (fuzzy logic) fue introducida por **Lotfi Zadeh** en 1965 como una extensión de la lógica booleana clásica, con el objetivo de manejar la incertidumbre y la imprecisión inherente a muchos sistemas reales. En lugar de utilizar valores binarios

(verdadero o falso), la lógica difusa permite que los valores de verdad sean cualquier número entre 0 y 1, lo que significa que las proposiciones pueden ser parcialmente verdaderas o parcialmente falsas. Este enfoque permite modelar situaciones en las que las decisiones no son absolutas, sino que se toman en función de grados de certeza.

En un sistema basado en lógica difusa, las reglas y relaciones entre variables no son estrictamente determinísticas. Por ejemplo, en lugar de decir que "si la velocidad es alta, entonces la frenada debe ser fuerte", la lógica difusa permite expresar relaciones más suaves, como "si la velocidad es algo alta, entonces la frenada debe ser moderada". Esto es útil para modelar sistemas del mundo real donde los valores exactos no siempre son conocidos o relevantes, y donde las decisiones deben tomarse sobre la base de aproximaciones.

La lógica difusa se utiliza ampliamente en sistemas de control, especialmente en áreas como la automoción, la robótica y la electrónica, debido a su capacidad para manejar la incertidumbre y proporcionar soluciones más adaptativas.

2.2.- Variables lingüísticas y conjuntos difusos

En la lógica difusa, las **variables lingüísticas** son aquellas que toman valores no numéricos, sino que son representadas por palabras o términos, como "alto", "bajo", "calor", "frío", etc. Estas variables se definen a través de conjuntos difusos, los cuales son conjuntos matemáticos que permiten representar grados de pertenencia de un valor a un conjunto, en lugar de asignar solo valores binarios de pertenencia (pertenece o no pertenece).

Por ejemplo, la variable lingüística "velocidad" podría tomar los valores "baja", "media" o "alta", y estos valores estarían representados por conjuntos difusos que definen cómo un valor numérico (como 80 km/h) pertenece a cada uno de estos términos. Los conjuntos difusos no son límites estrictos; más bien, se solapan, permitiendo que un valor numérico pueda pertenecer a más de un conjunto con distintos grados de certeza. Esto es lo que hace que la lógica difusa sea más flexible y capaz de modelar sistemas más realistas.

Cada conjunto difuso se define mediante una **función de membresía**, que asigna un grado de pertenencia (un valor entre 0 y 1) a cada valor de entrada. Por ejemplo, para la variable "velocidad", la función de membresía puede asignar a un valor de 80 km/h un grado de pertenencia del 0.6 en "baja", 0.8 en "media" y 0.2 en "alta". Esto permite una interpretación más matizada de las variables que, en un enfoque tradicional, se manejarían de manera más rígida.

2.3.- Reglas de inferencia difusa

Las **reglas de inferencia difusa** son la base del razonamiento en los sistemas difusos. Estas reglas describen cómo se deben combinar las variables lingüísticas para tomar decisiones o realizar inferencias. Las reglas difusas siguen el formato "Si A es X y B es Y, entonces C es Z", donde A y B son las variables de entrada y C es la variable de salida. Sin embargo, a diferencia de las reglas de un sistema lógico tradicional, las reglas difusas permiten trabajar con valores imprecisos y grados de pertenencia.

Por ejemplo, una regla podría ser:

- "Si la velocidad es alta y la distancia es corta, entonces la frenada debe ser fuerte."

Para determinar el valor de salida (en este caso, la frenada), el sistema utiliza las funciones de membresía de las variables de entrada y las combina según la regla. En este caso, si la velocidad es "algo alta" y la distancia es "algo corta", se calcula el grado de certeza de esta regla para determinar el valor de la frenada.

Una vez que todas las reglas han sido evaluadas, los resultados de cada regla se combinan mediante un proceso conocido como **agregación**. Esto da como resultado un valor de salida difuso que luego se "defuzifica" (es decir, se convierte en un valor numérico preciso) utilizando técnicas como el método de **centroide**.

2.4.- Aplicaciones en sistemas de control

Los sistemas de control difuso han encontrado aplicaciones en una amplia variedad de áreas, especialmente en aquellos casos donde las condiciones del sistema son inciertas, variables o difíciles de modelar con precisión matemática. Algunos ejemplos incluyen:

1. **Control de temperatura**: En sistemas de calefacción o aire acondicionado, la lógica difusa puede usarse para ajustar la temperatura de manera continua, considerando variaciones en las condiciones ambientales y la respuesta del sistema de calefacción o refrigeración.
2. **Automoción**: En los sistemas de control de frenado y estabilidad de vehículos, la lógica difusa puede adaptarse a diferentes condiciones de manejo, como el tipo de carretera, la velocidad del vehículo o el estado de los frenos. Un ejemplo de ello es el sistema de control de frenado adaptativo, como el que se ha propuesto en este trabajo.
3. **Robótica**: Los robots que interactúan con su entorno, como los robots móviles o los brazos robóticos, pueden utilizar controladores difusos para manejar incertidumbres en la percepción del entorno y realizar tareas complejas como la navegación, el agarre o el ensamblaje.
4. **Electrodomésticos inteligentes**: En sistemas de electrodomésticos inteligentes, como lavadoras o frigoríficos, la lógica difusa puede optimizar el funcionamiento al considerar factores como la carga, la temperatura o el tiempo de operación, ajustando los parámetros de funcionamiento para obtener los mejores resultados.
5. **Control en procesos industriales**: En procesos de manufactura, donde existen muchas variables imprecisas o ruidosas (como temperatura, presión, velocidad de producción), los controladores difusos pueden ofrecer soluciones más robustas y adaptativas que los controladores clásicos.

El uso de la lógica difusa permite crear sistemas de control más flexibles, adaptativos y robustos, capaces de operar en entornos dinámicos y complejos, y de tomar decisiones más cercanas a las que haría un ser humano en situaciones de incertidumbre.

3.- Descripción del Sistema

3.1.- Variables de entrada y salida

En el sistema de control difuso que hemos implementado, las variables de entrada y salida son fundamentales para modelar el comportamiento del sistema de frenado de un vehículo. Las variables se definen de la siguiente manera:

- Variables de entrada:
  - **Velocidad**: La velocidad del vehículo es una de las principales variables de entrada al sistema. Se mide en kilómetros por hora (km/h) y determina cómo se debe ajustar la acción de frenado según la rapidez del vehículo. La velocidad se divide en varias categorías, como "muy despacio", "despacio", "normal", "rápido" y "muy rápido", basadas en valores numéricos.
  - **Distancia**: La distancia entre el vehículo y un obstáculo o el siguiente vehículo es la segunda variable de entrada. Se mide en metros (m) y refleja el espacio disponible para frenar antes de una posible colisión. Al igual que la velocidad, la distancia se clasifica en términos lingüísticos como "poca", "aceptable" y "mucha".
- Variable de salida:
  - **Frenada**: La variable de salida del sistema es el coeficiente de frenada, que determina la intensidad del frenado necesario para evitar una colisión o reducir la velocidad del vehículo. Se mide en una escala de 0 a 100, donde 0 significa que no se necesita frenado (frenada nula) y 100 significa que se requiere un frenado muy fuerte.

Estas tres variables están interrelacionadas mediante el sistema de reglas difusas, las cuales determinan el grado de frenada necesario en función de la velocidad y la distancia.

3.2.- Definición de conjuntos difusos

Los conjuntos difusos son cruciales para representar las variables de entrada y salida en términos lingüísticos, lo que permite modelar la incertidumbre y las transiciones suaves entre valores. Para cada variable, se han definido varios conjuntos difusos, que a su vez están asociados con funciones de membresía. A continuación, se detallan los conjuntos difusos utilizados en este sistema:

- Conjuntos difusos para la variable "Velocidad":
  - **Muy despacio**: Representa velocidades muy bajas, entre 30 km/h y 60 km/h.
  - **Despacio**: Representa velocidades bajas, entre 30 km/h y 90 km/h.
  - **Normal**: Representa velocidades medias, entre 60 km/h y 120 km/h.
  - **Rápido**: Representa velocidades altas, entre 90 km/h y 150 km/h.
  - **Muy rápido**: Representa velocidades muy altas, entre 120 km/h y 150 km/h.
- Conjuntos difusos para la variable "Distancia":
  - **Poca**: Representa distancias pequeñas, menores de 100 metros.
  - **Aceptable**: Representa distancias intermedias, entre 0 metros y 200 metros.
  - **Mucha**: Representa distancias grandes, cercanas a los 300 metros.
- Conjuntos difusos para la variable "Frenada":
  - **Nula**: Representa una frenada muy ligera o inexistente, con valores cercanos a 0.
  - **Ligera**: Representa una frenada moderada, con valores entre 20 y 60.
  - **Fuerte**: Representa una frenada intensa, con valores entre 60 y 100.

Cada uno de estos conjuntos difusos se ha definido mediante funciones de membresía **triangulares**, las cuales asignan a cada valor un grado de pertenencia a cada conjunto. Estas funciones permiten que un valor de entrada pertenezca parcialmente a varios conjuntos difusos.

3.3.- Reglas del sistema de control

El sistema de control difuso se basa en un conjunto de reglas de inferencia que relacionan las variables de entrada (velocidad y distancia) con la variable de salida (frenada). Estas reglas definen el comportamiento del sistema y determinan qué tan fuerte debe ser la acción de frenado, dependiendo de la velocidad del vehículo y la distancia con respecto al obstáculo. Las reglas que se han implementado son las siguientes:

1. Si la velocidad es "muy despacio" y la distancia es "mucha", entonces la frenada es "nula".
2. Si la velocidad es "muy despacio" y la distancia es "aceptable", entonces la frenada es "nula".
3. Si la velocidad es "muy despacio" y la distancia es "poca", entonces la frenada es "nula".
4. Si la velocidad es "despacio" y la distancia es "mucha", entonces la frenada es "nula".
5. Si la velocidad es "despacio" y la distancia es "aceptable", entonces la frenada es "ligera".
6. Si la velocidad es "despacio" y la distancia es "poca", entonces la frenada es "ligera".
7. Si la velocidad es "normal" y la distancia es "mucha", entonces la frenada es "nula".
8. Si la velocidad es "normal" y la distancia es "aceptable", entonces la frenada es "ligera".
9. Si la velocidad es "normal" y la distancia es "poca", entonces la frenada es "fuerte".
10. Si la velocidad es "rápido" y la distancia es "mucha", entonces la frenada es "nula".
11. Si la velocidad es "rápido" y la distancia es "aceptable", entonces la frenada es "ligera".
12. Si la velocidad es "rápido" y la distancia es "poca", entonces la frenada es "fuerte".
13. Si la velocidad es "muy rápido" y la distancia es "mucha", entonces la frenada es "nula".
14. Si la velocidad es "muy rápido" y la distancia es "aceptable", entonces la frenada es "ligera".
15. Si la velocidad es "muy rápido" y la distancia es "poca", entonces la frenada es "fuerte".

Estas reglas permiten calcular un valor de frenada adecuado, basándose en las condiciones actuales del vehículo, asegurando una respuesta fluida y adaptativa del sistema. La combinación de estas reglas se lleva a cabo mediante el proceso de inferencia difusa, utilizando la técnica de **agregación**.

3.4.- Arquitectura general del sistema

La arquitectura general del sistema se divide en varias etapas clave que permiten la entrada de datos, el procesamiento de la lógica difusa y la salida de resultados:

1. **Entrada de datos**: El sistema toma como entrada los valores de las variables "velocidad" y "distancia". Estos valores son proporcionados por el usuario a través de una interfaz gráfica.
2. **Definición de variables lingüísticas**: Las variables de entrada se traducen a términos lingüísticos mediante las funciones de membresía, y se asignan a los conjuntos difusos correspondientes.
3. **Evaluación de reglas**: Una vez que las variables de entrada han sido evaluadas, el sistema aplica las reglas difusas para determinar el grado de pertenencia a los diferentes conjuntos de salida.
4. **Agregación de resultados**: Los resultados de todas las reglas se combinan en un único valor difuso mediante el proceso de agregación.
5. **Defuzzificación**: El valor de salida difuso se convierte en un valor preciso mediante un proceso de defuzzificación (por ejemplo, usando el método de centroide), lo que da como resultado un valor numérico de frenada.
6. **Salida de datos**: Finalmente, el sistema muestra el valor de frenada calculado, interpretando si es necesario aplicar una frenada nula, ligera o fuerte.

La arquitectura también incluye una interfaz gráfica de usuario (GUI), que permite al usuario interactuar con el sistema, introducir los valores de entrada y visualizar los resultados de forma clara y comprensible.

4.- Implementación en Python

4.1.- Bibliotecas utilizadas

Para implementar el sistema de control difuso, se utilizan varias bibliotecas en Python que facilitan el desarrollo y la simulación de sistemas de control basados en lógica difusa. Las principales bibliotecas utilizadas en este trabajo son:

- **NumPy**: Esta biblioteca se utiliza para realizar cálculos matemáticos eficientes, como la manipulación de matrices y la implementación de funciones numéricas necesarias para el sistema de control.
- **Matplotlib**: Se usa para la visualización gráfica de los resultados del sistema de control, como la representación de los conjuntos difusos y la superficie de respuesta del sistema.
- **scikit-fuzzy**: Esta biblioteca es crucial para el manejo de la lógica difusa en Python. Proporciona herramientas para la creación de funciones de membresía, la definición de conjuntos difusos, la evaluación de reglas y la defuzzificación.
- **Pandas**: Aunque no es estrictamente necesaria para la parte de lógica difusa, esta biblioteca se usa para la manipulación de datos (por ejemplo, si se desea almacenar los resultados de las simulaciones en un formato estructurado como DataFrame).

4.2.- Definición de las variables difusas

Las variables difusas son la base del sistema, y se definen utilizando la biblioteca **scikit-fuzzy**. Para cada variable de entrada (como velocidad y distancia) y salida (como frenada), se crean funciones de membresía que modelan cómo los valores numéricos se asignan a los términos lingüísticos (por ejemplo, "despacio", "normal", "fuerte", etc.).

4.3.- Definición de reglas

Las reglas del sistema son la parte clave de la lógica difusa. Usamos las funciones de membresía definidas anteriormente para crear reglas basadas en las relaciones entre las variables de entrada y

salida. En Python, podemos definir estas reglas utilizando operadores lógicos de **scikit-fuzzy**.

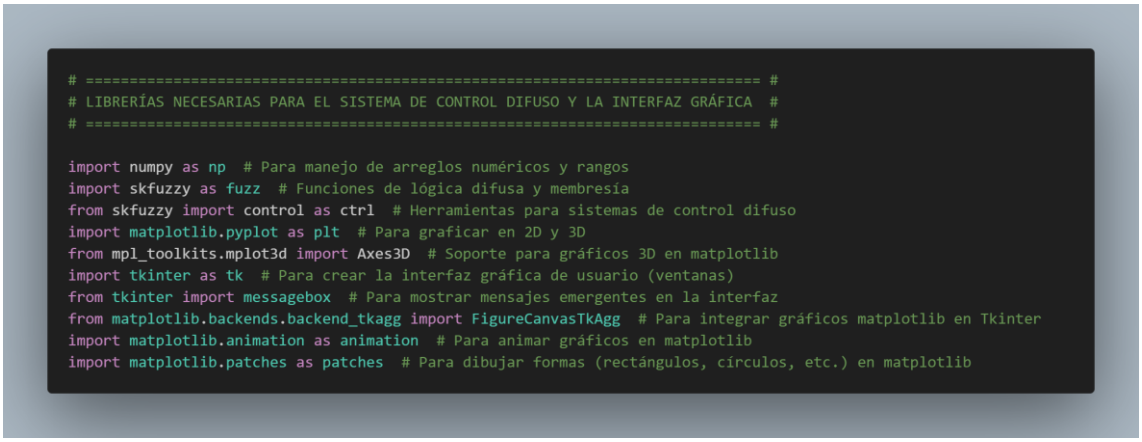
4.4.- Simulación del sistema de control

Una vez que las variables difusas y las reglas están definidas, podemos simular el sistema de control utilizando la **inferencia difusa** para determinar el valor de salida.

4.5.- Visualización de conjuntos y superficie de respuesta

La visualización de los conjuntos difusos y la superficie de respuesta es importante para comprender cómo interactúan las variables. Usamos **matplotlib** para graficar los conjuntos difusos y la superficie de respuesta de las variables.

4.6.- Estructura y explicación del código fuente



```
# ========================================================================== #
# LIBRERÍAS NECESARIAS PARA EL SISTEMA DE CONTROL DIFUSO Y LA INTERFAZ GRÁFICA  #
# ========================================================================== #

import numpy as np  # Para manejo de arreglos numéricos y rangos
import skfuzzy as fuzz  # Funciones de lógica difusa y membresía
from skfuzzy import control as ctrl  # Herramientas para sistemas de control difuso
import matplotlib.pyplot as plt  # Para graficar en 2D y 3D
from mpl_toolkits.mplot3d import Axes3D  # Soporte para gráficos 3D en matplotlib
import tkinter as tk  # Para crear la interfaz gráfica de usuario (ventanas)
from tkinter import messagebox  # Para mostrar mensajes emergentes en la interfaz
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg  # Para integrar gráficos matplotlib en Tkinter
import matplotlib.animation as animation  # Para animar gráficos en matplotlib
import matplotlib.patches as patches  # Para dibujar formas (rectángulos, círculos, etc.) en matplotlib
```

Esta sección del código importa todas las librerías necesarias para crear un sistema de control difuso en Python, visualizar sus resultados con gráficos y animaciones, y ofrecer una interfaz gráfica interactiva al usuario.

```
# === Definición de variables difusas ===

# Se definen las variables difusas que se utilizarán en el sistema de control.
velocidad = ctrl.Antecedent(np.arange(30, 151, 1), 'velocidad')  # Velocidad en km/h
distancia = ctrl.Antecedent(np.arange(0, 301, 1), 'distancia')  # Distancia en metros
frenada = ctrl.Consequent(np.arange(0, 101, 1), 'frenada')  # Coeficiente de frenada
```

Esta parte del código crea las variables difusas de entrada (velocidad y distancia) y de salida (coeficiente de frenada) que serán utilizadas por el sistema de control difuso, definiendo sus rangos de valores posibles.

```
# Definición de las funciones de membresía de cada variable
# Velocidad
velocidad['muy_despacio'] = fuzz.trimf(velocidad.universe, [30, 30, 60])
velocidad['despacio'] = fuzz.trimf(velocidad.universe, [30, 60, 90])
velocidad['normal'] = fuzz.trimf(velocidad.universe, [60, 90, 120])
velocidad['rapido'] = fuzz.trimf(velocidad.universe, [90, 120, 150])
velocidad['muy_rapido'] = fuzz.trimf(velocidad.universe, [120, 150, 150])
```

Este código define los conjuntos difusos de la variable velocidad mediante funciones de membresía triangulares, asignando etiquetas como 'muy_despacio', 'despacio', 'normal', 'rapido' y 'muy_rapido' a diferentes rangos de valores.

```
# Distancia
distancia['poca'] = fuzz.trimf(distancia.universe, [0, 0, 100])
distancia['aceptable'] = fuzz.trimf(distancia.universe, [0, 100, 200])
distancia['mucha'] = fuzz.trimf(distancia.universe, [100, 300, 300])
```

Este código define los conjuntos difusos para la variable distancia mediante funciones de membresía triangulares, clasificando la distancia como 'poca', 'aceptable' o 'mucha' según su valor.

```
# Frenada
frenada['nula'] = fuzz.trimf(frenada.universe, [0, 0, 40])
frenada['ligera'] = fuzz.trimf(frenada.universe, [20, 50, 80])
frenada['fuerte'] = fuzz.trimf(frenada.universe, [60, 100, 100])
```

Este código define los conjuntos difusos para la variable de salida 'frenada' usando funciones de membresía triangulares, clasificando el coeficiente de frenada como 'nula', 'ligera' o 'fuerte' según su valor.

```
# Reglas de control difuso que relacionan las variables de entrada y salida
reglas = [
    ctrl.Rule(velocidad['muy_despacio'] & distancia['mucha'], frenada['nula']),
    ctrl.Rule(velocidad['muy_despacio'] & distancia['aceptable'], frenada['nula']),
    ctrl.Rule(velocidad['muy_despacio'] & distancia['poca'], frenada['nula']),
    ctrl.Rule(velocidad['despacio'] & distancia['mucha'], frenada['nula']),
    ctrl.Rule(velocidad['despacio'] & distancia['aceptable'], frenada['ligera']),
    ctrl.Rule(velocidad['despacio'] & distancia['poca'], frenada['ligera']),
    ctrl.Rule(velocidad['normal'] & distancia['mucha'], frenada['nula']),
    ctrl.Rule(velocidad['normal'] & distancia['aceptable'], frenada['ligera']),
    ctrl.Rule(velocidad['normal'] & distancia['poca'], frenada['fuerte']),
    ctrl.Rule(velocidad['rapido'] & distancia['mucha'], frenada['nula']),
    ctrl.Rule(velocidad['rapido'] & distancia['aceptable'], frenada['ligera']),
    ctrl.Rule(velocidad['rapido'] & distancia['poca'], frenada['fuerte']),
    ctrl.Rule(velocidad['muy_rapido'] & distancia['mucha'], frenada['nula']),
    ctrl.Rule(velocidad['muy_rapido'] & distancia['aceptable'], frenada['ligera']),
    ctrl.Rule(velocidad['muy_rapido'] & distancia['poca'], frenada['fuerte'])
]
```

Este código define las reglas del sistema de control difuso, estableciendo cómo se debe calcular la frenada en función de las combinaciones de velocidad y distancia mediante sentencias lógicas IF-THEN.

```
# Se crea el sistema de control difuso utilizando las reglas definidas
sistema_control = ctrl.ControlSystem(reglas)
sistema = ctrl.ControlSystemSimulation(sistema_control)
```

Esta parte del código crea el sistema de control difuso a partir de las reglas definidas y prepara una simulación para calcular la salida (frenada) según los valores de entrada proporcionados.

```python
# Función para calcular el coeficiente de frenada según la velocidad y distancia
def calcular_frenada(vel, dist):
    if vel < 30 or vel > 150:  # Validación de la velocidad
        return None, "La velocidad debe estar entre 30 y 150 km/h"
    if dist < 0 or dist > 300:  # Validación de la distancia
        return None, "La distancia debe estar entre 0 y 300 metros"
    sistema.input['velocidad'] = vel  # Asigna la velocidad al sistema de control
    sistema.input['distancia'] = dist  # Asigna la distancia al sistema de control
    sistema.compute()  # Realiza la computación del sistema de control
    return sistema.output['frenada'], None  # Retorna el resultado de la frenada
```

Esta función recibe la velocidad y distancia, valida que estén dentro de los rangos permitidos, asigna estos valores al sistema difuso, ejecuta la inferencia y devuelve el coeficiente de frenada calculado.

```python
# Función para obtener la figura con las funciones de membresía de las variables
def get_conjuntos_figure():
    fig, axs = plt.subplots(3, 1, figsize=(6, 8))  # Crear una figura con 3 subgráficos
    # Graficar las funciones de membresía de velocidad
    for name in velocidad.terms:
        axs[0].plot(velocidad.universe, velocidad[name].mf, label=name)
    axs[0].set_title('Velocidad')
    axs[0].legend()
    axs[0].set_xlim([30, 150])  # Limites en el eje X para velocidad
    # Graficar las funciones de membresía de distancia
    for name in distancia.terms:
        axs[1].plot(distancia.universe, distancia[name].mf, label=name)
    axs[1].set_title('Distancia')
    axs[1].legend()
    axs[1].set_xlim([0, 300])  # Limites en el eje X para distancia
    # Graficar las funciones de membresía de frenada
    for name in frenada.terms:
        axs[2].plot(frenada.universe, frenada[name].mf, label=name)
    axs[2].set_title('Frenada')
    axs[2].legend()
    axs[2].set_xlim([0, 100])  # Limites en el eje X para frenada
    plt.tight_layout()  # Ajustar el layout de los subgráficos
    return fig  # Retorna la figura con las funciones de membresía
```

Esta función genera y retorna una figura con tres subgráficos que muestran las funciones de membresía de las variables velocidad, distancia y frenada, permitiendo visualizar cómo se definen los conjuntos difusos del sistema.

```python
# Función para obtener la figura de la superficie de respuesta
def get_superficie_figure():
    x = np.linspace(30, 150, 25)  # Valores de velocidad
    y = np.linspace(0, 300, 25)  # Valores de distancia
    xg, yg = np.meshgrid(x, y)  # Crear una malla de valores para las dos variables
    z = np.zeros_like(xg)  # Inicializar la matriz de salida (frenada)
    # Calcular el coeficiente de frenada para cada combinación de velocidad y distancia
    for i in range(len(xg)):
        for j in range(len(xg[0])):
            sistema.input['velocidad'] = xg[i, j]
            sistema.input['distancia'] = yg[i, j]
            sistema.compute()  # Calcular el resultado
            z[i, j] = sistema.output['frenada']
    fig = plt.figure(figsize=(6, 5))  # Crear la figura para el gráfico 3D
    ax = fig.add_subplot(111, projection='3d')  # Crear el subplot 3D
    surf = ax.plot_surface(xg, yg, z, cmap='viridis', rstride=1, cstride=1)  # Graficar la superficie
    ax.set_xlabel('Velocidad (km/h)')  # Etiqueta para el eje X
    ax.set_ylabel('Distancia (m)')  # Etiqueta para el eje Y
    ax.set_zlabel('Coef. Frenada')  # Etiqueta para el eje Z
    ax.set_title('Superficie de Respuesta')  # Título del gráfico
    fig.colorbar(surf, shrink=0.5, aspect=5)  # Agregar una barra de color
    return fig  # Retorna la figura con la superficie de respuesta
```

Esta función calcula y genera una figura 3D que muestra la superficie de respuesta del sistema difuso, representando cómo varía el coeficiente de frenada en función de la velocidad y la distancia.

```python
# Función para simular el coche en la interfaz gráfica
def simular_coche(canvas, ax, vel, dist, frenada_val):
    ax.clear()  # Limpiar el gráfico actual
    ax.set_xlim(0, 320)  # Establecer los límites del eje X (distancia)
    ax.set_ylim(-5, 5)  # Establecer los límites del eje Y
    ax.set_yticks([])  # Eliminar las marcas en el eje Y
    ax.set_xlabel("Distancia (m)")  # Etiqueta para el eje X
    ax.set_title("Simulación de frenada")  # Título del gráfico
```

Esta función prepara el área de la gráfica para simular visualmente el movimiento y frenado del coche, limpiando el gráfico y configurando los ejes y etiquetas antes de dibujar la animación.

```python
# Crear un obstáculo en la simulación
obstaculo = patches.Rectangle((dist - 3, -1), 6, 2, color="red", label="Obstáculo")
ax.add_patch(obstaculo)
```

Estas líneas crean y añaden un rectángulo rojo al gráfico para representar el obstáculo (el coche de delante) en la simulación.

```
# Definir el tamaño y las ruedas del coche
coche_width = 8  # Ancho del coche
coche_height = 3  # Alto del coche
rueda_radius = 1  # Radio de las ruedas
```

Estas líneas definen las dimensiones del coche y el tamaño de las ruedas que se dibujarán en la simulación.

```
# Crear la carrocería y las ruedas del coche
carroceria = patches.Rectangle((0, -coche_height/2), coche_width, coche_height, color='blue', label='Coche')
rueda_delantera = patches.Circle((coche_width*0.75, -coche_height/2 - rueda_radius), rueda_radius, color='black')
rueda_trasera = patches.Circle((coche_width*0.25, -coche_height/2 - rueda_radius), rueda_radius, color='black')

ax.add_patch(carroceria)  # Añadir la carrocería
ax.add_patch(rueda_delantera)  # Añadir la rueda delantera
ax.add_patch(rueda_trasera)  # Añadir la rueda trasera

ax.legend(loc="upper right")  # Mostrar la leyenda

t_total = 5  # Tiempo total de simulación
frames = 50  # Número de frames de animación
dt = t_total / frames  # Tiempo por frame
pos = 0  # Posición inicial
v = vel / 3.6  # Convertir la velocidad a m/s
frenada_f = frenada_val / 100  # Coeficiente de frenada en escala de 0 a 1

a = -frenada_f * 8  # Aceleración negativa por frenado
posiciones = []  # Lista para almacenar las posiciones del coche
```

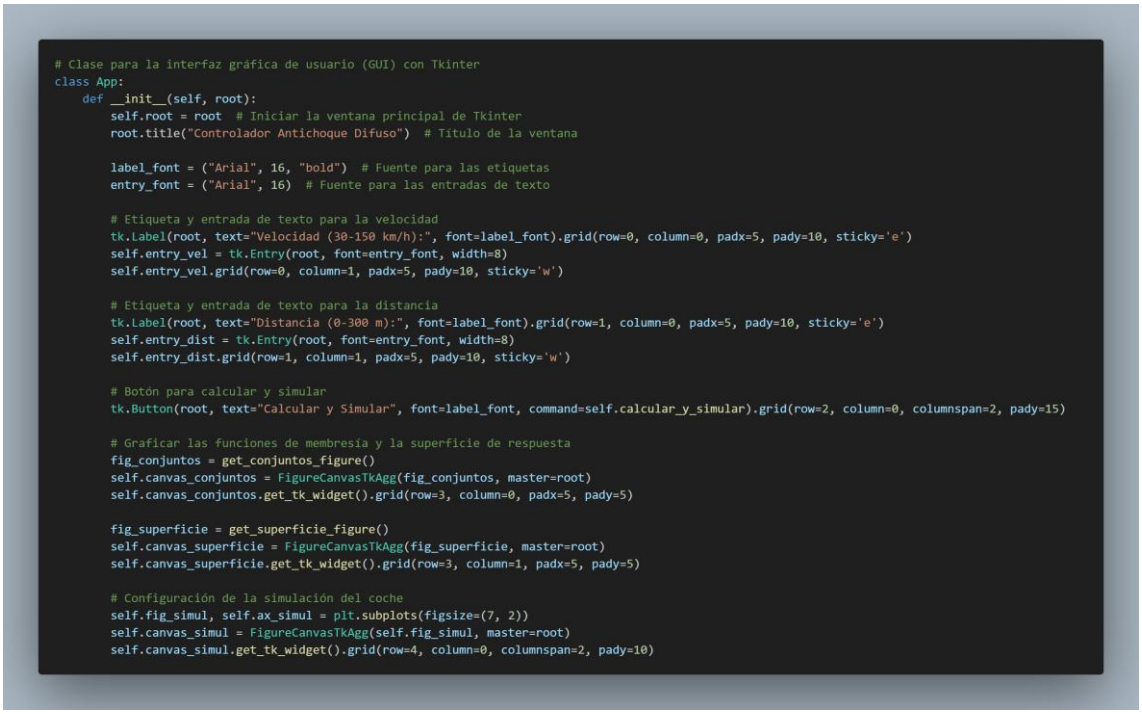
Estas líneas crean y añaden al gráfico la carrocería y las ruedas del coche, configuran la leyenda y establecen los parámetros de la animación, como el tiempo total, número de frames, velocidad inicial, coeficiente de frenada y aceleración.

```
# Simular el movimiento del coche
for i in range(frames):
    posiciones.append(pos)
    v = max(0, v + a * dt)  # Asegurarse de que la velocidad no sea negativa
    pos = pos + v * dt  # Calcular la nueva posición
    if pos >= dist - coche_width:  # Detenerse si llega al obstáculo
        pos = dist - coche_width
        posiciones += [pos] * (frames - len(posiciones))  # Rellenar con la posición final
        break
```

Este bloque simula el movimiento del coche fotograma a fotograma, actualizando su posición y velocidad en cada paso, y deteniéndolo cuando llega al obstáculo para evitar que lo atraviese.

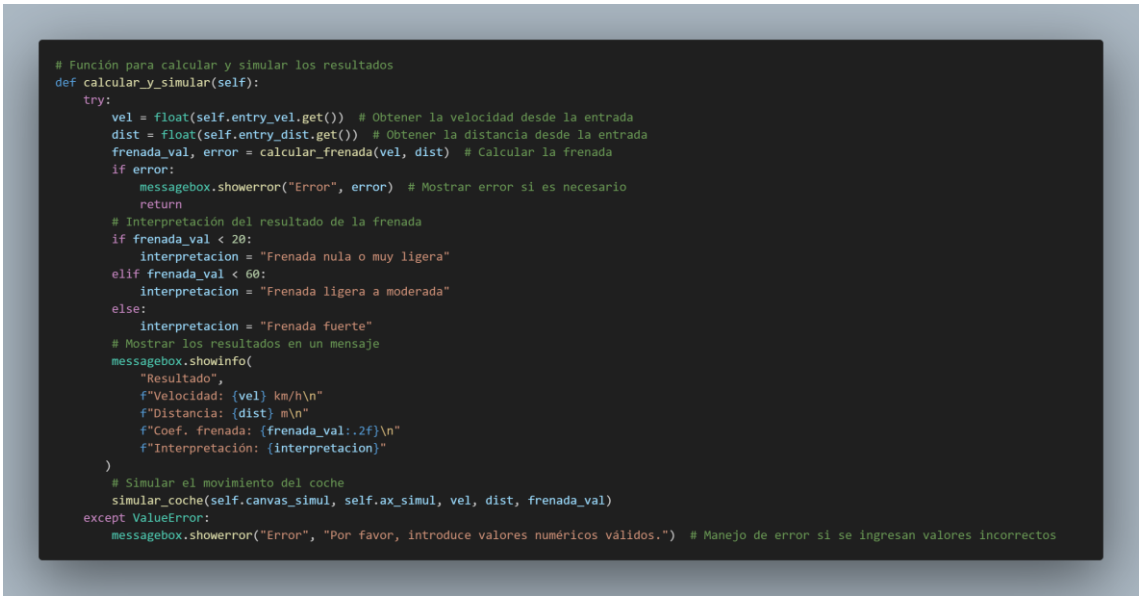
```
# Función de animación para mover el coche
def animate(i):
    carroceria.set_xy((posiciones[i], -coche_height/2))
    rueda_delantera.center = (posiciones[i] + coche_width*0.75, -coche_height/2 - rueda_radius)
    rueda_trasera.center = (posiciones[i] + coche_width*0.25, -coche_height/2 - rueda_radius)
    return carroceria, rueda_delantera, rueda_trasera

ani = animation.FuncAnimation(canvas.figure, animate, frames=len(posiciones), interval=50, blit=True, repeat=False)
canvas.draw()  # Actualizar el gráfico
```

Esta función interna actualiza la posición de la carrocería y las ruedas del coche en cada fotograma de la animación, y luego se utiliza para crear y mostrar la animación del movimiento del coche en la interfaz gráfica.



```python
# Clase para la interfaz gráfica de usuario (GUI) con Tkinter
class App:
    def __init__(self, root):
        self.root = root  # Iniciar la ventana principal de Tkinter
        root.title("Controlador Antichoque Difuso")  # Título de la ventana

        label_font = ("Arial", 16, "bold")  # Fuente para las etiquetas
        entry_font = ("Arial", 16)  # Fuente para las entradas de texto

        # Etiqueta y entrada de texto para la velocidad
        tk.Label(root, text="Velocidad (30-150 km/h):", font=label_font).grid(row=0, column=0, padx=5, pady=10, sticky='e')
        self.entry_vel = tk.Entry(root, font=entry_font, width=8)
        self.entry_vel.grid(row=0, column=1, padx=5, pady=10, sticky='w')

        # Etiqueta y entrada de texto para la distancia
        tk.Label(root, text="Distancia (0-300 m):", font=label_font).grid(row=1, column=0, padx=5, pady=10, sticky='e')
        self.entry_dist = tk.Entry(root, font=entry_font, width=8)
        self.entry_dist.grid(row=1, column=1, padx=5, pady=10, sticky='w')

        # Botón para calcular y simular
        tk.Button(root, text="Calcular y Simular", font=label_font, command=self.calcular_y_simular).grid(row=2, column=0, columnspan=2, pady=15)

        # Graficar las funciones de membresía y la superficie de respuesta
        fig_conjuntos = get_conjuntos_figure()
        self.canvas_conjuntos = FigureCanvasTkAgg(fig_conjuntos, master=root)
        self.canvas_conjuntos.get_tk_widget().grid(row=3, column=0, padx=5, pady=5)

        fig_superficie = get_superficie_figure()
        self.canvas_superficie = FigureCanvasTkAgg(fig_superficie, master=root)
        self.canvas_superficie.get_tk_widget().grid(row=3, column=1, padx=5, pady=5)

        # Configuración de la simulación del coche
        self.fig_simul, self.ax_simul = plt.subplots(figsize=(7, 2))
        self.canvas_simul = FigureCanvasTkAgg(self.fig_simul, master=root)
        self.canvas_simul.get_tk_widget().grid(row=4, column=0, columnspan=2, pady=10)
```

Esta clase define la interfaz gráfica de usuario usando Tkinter, creando los campos de entrada para velocidad y distancia, botones, y áreas para mostrar los gráficos de funciones de membresía, la superficie de respuesta difusa y la simulación animada del coche.



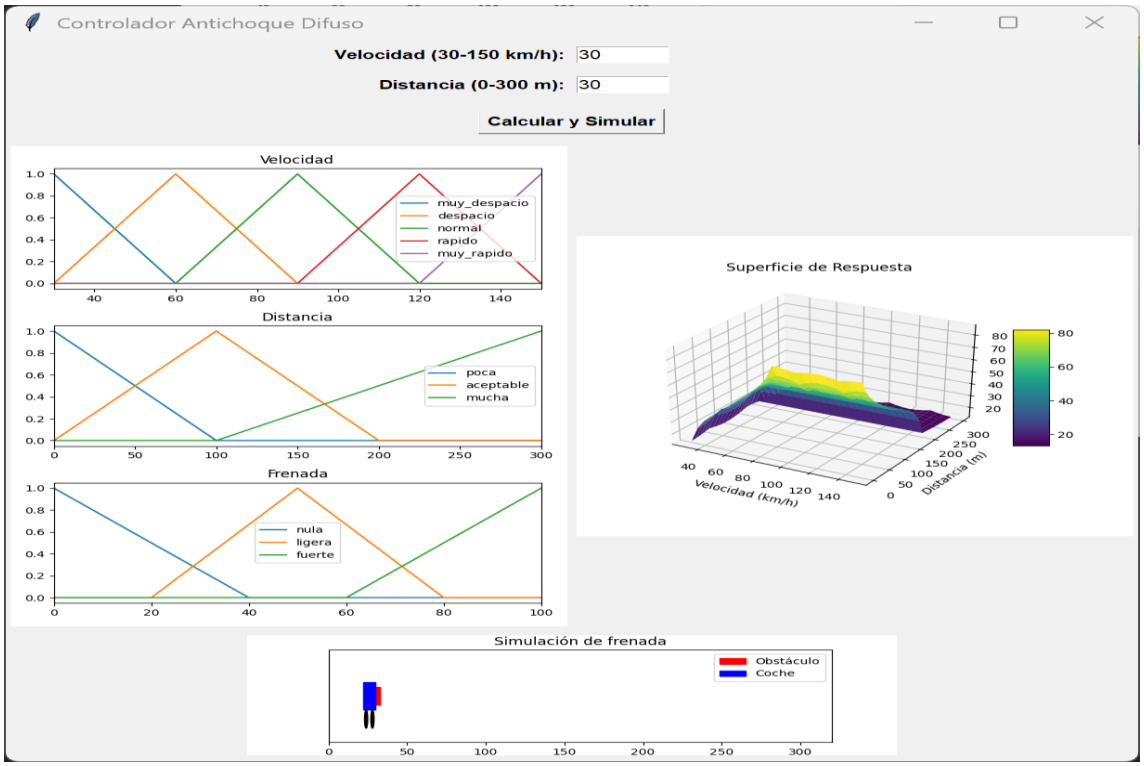
```python
# Función para calcular y simular los resultados
def calcular_y_simular(self):
    try:
        vel = float(self.entry_vel.get())  # Obtener la velocidad desde la entrada
        dist = float(self.entry_dist.get())  # Obtener la distancia desde la entrada
        frenada_val, error = calcular_frenada(vel, dist)  # Calcular la frenada
        if error:
            messagebox.showerror("Error", error)  # Mostrar error si es necesario
            return
        # Interpretación del resultado de la frenada
        if frenada_val < 20:
            interpretacion = "Frenada nula o muy ligera"
        elif frenada_val < 60:
            interpretacion = "Frenada ligera a moderada"
        else:
            interpretacion = "Frenada fuerte"
        # Mostrar los resultados en un mensaje
        messagebox.showinfo(
            "Resultado",
            f"Velocidad: {vel} km/h\n"
            f"Distancia: {dist} m\n"
            f"Coef. frenada: {frenada_val:.2f}\n"
            f"Interpretación: {interpretacion}"
        )
        # Simular el movimiento del coche
        simular_coche(self.canvas_simul, self.ax_simul, vel, dist, frenada_val)
    except ValueError:
        messagebox.showerror("Error", "Por favor, introduce valores numéricos válidos.")  # Manejo de error si se ingresan valores incorrectos
```

Esta función obtiene los valores de velocidad y distancia ingresados por el usuario, calcula el coeficiente de frenada usando el sistema difuso, muestra el resultado con una interpretación, y ejecuta la simulación visual del coche; además, maneja errores de entrada no numérica o fuera de rango.

```python
# Iniciar la aplicación
if __name__ == "__main__":
    root = tk.Tk()  # Crear la ventana principal de Tkinter
    app = App(root)  # Crear la instancia de la aplicación
    root.mainloop()  # Ejecutar el bucle principal de la interfaz gráfica
```

Este bloque inicia la aplicación creando la ventana principal de Tkinter, instanciando la interfaz gráfica y ejecutando el bucle principal para mostrar la GUI y responder a las acciones del usuario.

4.7.- Pantalla del programa



Esta interfaz gráfica muestra un sistema antichoque de lógica difusa, visualizando los conjuntos difusos definidos, la superficie de respuesta global del sistema, y una simulación para los valores de entrada actuales de velocidad (30 km/h) y distancia (30 m).

Para una velocidad de 30 km/h y una distancia de 30 m, el sistema calcula un coeficiente de frenada de 14.26, lo que se interpreta como una "Frenada nula o muy ligera".

5.- Interfaz Gráfica (GUI)

El sistema de control difuso implementado se complementa con una **interfaz gráfica de usuario (GUI)** desarrollada con **Tkinter**, una librería estándar de Python para la creación de interfaces gráficas. Esta interfaz facilita la interacción del usuario con el sistema, permitiéndole ingresar valores, observar los gráficos de los conjuntos difusos, interpretar los resultados del sistema, y visualizar la simulación de la frenada.

5.1.- Uso de Tkinter

Tkinter se utiliza para construir y organizar la ventana principal de la aplicación y sus distintos elementos (widgets), como etiquetas, campos de entrada, botones y contenedores para gráficos.

Se emplean los siguientes elementos clave:

- **Tk()**: crea la ventana principal.
- **Label**: muestra texto explicativo (como "Velocidad" y "Distancia").
- **Entry**: permite al usuario ingresar los valores numéricos de velocidad y distancia.
- **Button**: al hacer clic, ejecuta la función que calcula la frenada y lanza la simulación.
- **messagebox**: muestra cuadros emergentes informativos o de error al usuario.

- **FigureCanvasTkAgg**: permite insertar gráficos de **matplotlib** directamente dentro de la ventana de Tkinter, integrando así visualizaciones dentro de la interfaz.

5.2.- Entradas del usuario

El usuario debe proporcionar dos datos clave:

- **Velocidad (entre 30 y 150 km/h)**.
- **Distancia al obstáculo (entre 0 y 300 metros)**.

Estas entradas se capturan mediante campos **Entry**. Cuando se presiona el botón "Calcular y Simular", el programa:

1. Verifica que los datos sean válidos.
2. Valida que estén dentro de los rangos establecidos por el sistema difuso.
3. Llama al motor de inferencia difusa para calcular el valor del coeficiente de frenada.

5.3.- Resultados e interpretación

Una vez procesados los datos:

- El sistema muestra el **coeficiente de frenada** calculado.
- Se interpreta el resultado con una etiqueta textual, por ejemplo:
  - "FRENADA NULA O MUY LIGERA"
  - "FRENADA LIGERA A MODERADA"
  - "FRENADA FUERTE"

Estos resultados se muestran mediante una **ventana emergente** (**messagebox.showinfo**), que presenta al usuario los valores ingresados, el resultado del sistema difuso y su interpretación.

5.4.- Animación de la simulación

La interfaz también incluye una **simulación visual animada**, desarrollada con **matplotlib.animation**. Esta animación representa un coche que se desplaza hacia un obstáculo y frena en función del valor calculado por el sistema difuso.

Características de la simulación:

- El coche es representado por un rectángulo azul con ruedas negras.
- El obstáculo se muestra como un rectángulo rojo.
- La posición del coche varía con el tiempo según:
  - La velocidad inicial.
  - El coeficiente de frenada (usado para calcular la desaceleración).
- La animación se detiene si el coche alcanza o se aproxima al obstáculo.

La animación se actualiza en tiempo real dentro de la interfaz utilizando **FuncAnimation** y se dibuja en el canvas embebido (**FigureCanvasTkAgg**).

6.- Resultados y Pruebas

Para validar el funcionamiento del sistema de control difuso, se han realizado diversas **pruebas con casos representativos**, abarcando diferentes combinaciones de velocidad y distancia. El objetivo es comprobar que el sistema responde de manera coherente con la lógica esperada, activando distintos niveles de frenado según la situación.

6.1.- Casos de prueba

Se han utilizado múltiples combinaciones de entrada para observar el comportamiento del sistema. A continuación, se muestran algunos casos de prueba relevantes:

Velocidad (km/h)	Distancia (m)	Coef. Frenada	Interpretación esperada
40	250	14.44	Frenada nula o muy ligera
60	100	50.00	Frenada ligera a moderada
90	50	63,23	Frenada fuerte
130	200	15.56	Frenada nula o muy ligera
150	20	73.45	Frenada fuerte
80	290	14.44	Frenada nula o muy ligera
100	150	42.06	Frenada ligera a moderada

Estos casos cubren situaciones comunes: velocidades bajas con distancias amplias (donde no se necesita frenar) hasta situaciones

críticas (alta velocidad y poca distancia) que exigen una respuesta intensa de frenado.

6.2.- Análisis de salidas del sistema

El sistema genera un valor de salida difuso continuo que representa el **nivel de frenado** necesario, en un rango de 0 a 100. El análisis de los resultados muestra lo siguiente:

- Para **bajas velocidades** y **altas distancias**, la salida es cercana a 0: el sistema entiende que no es necesario frenar.
- A medida que la **distancia disminuye**, la salida se incrementa progresivamente, incluso para velocidades moderadas.
- A **altas velocidades** con **distancias cortas**, el sistema responde con una salida alta, lo que representa una **frenada fuerte**, como se espera.
- Las reglas difusas logran transiciones suaves entre niveles de frenado, lo que evita saltos bruscos e ilógicos en la salida.

Esto demuestra que el sistema es **coherente**, **gradual y sensible al contexto**.

6.3.- Comportamiento esperado del coche

La simulación animada del coche representa visualmente la reacción ante diferentes situaciones. El comportamiento observado concuerda con los resultados del sistema difuso:

- Cuando la frenada es nula o baja, el coche **continúa su marcha casi sin reducir la velocidad**.
- Cuando el sistema activa una frenada moderada, el coche **disminuye su velocidad gradualmente**, deteniéndose más cerca del obstáculo.
- En situaciones de frenada fuerte, el coche **frena bruscamente**, deteniéndose rápidamente, incluso antes de alcanzar la distancia crítica.

Además:

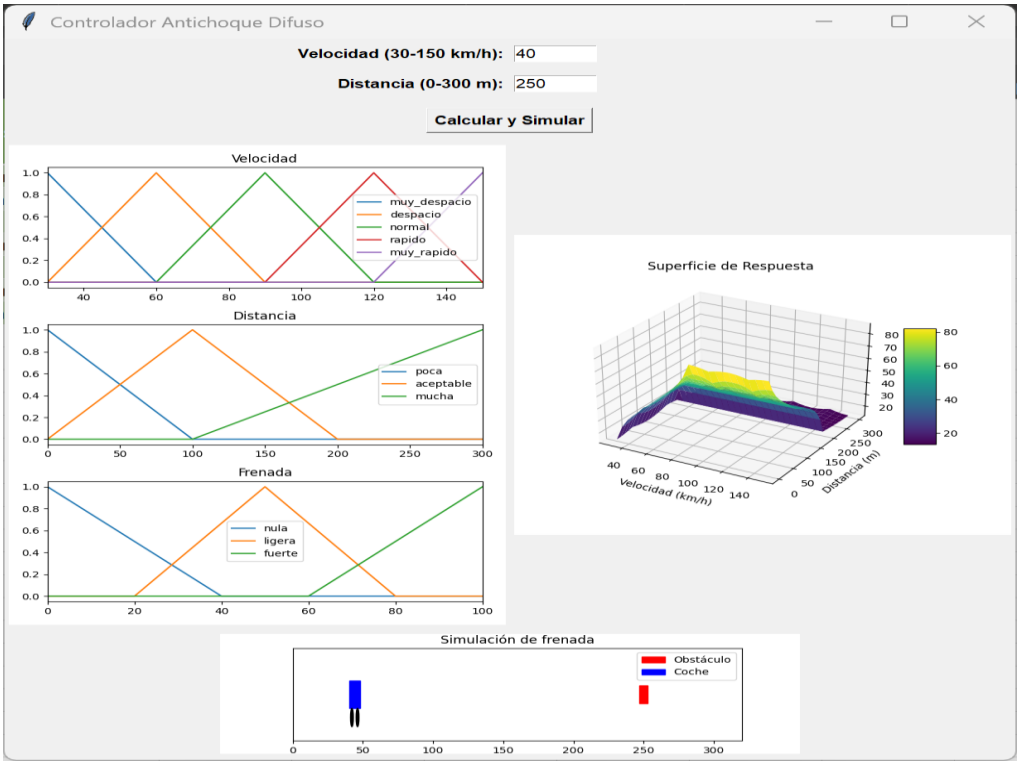
- El coche **no atraviesa el obstáculo**, gracias al control implementado en la lógica de animación.

- La simulación ofrece una representación intuitiva del nivel de frenado aplicado, ayudando al usuario a **visualizar el impacto del sistema difuso** sobre una situación real.

6.4.- Pantalla de las pruebas

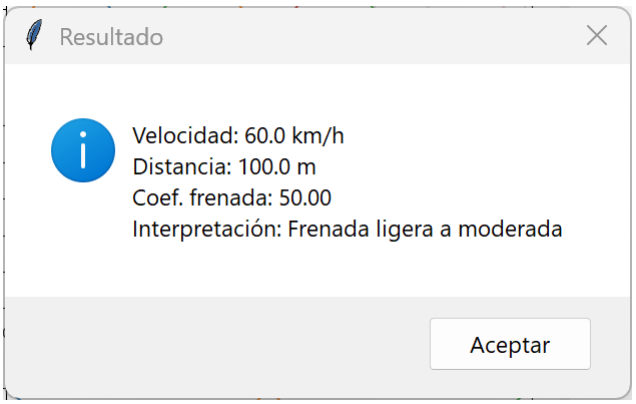
6.4.1.- Velocidad 40 km/h y 250 m distancia

Con una velocidad de 40 km/h y una distancia de 250 m, el sistema determina un coeficiente de frenada de 14.44, lo que corresponde a una "**Frenada nula o muy ligera**".

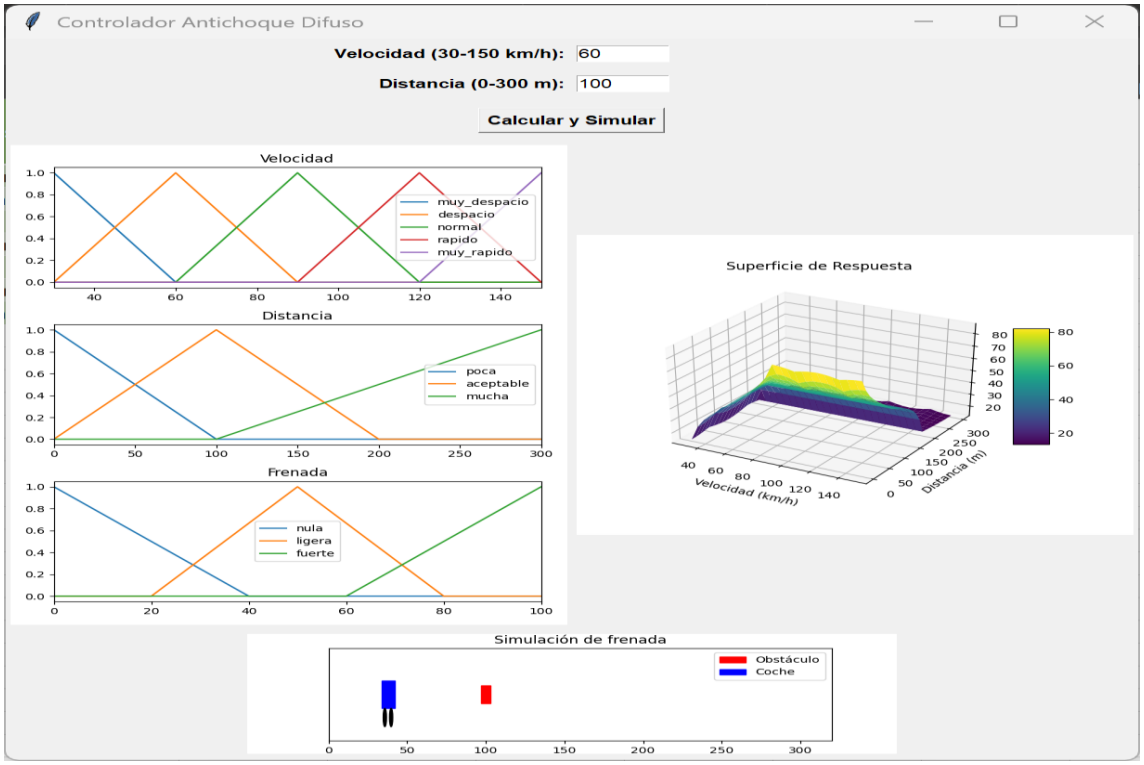


Esta interfaz muestra la configuración del sistema antichoque difuso y la simulación resultante para una velocidad de 40 km/h y una distancia al obstáculo de 250 m.

6.4.2.- Velocidad 60 km/h y 100 m distancia

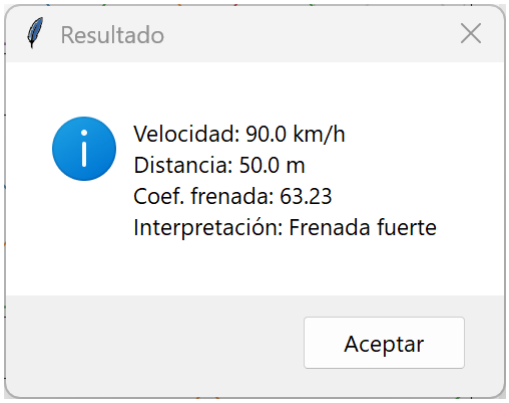


A 60 km/h y con 100 m de distancia, el sistema calcula un coeficiente de frenada de 50.00, interpretado como una "**Frenada ligera a moderada**".

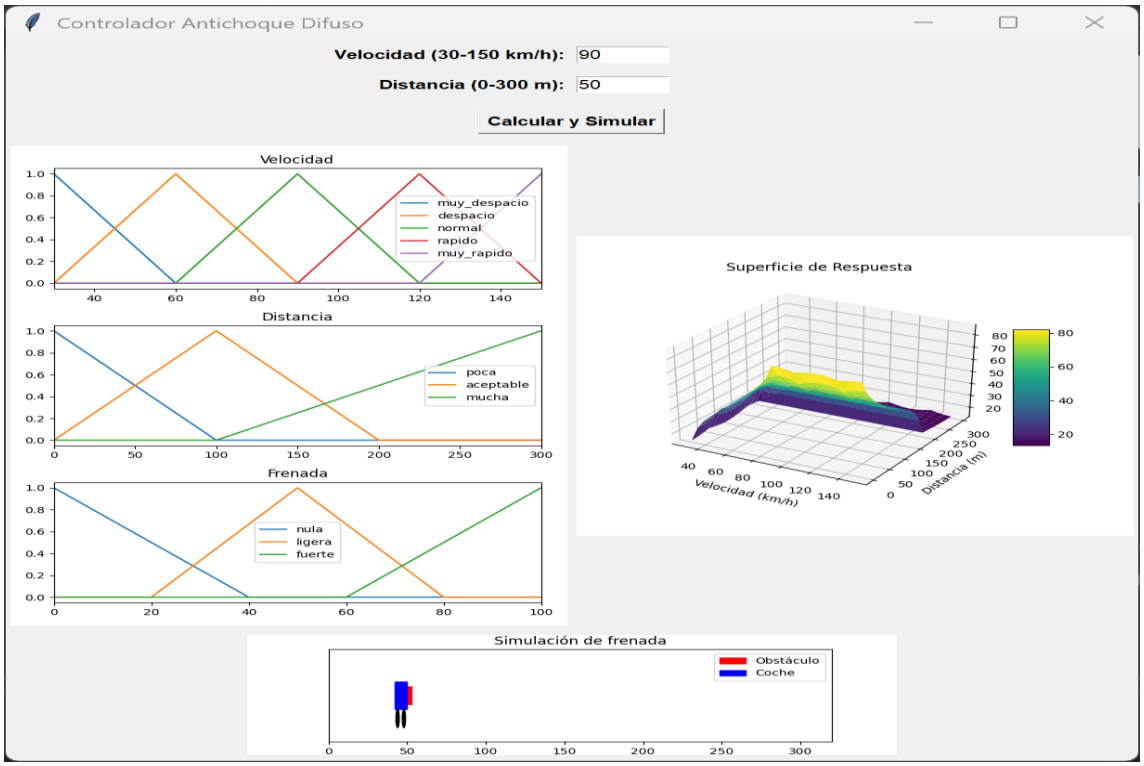


Esta interfaz ilustra el sistema antichoque difuso configurado y simulado para una velocidad de 60 km/h y una distancia de 100 m.

6.4.3.- Velocidad 90 km/h y 50 m distancia



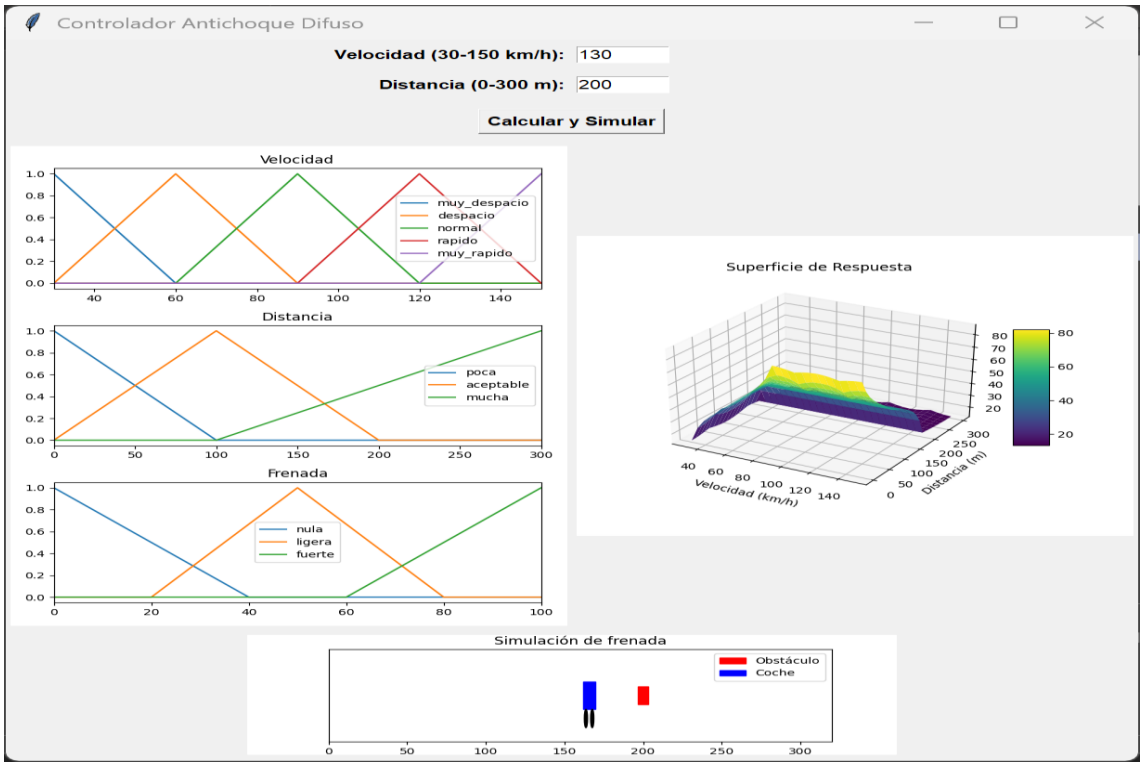
A 90 km/h y 50 m de distancia, el sistema calcula un coeficiente de frenada de 63.23, lo que se interpreta como una "**Frenada fuerte**".



Esta interfaz muestra la configuración y la simulación del sistema antichoque difuso para las entradas actuales de 90 km/h de velocidad y 50 m de distancia.

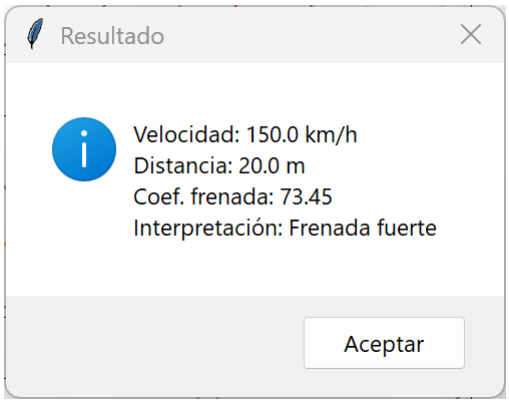
6.4.4.- Velocidad 130 km/h y 200 m distancia

A 130 km/h y con 200 m de distancia, el sistema calcula un coeficiente de frenada de 15.56, lo que resulta en una "**Frenada nula o muy ligera**".

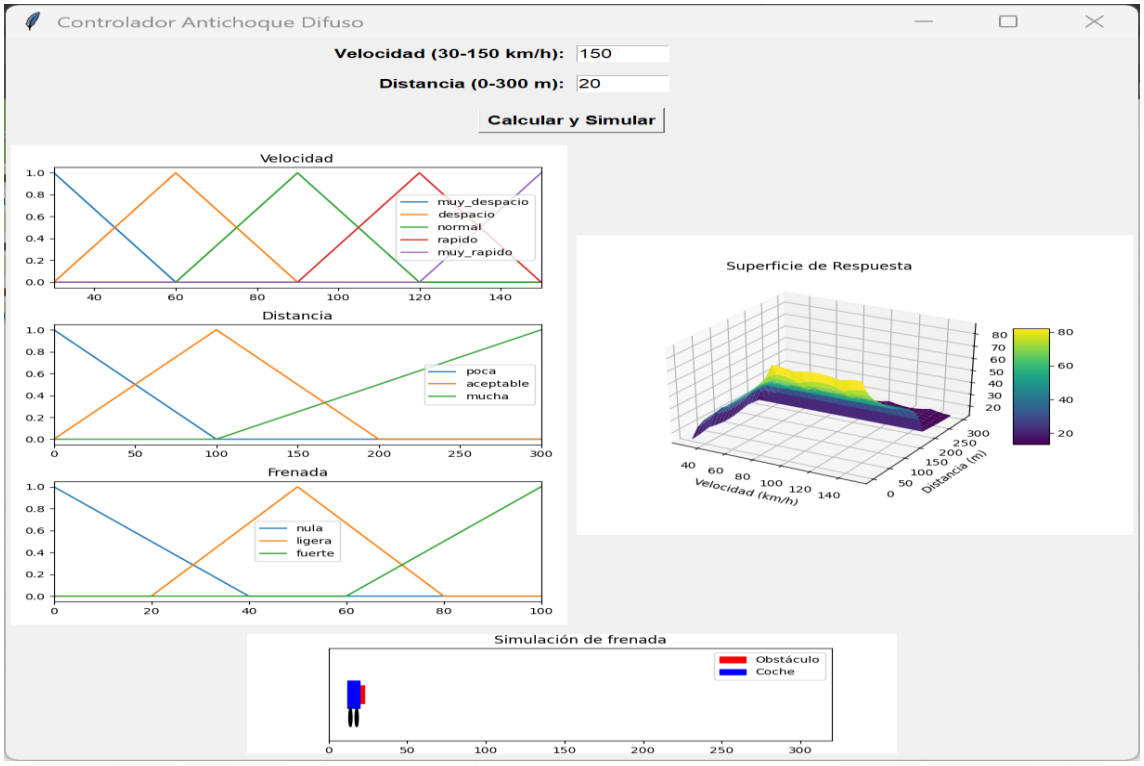


Esta interfaz muestra la configuración del sistema antichoque difuso y la simulación resultante para una velocidad de 130 km/h y una distancia de 200 m.

6.4.5.- Velocidad 150 km/h y 20 m distancia



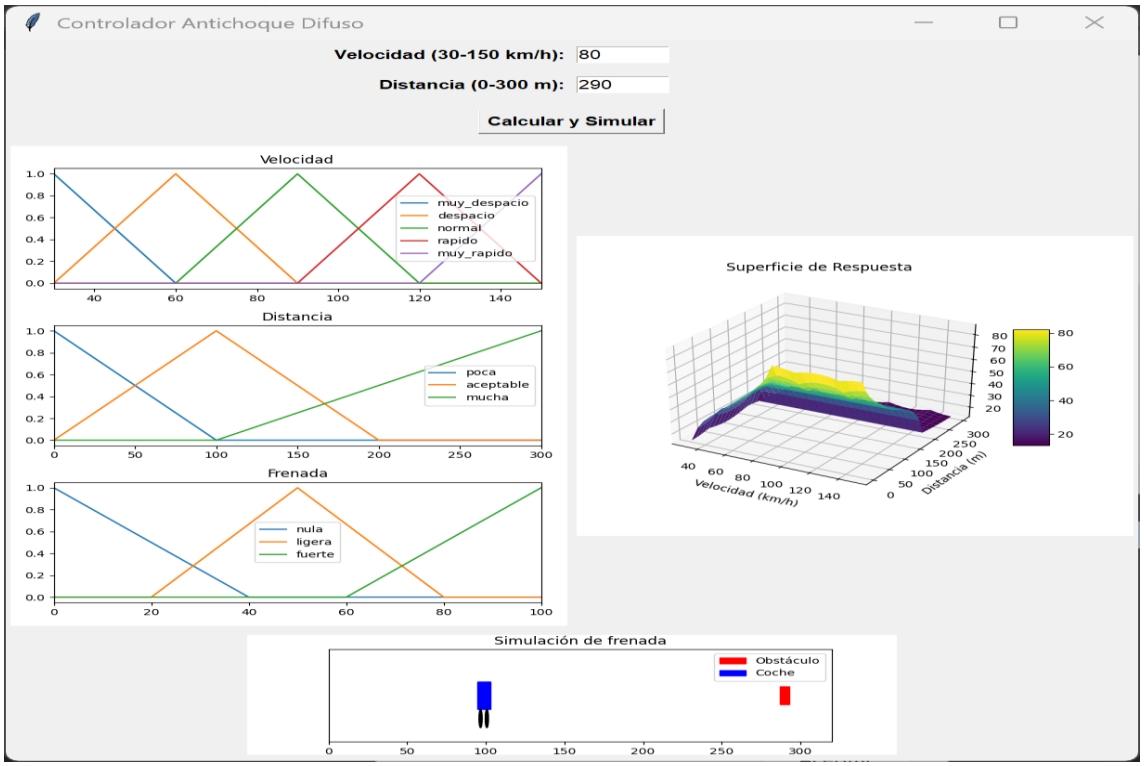
A 150 km/h y con tan solo 20 m de distancia, el sistema calcula un coeficiente de frenada de 73.45, lo que se interpreta como una "**Frenada fuerte**".



Esta interfaz muestra la configuración del sistema antichoque difuso y la simulación correspondiente a una velocidad de 150 km/h y una distancia de 20 m.

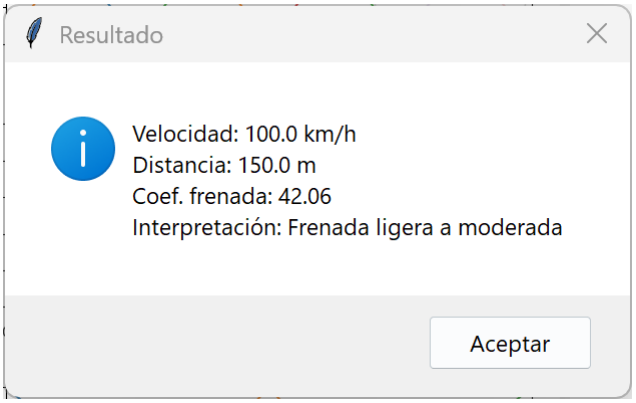
6.4.6.- Velocidad 80 km/h y 290 m distancia

A 80 km/h y con una distancia de 290 m, el sistema calcula un coeficiente de frenada de 14.44, interpretado como una "**Frenada nula o muy ligera**".

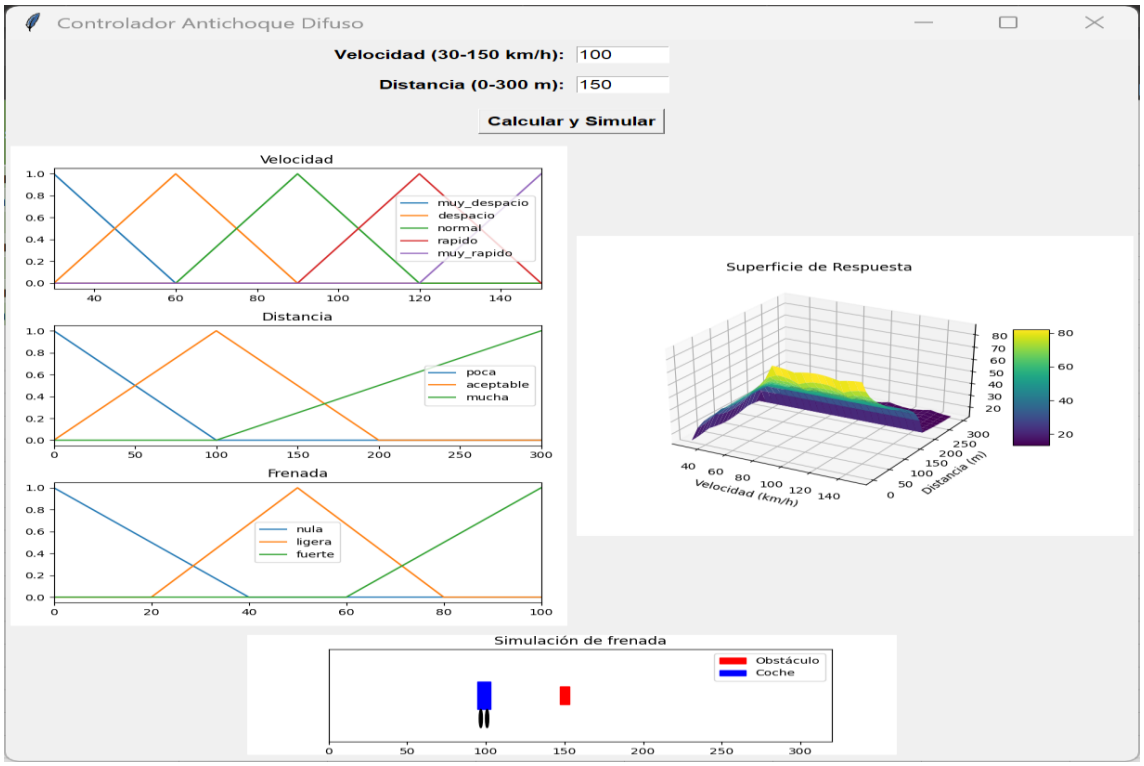


Esta interfaz muestra la configuración del sistema antichoque difuso y la simulación para una velocidad de 80 km/h y una distancia de 290 m.

6.4.7.- Velocidad 100 km/h y 150 m distancia



A 100 km/h y con una distancia de 150 m, el sistema calcula un coeficiente de frenada de 42.06, interpretado como una "**Frenada ligera a moderada**".



Esta interfaz visualiza la configuración y simulación del sistema antichoque difuso con una velocidad de 100 km/h y una distancia de 150 m.

7.- Conclusiones

El sistema de control difuso implementado ha demostrado ser eficaz para simular un controlador antichoque que toma decisiones en función de la velocidad del vehículo y la distancia al obstáculo. A través de una lógica basada en reglas difusas, se logra una respuesta suave, coherente y adaptativa ante diversas condiciones de entrada.

7.1.- Evaluación del sistema

El sistema ha sido evaluado tanto desde el punto de vista **lógico** como **visual**. Entre los aspectos más destacables se encuentran:

- **Correcta interpretación de las reglas difusas**: el sistema reacciona de forma razonable y predecible en todos los escenarios de prueba.
- **Transición fluida entre diferentes grados de frenado**, evitando respuestas bruscas.
- **Interfaz clara e intuitiva**, lo que permite al usuario interactuar fácilmente con el sistema e interpretar los resultados.
- **Simulación visual útil** para comprender el comportamiento del coche ante diferentes condiciones de velocidad y distancia.

En general, el sistema cumple su objetivo didáctico y demuestra el poder de la lógica difusa para modelar sistemas complejos de decisión.

7.2.- Posibles mejoras

Aunque el sistema funciona adecuadamente, existen varias **áreas de mejora** que podrían aumentar su precisión, realismo y utilidad:

- **Optimización de las reglas**: actualmente, las reglas son definidas manualmente. Se podría aplicar aprendizaje automático para ajustarlas a datos reales de conducción.
- **Considerar más variables**: por ejemplo, condiciones del clima, tipo de carretera, estado del vehículo, entre otras.
- **Mejorar la animación**: incluir rotación de ruedas, efectos de frenado (como luces de freno), sonido, o un entorno más realista.

- **Guardar resultados**: permitir exportar los datos de simulación para su análisis posterior.
- **Versión web o multiplataforma**: crear una versión del sistema en formato web o móvil para facilitar su acceso y demostración.

7.3.- Aplicaciones futuras

Este tipo de sistemas pueden tener diversas aplicaciones reales y educativas:

- **Educación**: herramienta didáctica para enseñar lógica difusa y sistemas de control inteligente.
- **Simuladores de conducción**: para entrenar conductores en diferentes condiciones de frenado.
- **Sistemas de asistencia a la conducción (ADAS)**: el modelo puede ser base para un sistema real de frenado automático o anticolisión.
- **Prototipado rápido**: útil para validar modelos iniciales de comportamiento vehicular en entornos controlados.
- **Integración en vehículos autónomos**: como parte de módulos de toma de decisiones dentro de sistemas de navegación y seguridad.

8.- Bibliografía

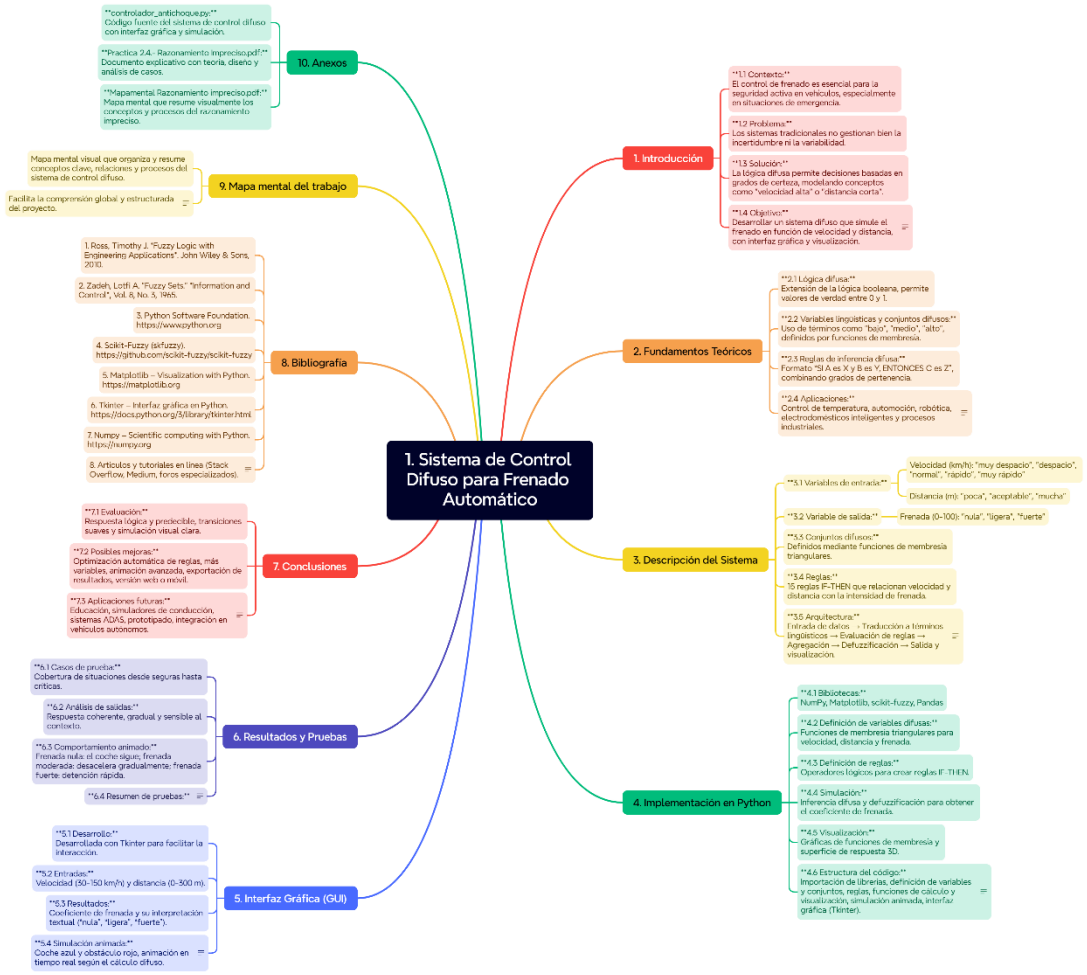
A continuación, se enumeran las fuentes consultadas y utilizadas para el desarrollo del sistema de control difuso y su interfaz gráfica en Python:

1. **Ross, Timothy J.**
   FUZZY LOGIC WITH ENGINEERING APPLICATIONS.
   John Wiley & Sons, 2010.
   (REFERENCIA TEÓRICA PRINCIPAL PARA LA COMPRENSIÓN DE SISTEMAS DE LÓGICA DIFUSA APLICADOS A LA INGENIERÍA).
2. **Zadeh, Lotfi A.**
   "Fuzzy Sets."
   INFORMATION AND CONTROL, Vol. 8, No. 3, 1965, pp. 338–353.
   (FUNDAMENTO TEÓRICO DE LA LÓGICA DIFUSA, PROPUESTA POR SU CREADOR).
3. **Python Software Foundation**
   https://www.python.org

(LENGUAJE DE PROGRAMACIÓN UTILIZADO PARA EL DESARROLLO DEL SISTEMA).

4. **Scikit-Fuzzy (skfuzzy)**
   https://github.com/scikit-fuzzy/scikit-fuzzy
   (LIBRERÍA EMPLEADA PARA LA CREACIÓN DEL SISTEMA DE CONTROL DIFUSO).
5. **Matplotlib – Visualization with Python**
   https://matplotlib.org
   (LIBRERÍA PARA GRAFICAR Y ANIMAR LOS RESULTADOS DEL SISTEMA DE CONTROL).
6. **Tkinter – Interfaz gráfica en Python**
   Documentación oficial de Tkinter:
   https://docs.python.org/3/library/tkinter.html
   (LIBRERÍA ESTÁNDAR UTILIZADA PARA EL DESARROLLO DE LA INTERFAZ GRÁFICA).
7. **Numpy – Fundamental package for scientific computing with Python**
   https://numpy.org
   (LIBRERÍA PARA EL MANEJO EFICIENTE DE ARREGLOS Y OPERACIONES NUMÉRICAS).
8. **Artículos y tutoriales en línea**
   Diversas fuentes en línea fueron consultadas para resolver dudas específicas de implementación, tales como Stack Overflow, Medium, y foros especializados en Python y sistemas de control difuso.

9.- Mapa mental del trabajo



Este mapa mental resume visualmente la estructura completa de un trabajo sobre un sistema de control difuso para frenado automático, abarcando su teoría, descripción, implementación en Python, interfaz gráfica, pruebas, conclusiones y documentación anexa.

10.- Anexos

En esta sección se presentan los ficheros entregados como parte de la práctica, los cuales respaldan el desarrollo completo del sistema de control difuso para frenado automático ante obstáculos.

El fichero **controlador_antichoque.py** implementa un sistema de control difuso con interfaz gráfica en el que se definen las variables

lingüísticas, las reglas de inferencia, funciones para calcular y visualizar la frenada y la simulación del coche, permitiendo al usuario interactuar y comprender de manera visual y numérica el funcionamiento del sistema.

El documento **Práctica 2-4 Lógica Difusa.pdf** complementa el código explicando la teoría y justificación del sistema de control difuso, detallando su objetivo, funcionamiento y diseño, e incluyendo ejemplos, capturas y análisis de casos para facilitar su comprensión tanto técnica como conceptual.

El archivo **Mapamental Práctica 2-4 Lógica Difusa.pdf** presenta un mapa mental que organiza y resume visualmente los conceptos clave, relaciones y procesos de la Lógica Difusa, facilitando la comprensión global y estructurada del tema mediante diagramas y conexiones gráficas.

Índice Alfabético

E

F

G

H

I

L

M

v

w