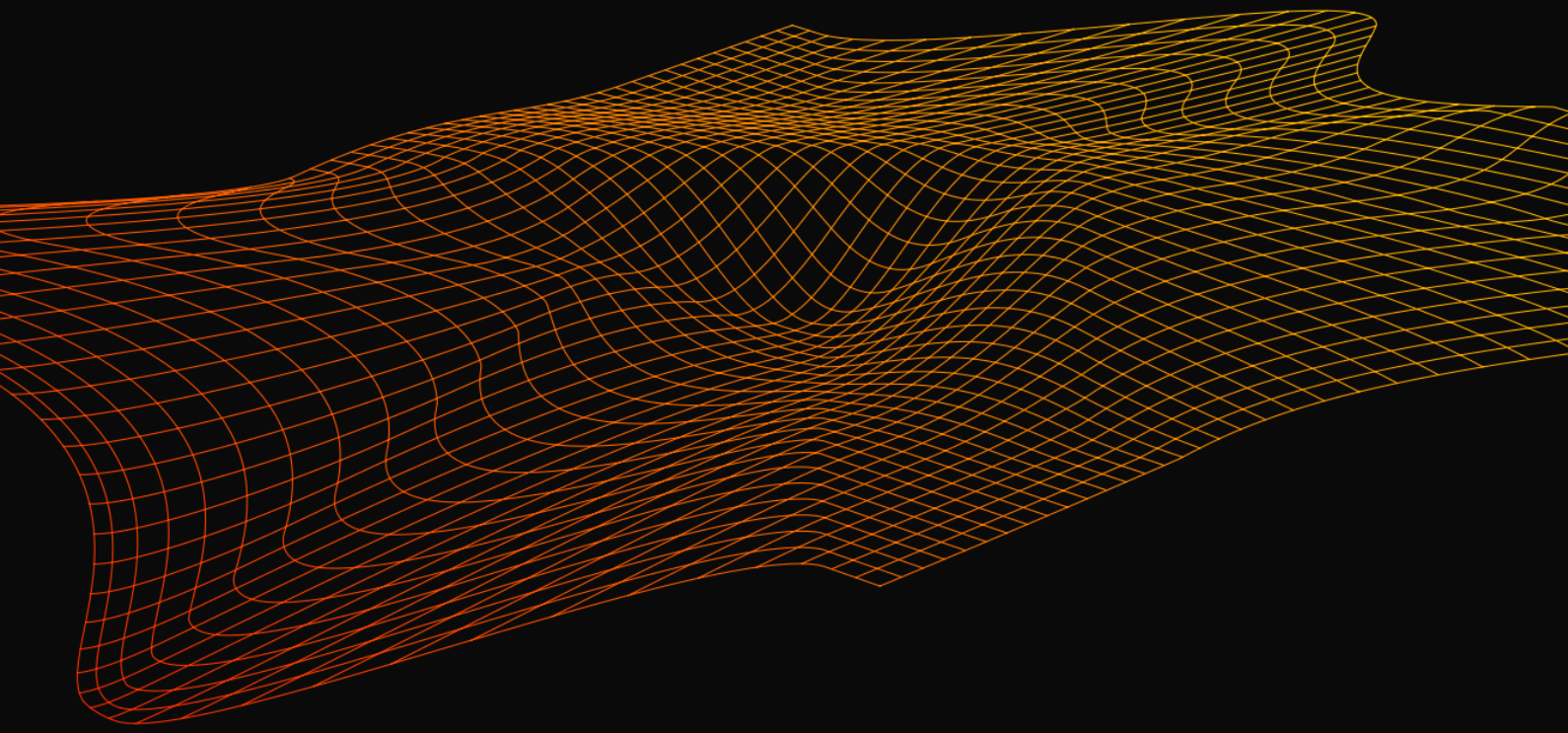
IMPLEMENTACION DE FUZZY LOGIC EN UNA RASSPERRY PI4

DETECCION DE MOVIMIENTO USANDO VISIÓN COMPUTACIONAL Y LÓGICA DIFUSA



LENGUAJES INTELIGENTES

INGENIERIA EN COMPUTACION INTELIGENTE

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

- MARTIN ISAI NUÑEZ VILLEDA
- PAOLA MONSERRAT OSORIO GARCIA
- JOVANY FLORES ROBLEDO
- ALEXIS ALBERTO ZUÑIGA ALONSO
- ALAN FERNANDO MARTINEZ MORENO

Prefacio

La lógica difusa es una herramienta que permite representar y razonar con información imprecisa, lo cual resulta útil en aplicaciones donde las decisiones no son estrictamente binarias. A diferencia de los sistemas tradicionales basados en valores exactos, los sistemas difusos modelan grados de pertenencia y permiten transiciones suaves entre diferentes estados.

En este proyecto se implementa un sistema experto difuso en Raspberry Pi, cuyo propósito es detectar el nivel de movimiento captado por una cámara y controlar la iluminación de LEDs en función de dicho movimiento. El sistema combina técnicas de visión artificial (OpenCV) con control de hardware (GPIO de la Raspberry Pi) y un motor de inferencia difuso desarrollado con la librería scikit-fuzzy en Python.

Se utiliza la cámara conectada a la Raspberry Pi para obtener imágenes en tiempo real.
El sistema compara frames consecutivos en escala de grises y calcula la cantidad de píxeles que han cambiado, obteniendo así un valor numérico que representa el nivel de movimiento.

CONFIGURACIÓN PINES

```
led_pins = [17, 27, 22]  # Pines GPIO para los LEDs

GPIO.setmode(GPIO.BCM)

for pin in led_pins:

  GPIO.setup(pin, GPIO.OUT)
```

se crea un arreglo con el número asignado al puerto de propósito general de la raspberry (GPIO) y se declaran esos pines como salida en un ciclo for

Base de conocimientos

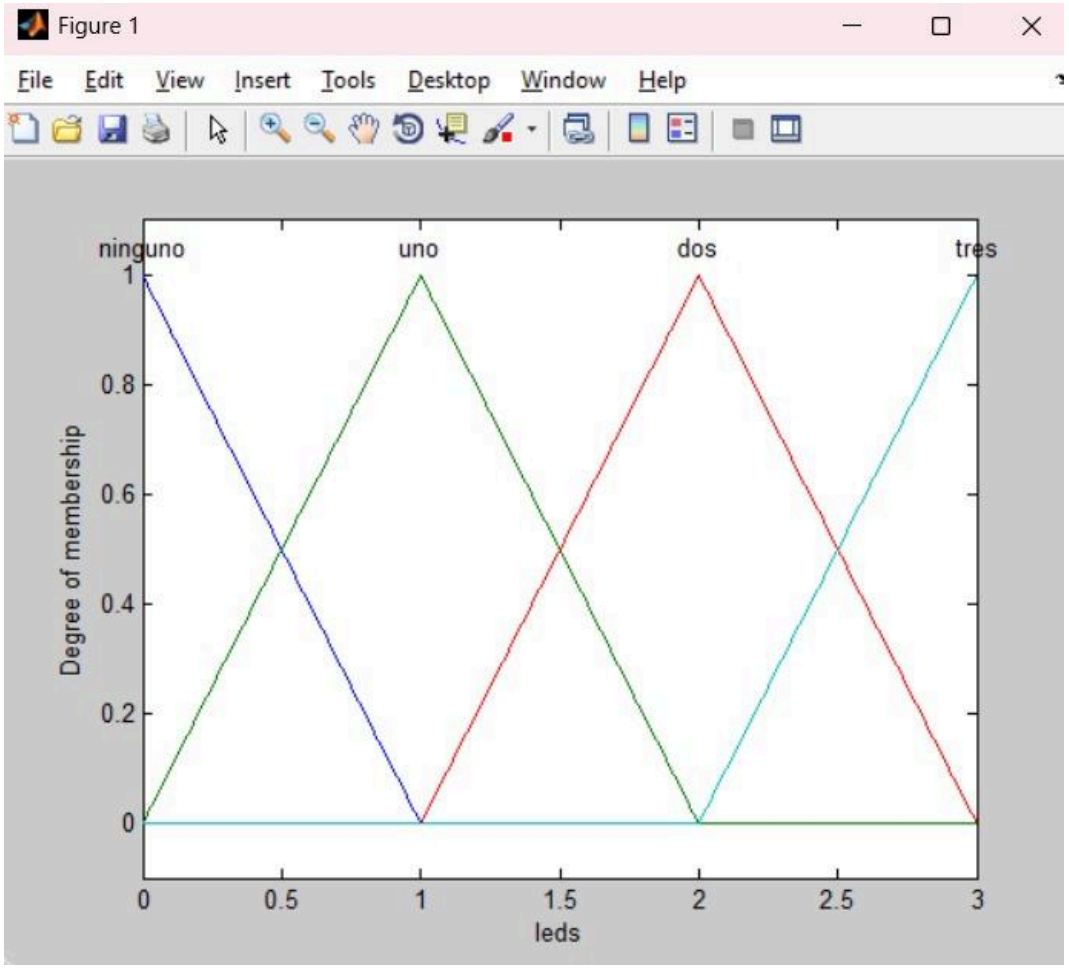
Para nuestra base de conocimientos se declaran dos variables difusas una para poder crear un antecedente y una consecuente además de sus funciones de membresía (declaración de la función), que después no ayudará a crear reglas en función del antecedente que experimentemos.

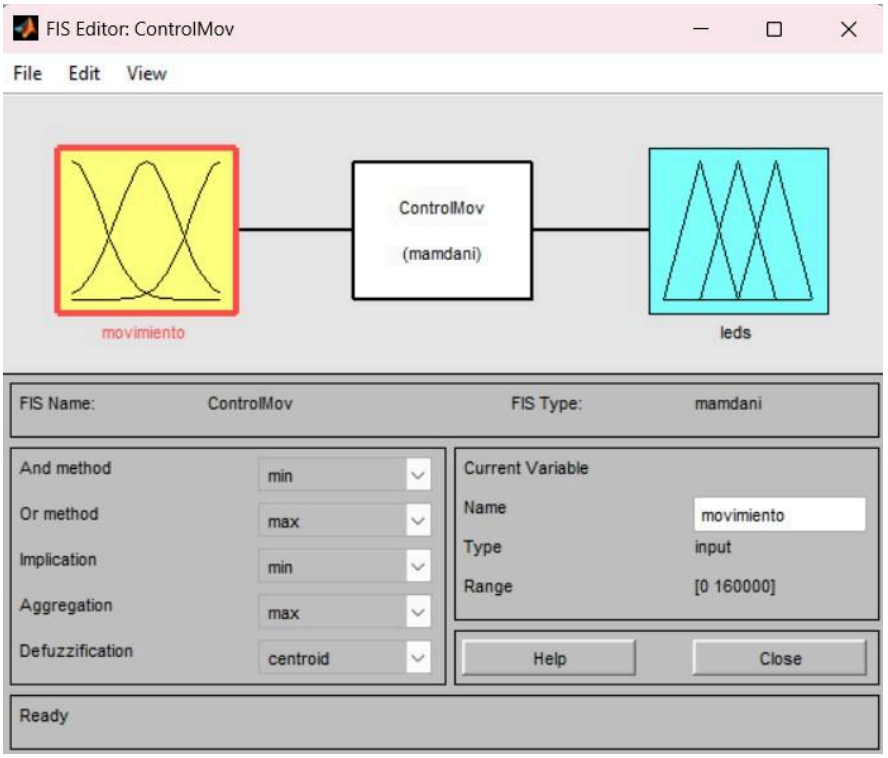
Dada la resolución de la cámara se pudo obtener máximos de 160000 pixeles de cambio así que en promedio ese es el tope de las funciones de membresía aunque matemáticamente debería ser más.

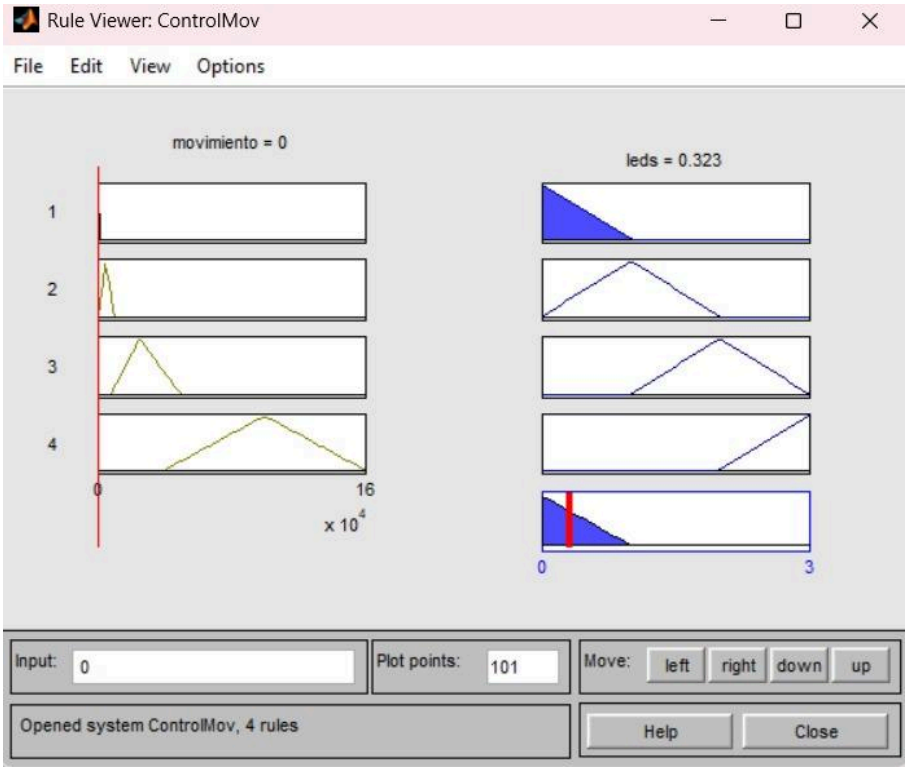
Y dada la naturalidad del proyecto se declara una variable difusa para los leds de la misma forma.

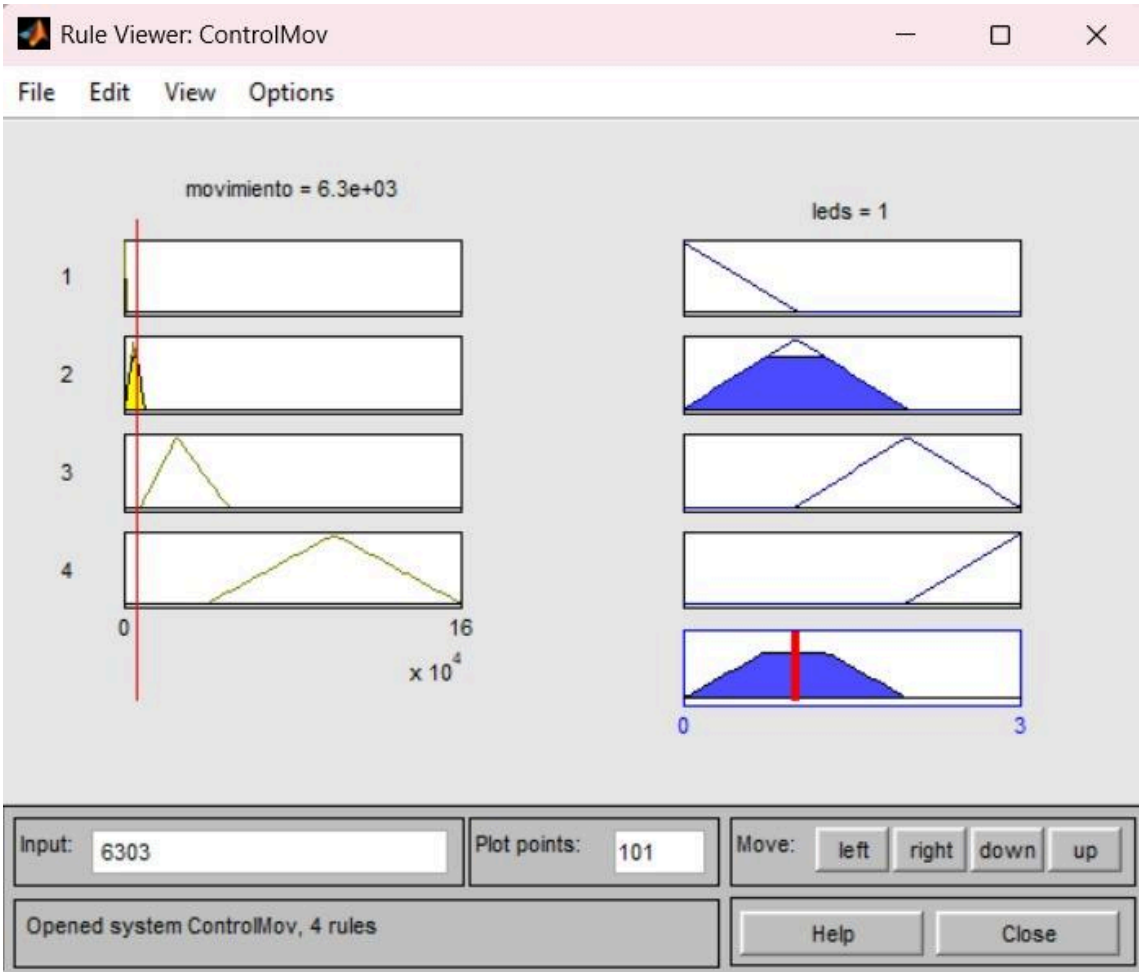
Las funciones de membresía son del tipo triangular.

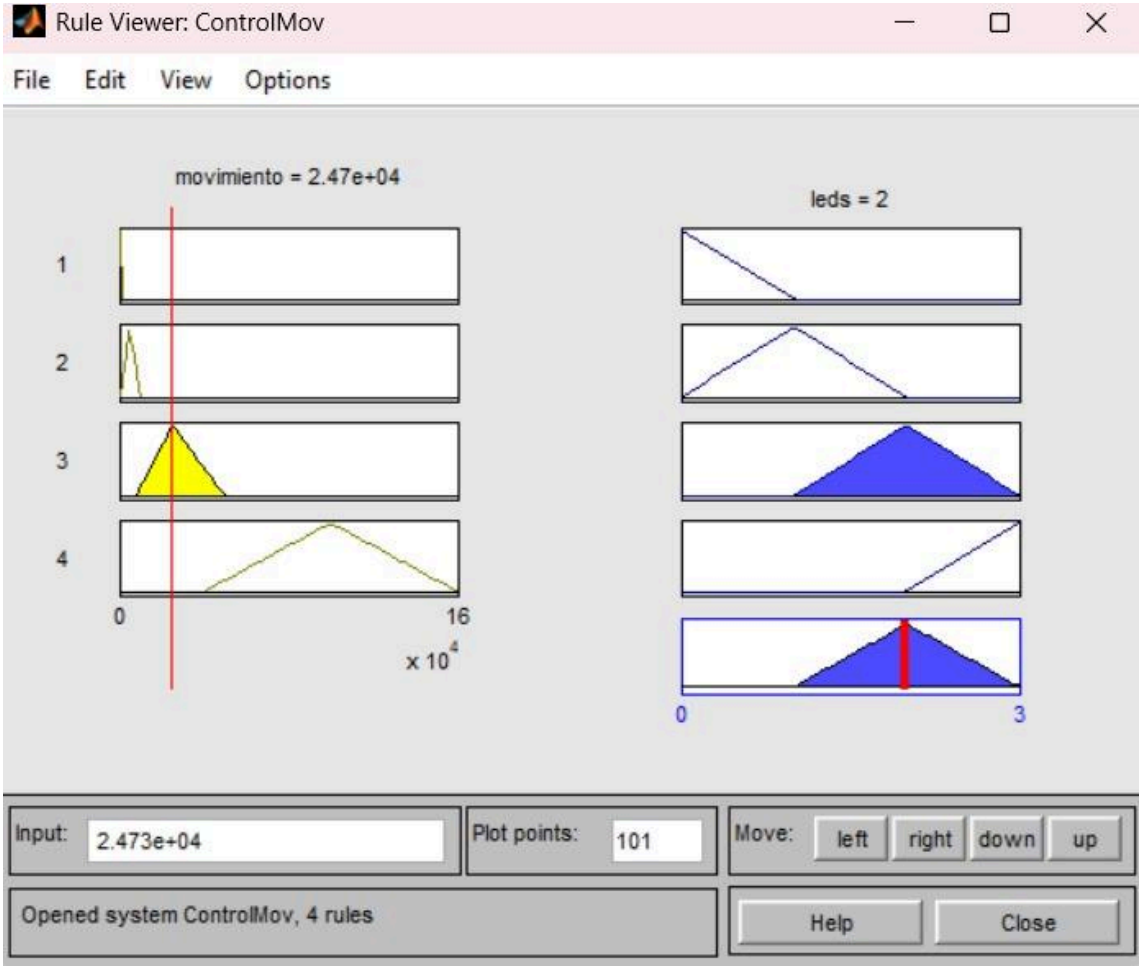
.

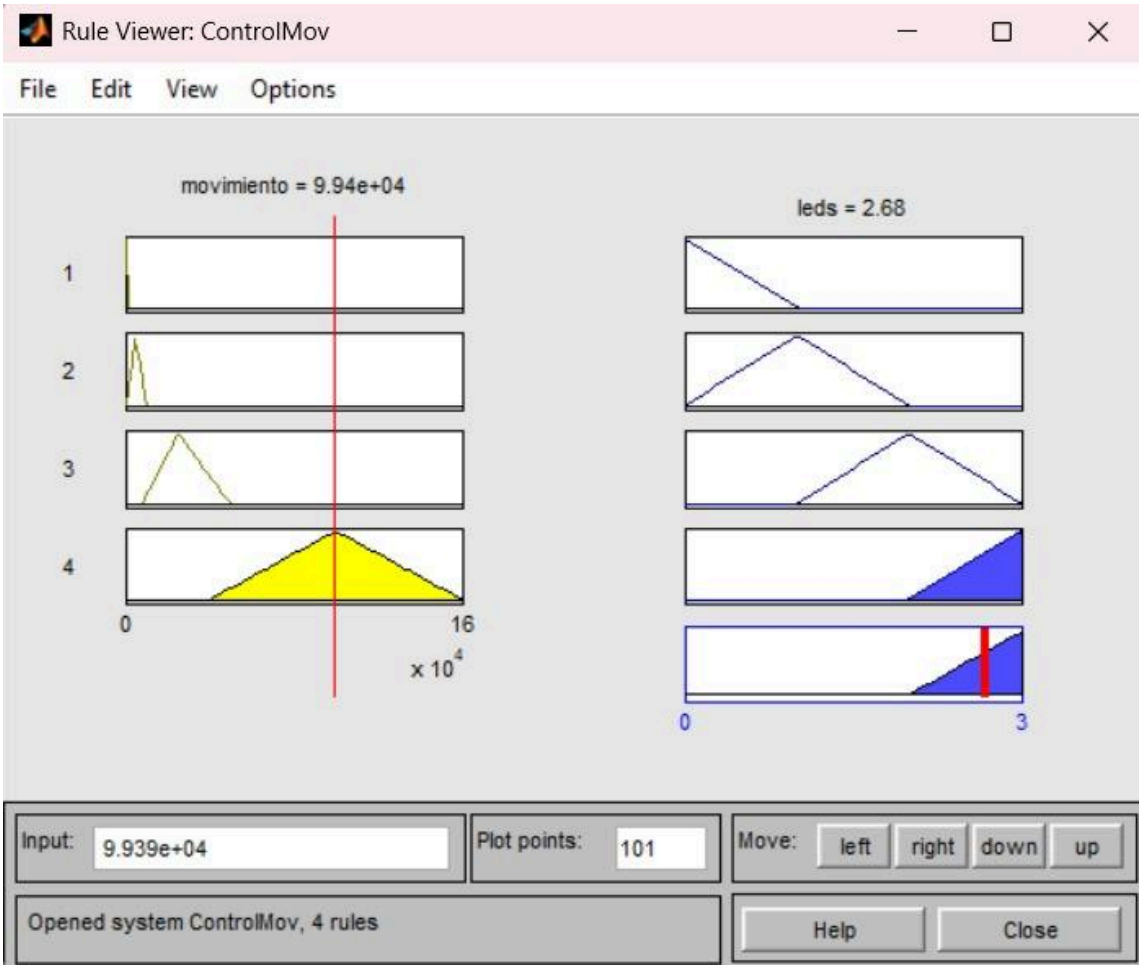



FIS Editor: ControlMov
File Edit View
ControlMov
(mamdani)
movimiento
leds
FIS Name: ControlMov
FIS Type: mamdani
And method min
Or method max
Implication min
Aggregation max
Defuzzification centroid
Current Variable
Name movimiento
Type input
Range [0 160000]
Help
Close
Ready


Rule Viewer: ControlMov
File Edit View Options
movimiento = 0
leds = 0.323
1
2
3
4
0
16
x 10^4
0
3
Input: 0
Plot points: 101
Move: left right down up
Opened system ControlMov, 4 rules
Help
Close


Rule Viewer: ControlMov
File Edit View Options
movimiento = 6.3e+03
leds = 1
1
2
3
4
0
16
x 10^4
0
3
Input: 6303
Plot points: 101
Move: left right down up
Opened system ControlMov, 4 rules
Help Close


Rule Viewer: ControlMov
File Edit View Options
movimiento = 2.47e+04
leds = 2
1
2
3
4
0
16
x 10^4
0
3
Input: 2.473e+04
Plot points: 101
Move: left right down up
Opened system ControlMov, 4 rules
Help Close



Variables difusas

```
# Universo de movimiento
velocidad = ctrl.Antecedent(np.arange(0, 409601, 1), 'movimiento')
# Universo de salida: 0 a 3 LEDs
leds = ctrl.Consequent(np.arange(0, 4, 1), 'leds')
```

Funciones de membresía

```
# Funciones de membresía para velocidad (entrada)
velocidad['nula'] = fuzz.trimf(velocidad.universe, [0, 0, 1000])
velocidad['bajo'] = fuzz.trimf(velocidad.universe, [500, 5000, 10000])
velocidad['media'] = fuzz.trimf(velocidad.universe, [8000, 25000, 50000])
velocidad['alto'] = fuzz.trimf(velocidad.universe, [40000, 100000, 160000])
```

```
# Funciones de membresía para LEDs (salida)
leds['ninguno'] = fuzz.trimf(leds.universe, [0, 0, 1])
leds['uno']     = fuzz.trimf(leds.universe, [0, 1, 2])
leds['dos']     = fuzz.trimf(leds.universe, [1, 2, 3])
leds['tres']    = fuzz.trimf(leds.universe, [2, 3, 3])
```

Máquina de inferencia

La manera de inferir las salidas para el motor de inferencia es intuitiva en función de qué estado se encuentre será el número de leds que se encienda.

Una vez establecidas las reglas suficientes y necesarias se crea el sistema difuso

```
regla1 = ctrl.Rule(velocidad['nula'], leds['ninguno'])

regla2 = ctrl.Rule(velocidad['bajo'], leds['uno'])

regla3 = ctrl.Rule(velocidad['media'], leds['dos'])

regla4 = ctrl.Rule(velocidad['alto'], leds['tres'])


# Sistema difuso

control_mov = ctrl.ControlSystem([regla1, regla2, regla3, regla4])

mov_sim = ctrl.ControlSystemSimulation(control_mov)
```

Evaluación del Sistema Experto

Interfaz de fuzzificación

Al estar trabajando con una cámara y vídeo se sabe que por cada imagen se está trabajando con una matriz de píxeles y dada la naturaleza de la misma el video es un conjunto de imágenes que a su vez son matrices.

Para poder obtener la entrada cruda para el sistema difuso se tuvo que preprocesar las imágenes a un valor entero de entrada.

¿ Cómo funciona ?

Se inicializa y solicita la cámara de no detectarse se enviará un código de error, dentro de dos variables se guardaran 2 matrices de píxeles(imágenes) capturadas en el momento se transformaran con ayuda de una función de la librería Opencv a una escala de grises, después con otra función de la misma librería se captura la diferencia absoluta entre píxeles de la misma (se binariza la diferencia) es decir que solo nos digas si cambio o no no cuanto cambio.

Finalmente el resultado se adhiere a un acumulador que será el número de píxeles que cambió entre frames o imágenes.

Ya obtenido ese valor crudo se ingresa al sistema con lógica difusa.

Donde en función del número de píxeles que cambian entre frames será el valor que inferir , por ejemplo:

Si entre frames cambio menos de 1000 pixeles entraría en movimiento nulo por lo que no se encenderá ningún led.

En función de la entrada que sea y su valor la salida del valor es decir el defuzzing es el número de leds que se encenderá,Finalmente este proceso estará en bucle indefinido, se detendrá hasta que se detecte la letra “q” o “esc”

```
try:
    while True:
        ret, frame = cap.read()
        if not ret:
            print("Error: No se pudo capturar el frame")
            break

        # Convertir a escala de grises
        antgray = cv2.cvtColor(ant, cv2.COLOR_BGR2GRAY)
        actgray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Diferencia entre frames
        diferencia = cv2.absdiff(antgray, actgray)
        _, thresh = cv2.threshold(diferencia, 30, 255, cv2.THRESH_BINARY)

        # Contar píxeles cambiados (movimiento detectado)
        mov_valor = np.sum(thresh > 0)
        mov_valor = min(mov_valor, 409600)

        # Evaluar con fuzzy logic
        mov_sim.input['movimiento'] = mov_valor
        mov_sim.compute()
        salida_leds = round(mov_sim.output['leds'])

        print(f"Movimiento: {mov_valor} -> Encender {salida_leds} LED(s)")

        # Mostrar cámara
        cv2.imshow("Camera Feed", frame)

        # Encender LEDs según salida
        for i, pin in enumerate(led_pins):
            GPIO.output(pin, GPIO.HIGH if i < salida_leds else GPIO.LOW)
```

```
        # Guardar frame actual como anterior
        ant = frame.copy()

        # Salir con 'q' o ESC
        if cv2.waitKey(1) & 0xFF in (ord('q'), 27):
            break

except KeyboardInterrupt:
    print("Interrumpido por el usuario")

finally:
    # Liberar recursos
    cap.release()
    cv2.destroyAllWindows()
    GPIO.cleanup()
```

CONCLUSIONES

La implementación de un sistema experto difuso para la detección de movimiento y control de LEDs en una Raspberry Pi permitió demostrar de manera práctica los principios fundamentales de la lógica difusa. A diferencia de un sistema clásico, que tomaría decisiones binarias ante la presencia o ausencia de movimiento, la lógica difusa posibilitó un rango de respuestas más realistas, graduadas según la magnitud del cambio detectado en las imágenes.

El uso de funciones de membresía triangulares permitió modelar adecuadamente la incertidumbre asociada al movimiento capturado por la cámara. Esto facilitó que el sistema pudiera interpretar pequeños desplazamientos como "bajo movimiento" y grandes variaciones como "alto movimiento", en lugar de limitarse a un simple umbral rígido. Asimismo, el motor de inferencia Mamdani resultó adecuado por su sencillez y claridad en la aplicación de reglas del tipo SI… ENTONCES…, las cuales constituyen la base de conocimiento del sistema.

En conclusión, este proyecto no solo cumple con los objetivos académicos planteados, sino que también sienta las bases para desarrollos futuros más complejos, como sistemas de seguridad inteligentes, domótica o aplicaciones IoT. La lógica difusa se presenta así como una herramienta poderosa para diseñar sistemas que requieren adaptabilidad y capacidad de decisión en condiciones de incertidumbre.

REFERENCIAS

OpenCV Team. (2025). *OpenCV Documentation*. OpenCV. https://docs.opencv.org/

Python Software Foundation. (2025). *Python Documentation*. Python.org. https://docs.python.org/3/

Raspberry Pi Foundation. (2025). *GPIO Python Documentation*. Raspberry Pi. https://www.raspberrypi.com/documentation/

Ross, T. J. (2010). *Fuzzy Logic with Engineering Applications* (3rd ed.). Wiley.

Scikit-Fuzzy Developers. (2025). *scikit-fuzzy: Fuzzy Logic Toolbox for Python*. https://pythonhosted.org/scikit-fuzzy/

GeeksforGeeks. (2023, August 2). *Introduction to Fuzzy Logic*. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-fuzzy-logic/