MEMORIA: SIGUE PAREDES

Autores: Rodrigo Durán Andrés y Rubén Cid Costa

Contenido

1. Introducción

Este proyecto consiste en usando el simulador Coppelia, conseguir que un robot (en este caso el robot PioneerP3DX), siga las paredes.

El escenario que usaremos en Coppelia consiste en 4 paredes que crean una habitación cerrada, con dos paredes perpendiculares y conectadas a las paredes, a forma de saliente.

El robot tiene 16 sensores de distancia (ultrasonidos), 2 mirando para cada eje, o sea, 2 mirando para adelante, 2 para atrás, 2 para la izquierda y 2 para la derecha. Además, tiene 2 sensores entre cada eje que se inclinan progresivamente.

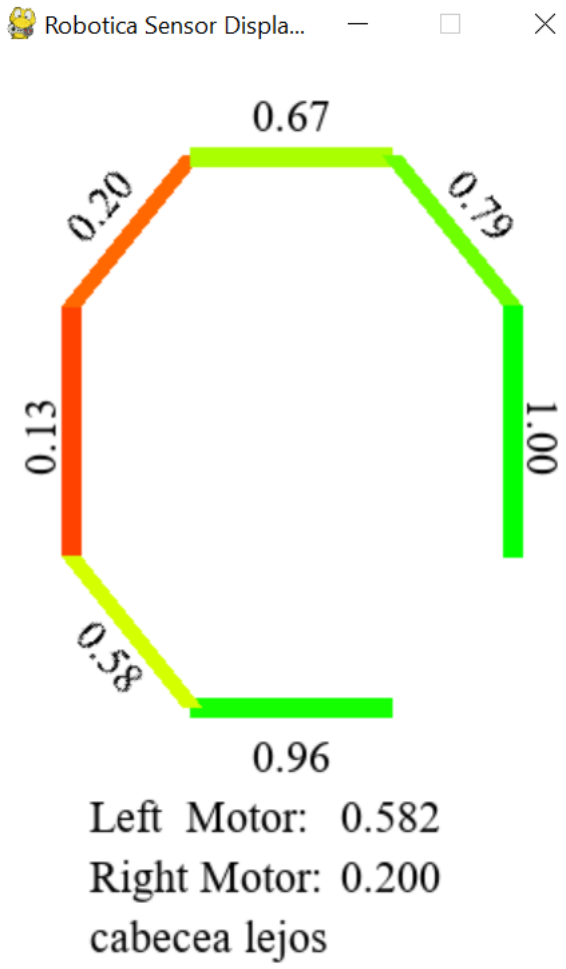
Los experimentos son una agrupación de nuevas reglas o cambios de código, que cada uno fue probándose individualmente (para ajustar sus parámetros), pero que están agrupados para una representación mas clara del progreso del código.

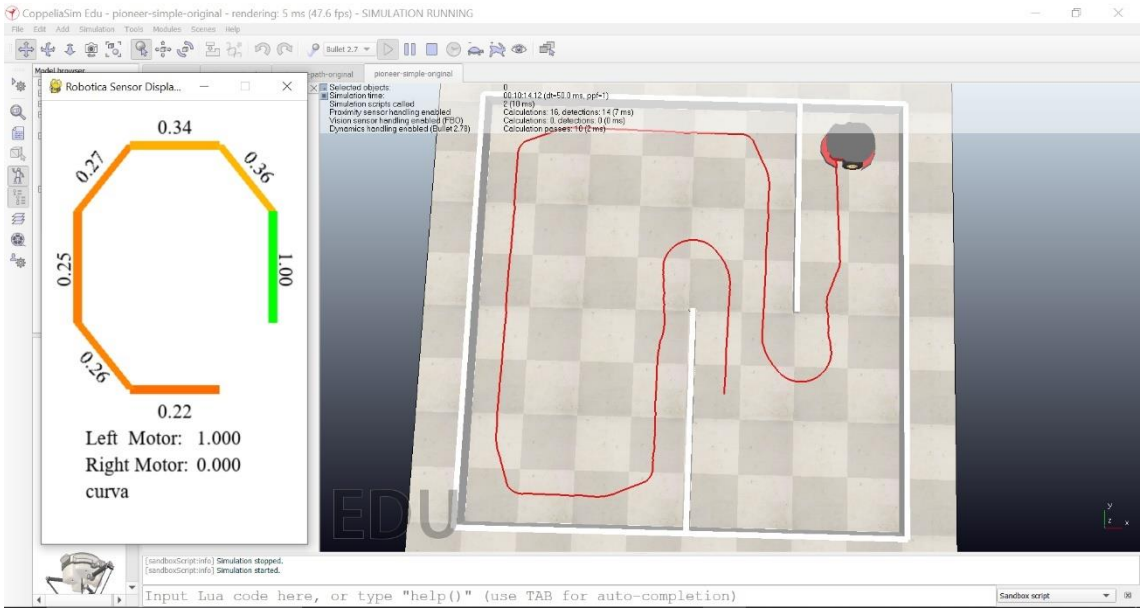
Enlace al repositorio de github

2. Proceso

2.1. Visualización de Sensores

Empezamos creando una interfaz con pygame, que nos muestra la media del input de dos sensores (no mostramos atrás izquierda, ya que no se cree que sea necesario), el output de las velocidades y un string de estado. Ej:





Usamos colores para indicar la cercanía de dicha sección a la pared. Esto fue muy útil para ver fácilmente en que parte del bucle estaba y que valores detectaban en tiempo real.

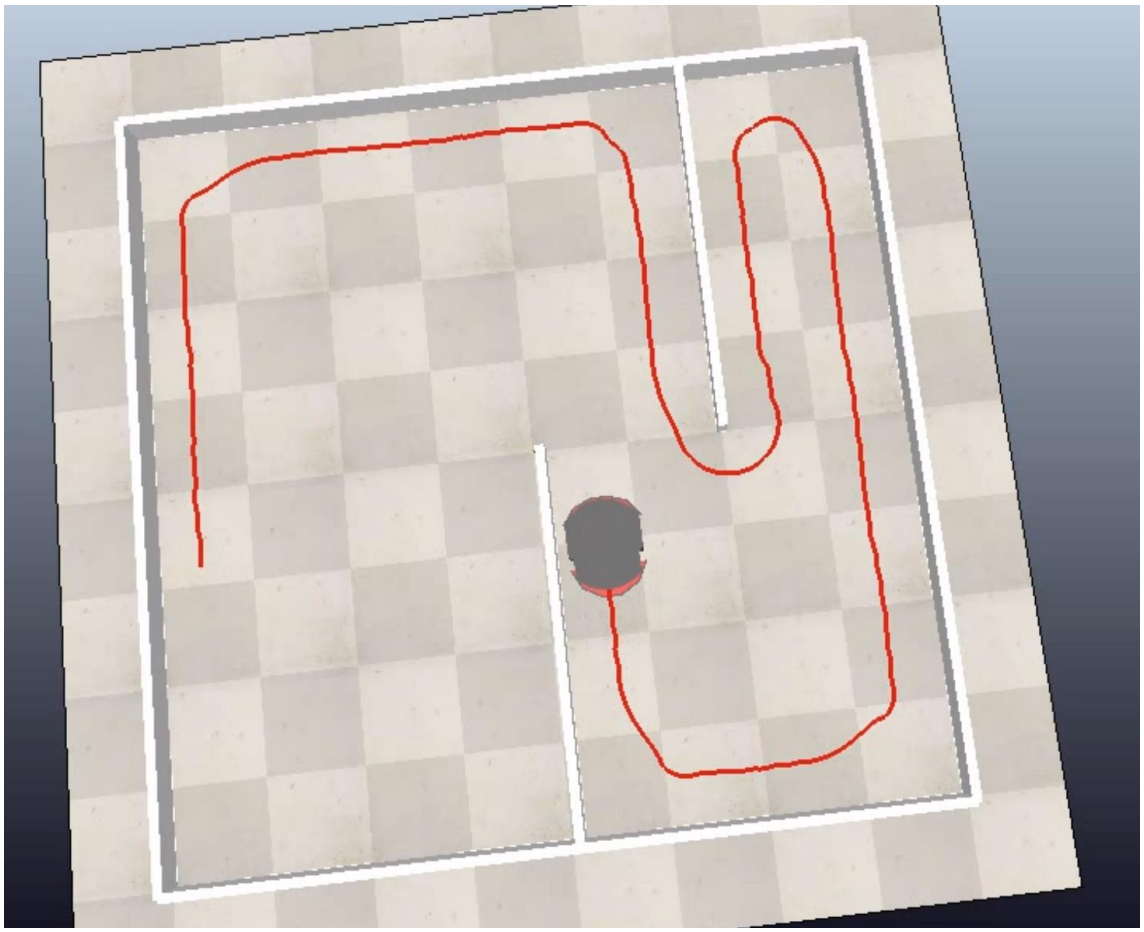
2.2. Primer experimento

Lo primero fue añadirle un movimiento por defecto (“lejos de todo”), para que no se quedase quieto, se eligió un giro hacia la izquierda.

Luego que se hizo fue la programación del “cabeceado”, consistía en calcular la diferencia entre los sensores 1 y 2 contra 14 y 13. Si los sensores indicaban distancias similares avanzaba más rápido (“cabecea largo”), si 1 y 2 eran mayores entonces acelera la rueda izquierda para alejarse de la pared (“cabecea lejos”) y si 14 y 13 eran mayores entonces acelera la rueda de la derecha para acercarse más a la pared (“cabecea cerca”); la aceleración de los motores es proporcional a la diferencia. Esta lógica de cabecear, le hemos solo se aplica cuando detecta una pared cercana a su izquierda (izq < 0.2).

Posteriormente se añadió que, si el frontal detectaba una pared cercana, fijase la rueda derecha se quedase quieta (Right Motor: 0) y la izquierda avanzase, de esta manera evitamos paredes en curva (“curva”).

Estas 3 lógicas nos dieron el siguiente resultado:



El video puede encontrarse en el siguiente enlace: enlace

Podemos ver que en las curvas las hacemos antes de tiempo, y después de girar vuelve a usar el movimiento por defecto para volver a acercarse a la pared. También uvo que poner el robot en esa posición de inicio para que el movimiento por defecto se moviese hacia la pared.

Pero el problema mas grande deriva de las paredes salientes y sus extremos que no detecta correctamente y se choca con ellos.

2.3. Segundo experimento

Luego añadimos más lógicas para bordear correctamente los extremos del saliente y mejorar su consistencia.

Primero, como se ve en el video a veces se acerca demasiado a la pared, asique si detecta (left < 0.1, “muy cerca de la pared”), se aleja ligeramente. Esto combinado con la condición anterior mantiene al robot entre 0.1 y 0.2 de distancia a la pared de la izquierda, salvo en excepciones como extremos y el inicio.

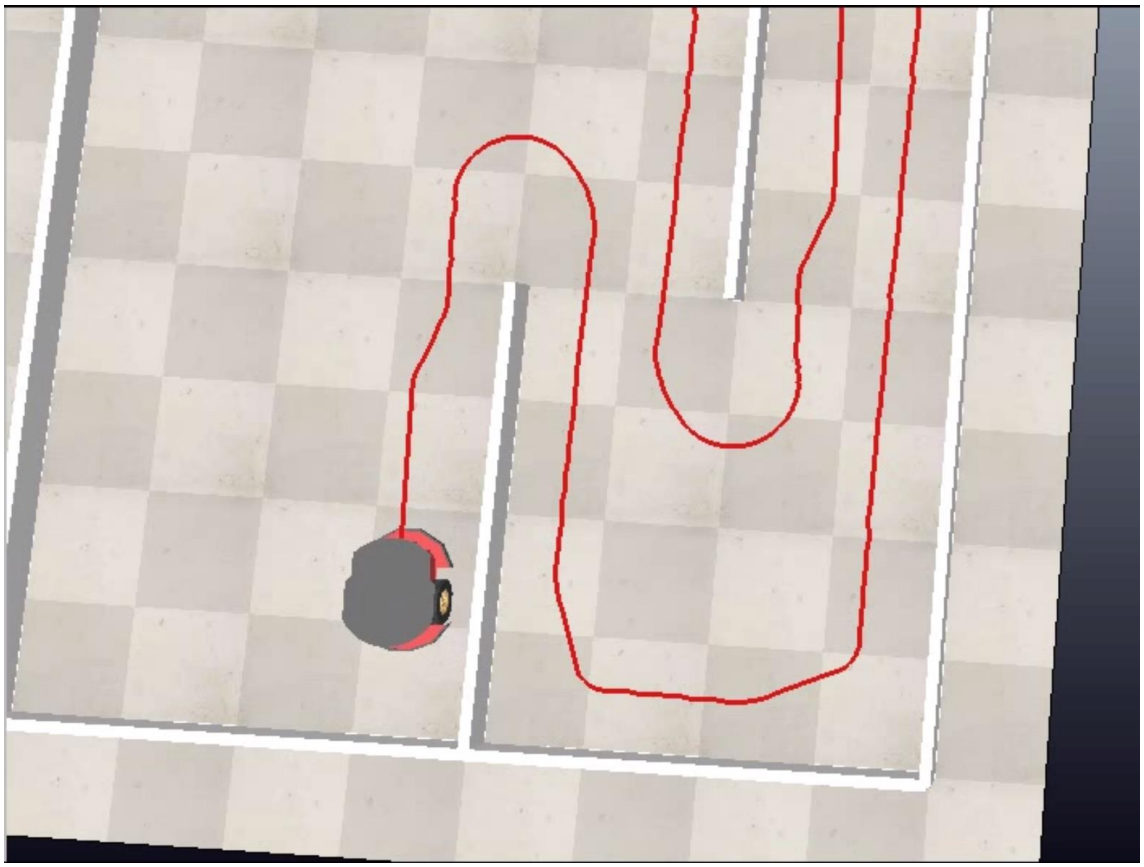
Para que tampoco se alejase mucho de la pared se añadió (“acercandose pared”) para que si detecta una pared en izq_fro, prácticamente se aplicara al final de las curvas para evitar el movimiento por defecto.

Otra, fue crear una variable global para que, en el INICIO, avance (“inicio”) hasta que se quede con una pared en frente (“curva inicio”) hasta que la izq detecte una pared que seguir.

Para los extremos, se pensó en:

1. “inicio curva”: Si el sensor 2 detecta que ya no tiene pared, avanza en vez de cabecear, para que no se acerque mas a la pared
2. “inicio extremo”: Compara la distancia de 0 y 15, ya que ambos miran hacia la izquierda, si el 0 da 0 y el 15 da 1, eso quiere decir que la pared se acaba, o sea, que está en uno de los salientes; es esta situación se le indico que siguiese recto hasta que el detector 0 detecte también 1.
3. “lejos de todo”: Hace el mismo movimiento que el predeterminado, o sea lejos de todo; a veces detectará el extremo de la pared, con el sensor 15, lo cual le hará avanzar un poco en vez de girar, lo cual ayuda evitar la pared con mayor margen.
4. “terminando vuelta extremo”: Luego usa los sensores de izq (que no detecte pared) y izq_front (que detecte una pared cercana) para determinar el otro lado de la pared esta por delante del robot y que el robot aún no se ha puesto en paralelo; en este caso gira a la derecha y avanza poco a poco, esto compensa el movimiento que tendrá cuando izq_front se aleje demasiado, causando un poco de cabeceo.
5. Cuando izq detecte algo cercano, podrá si está muy cerca hacer el movimiento de curva para alejarse bruscamente de la pared. O si cree que no se va a chocar volverá a seguir la pared.

Una vez añadido estas lógicas este es el resultado:



El video puede encontrarse en el siguiente enlace: enlace

Podemos ver que para los extremos de paredes los gira correctamente, pero el margen que deja de separación, aunque sirva para evitar los choques al terminar de girar, podría reducirse.

Intentar que el recorrido lo haga mas rápido. Sobre todo, en las rectas.

3. Código

DebugDisplay, es para mostrar la pantalla ya mencionada, el código de condicional es el que se encuentra en el método: avoid.

```
def avoid(readings):
  global INICIO
  # Initial State
  lspeed = 0; rspeed = 0; status = "Parado"
  izq, izq_fro, front, izq_back = (
        np.mean([readings[0], readings[15]]),
        np.mean([readings[1], readings[2]]),
        np.mean([readings[3], readings[4]]),
        np.mean([readings[13], readings[14]]))

  if INICIO:
    if front < 0.4 or izq < 0.4:
      INICIO = False
    lspeed, rspeed, status = 0.5, 0.5, "inicio"
  elif front < 0.4:
    if izq < 0.3:
      lspeed, rspeed, status = 1, 0, "curva"
  elif izq_fro < 0.4 and izq > 0.5:
    lspeed, rspeed, status = 0.3, 0.1, "terminando vuelta extremo"
  else:
    if (readings[0] > 0.9 and readings[15] < 0.4):
      lspeed, rspeed, status = +0.4, 0.4, "inicio extremo"
    elif izq < 0.2:
      if readings[2] == 1:
        lspeed, rspeed, status = +0.4, 0.4, "inicio curva"
      elif izq < 0.1:
        lspeed, rspeed, status = +0.3, 0.2, "muy cerca de la pared"
      else:
        if np.isclose(izq_fro, izq_back, atol = 0.05):
          lspeed, rspeed, status = +0.5, +0.5, "cabecea largo"
        elif izq_fro < izq_back:
          lspeed, rspeed, status = +0.2 + (izq_back - izq_fro), +0.2, "cabecea lejos"
        else:
          lspeed, rspeed, status = +0.2, +0.2 + (izq_fro - izq_back), "cabecea cerca"
    elif  izq_fro < 0.5:
      lspeed, rspeed, status = +0.3, +0.3, "acercandose pared"
    else:
      lspeed, rspeed, status = +0.1, +0.45, "lejos de todo"
  return lspeed, rspeed, status
```

3.1. Código Entero

```
def color_map(value):
    if value < 0.5:
        return pg.Color(255, round(value*(255-1)*2), 0)
    else:
        return pg.Color(round(255 - (value - 0.5)*(255-1)*2), 255, 0)

class DebugDisplay:
    def __init__(self):
        self.state   = None
        self.queue   = Queue()
        self.process = Process(target = self.debug_thread, args = (self.queue, ))

        self.process.daemon = True
        self.process.start()

    def debug_thread(self, queue):
        pg.init()
        self.width = 300
        self.height = 400
        self.screen = pg.display.set_mode([300, 500])
        pg.display.set_caption( "Robotica Sensor Display: Debug")
        self.font = pg.font.SysFont('Times New Roman', 22)

        keep_drawing = True
        while keep_drawing:
            self.screen.fill( (255, 255, 255)) # White Background

            for event in pg.event.get():
                if event.type == pg.QUIT: keep_drawing =  False

            while not queue.empty(): self.state = queue.get()
            # self.state = [(0, 2, 0.5, 0, 1, 0, 0, 0,), (9, 0)]

            self.debug_draw()

            pg.display.flip()

    def debug_draw(self):
        if self.state is None: return
        self.draw_sensors()
        self.draw_speed()

    def draw_sensors(self):

        readings = self.state[0]

        left = color_map(readings[0])
        pg.draw.line(self.screen, left, (40, 250), (40, 125), width=WIDTH)

        left_text = self.font.render("{:2.2f}".format(readings[0]), True, (0, 0, 0))
        left_text = pg.transform.rotate(left_text, 90)
        left_rect = left_text.get_rect()
        left_rect.centerx = 25
        left_rect.centery = 190
        self.screen.blit(left_text, left_rect)

        left_top = color_map(readings[1])
        pg.draw.line(self.screen, left_top, (40, 125), (100, 50), width=WIDTH)

        left_top_text = self.font.render("{:2.2f}".format(readings[1]), True, (0, 0, 0))
        left_top_text = pg.transform.rotate(left_top_text, 50)
        left_top_rect = left_top_text.get_rect()
        left_top_rect.centerx = 55
        left_top_rect.centery = 75
        self.screen.blit(left_top_text, left_top_rect)

        top_color = color_map(readings[2])
        pg.draw.line(self.screen, top_color, (100, 50), (200, 50), width=WIDTH)
        top_text = self.font.render("{:2.2f}".format(readings[2]), True, (0, 0, 0))
        # top_text = pg.transform.rotate(left_text, 90)
        top_rect = top_text.get_rect()
        top_rect.centerx = 150
        top_rect.centery = 30
        self.screen.blit(top_text, top_rect)

        right_top_color = color_map(readings[3])
        pg.draw.line(self.screen, right_top_color, (200, 50), (self.width - 40, 125), width=WIDTH)

        left_text = self.font.render("{:2.2f}".format(readings[3]), True, (0, 0, 0))
        left_text = pg.transform.rotate(left_text, - 50)
```

```
        lett_rect = left_text.get_rect()
        lett_rect.centerx = self.width - 55
        lett_rect.centery = 75
        self.screen.blit(left_text, lett_rect)

        right = color_map(readings[4])
        pg.draw.line(self.screen, right, (self.width - 40, 250), (self.width - 40, 125), width=WIDTH)

        right_text = self.font.render("{:2.2f}".format(readings[4]), True, (0, 0, 0))
        right_text = pg.transform.rotate(right_text, - 90)
        right_rect = right_text.get_rect()
        right_rect.centerx = self.width - 25
        right_rect.centery = 190
        self.screen.blit(right_text, right_rect)

        behind_color = color_map(readings[5])
        pg.draw.line(self.screen, behind_color, (100, 325), (200, 325), width=WIDTH)
        behind_text = self.font.render("{:2.2f}".format(readings[5]), True, (0, 0, 0))
        behind_rect = behind_text.get_rect()
        behind_rect.centerx = 150
        behind_rect.centery = 350
        self.screen.blit(behind_text, behind_rect)

        left_bot = color_map(readings[6])     # (40, 125), (100, 50)
        pg.draw.line(self.screen, left_bot, (40, 250), (100, 325), width=WIDTH)
        left_bot_text = self.font.render("{:2.2f}".format(readings[6]), True, (0, 0, 0))
        left_bot_text = pg.transform.rotate(left_bot_text, - 50)
        left_bop_rect = left_bot_text.get_rect()
        left_bop_rect.centerx = 60
        left_bop_rect.centery = 300
        self.screen.blit(left_bot_text, left_bop_rect)

    def draw_speed(self):
        speed = self.state[1]
        left_motor_text  = self.font.render("Left  Motor:", True, (0, 0, 0))
        right_motor_text = self.font.render("Right Motor:", True, (0, 0, 0))

        left_motor_rect = left_motor_text.get_rect()
        left_motor_rect.left = 50
        left_motor_rect.centery = 380
        right_motor_rect = right_motor_text.get_rect()
        right_motor_rect.left = 50
        right_motor_rect.centery = 410
        self.screen.blit(left_motor_text, left_motor_rect)
        self.screen.blit(right_motor_text, right_motor_rect)

        left_motor_text  = self.font.render("{:2.3f}".format(speed[0]), True, (0, 0, 0))
        right_motor_text = self.font.render("{:2.3f}".format(speed[1]), True, (0, 0, 0))

        left_motor_rect = left_motor_text.get_rect()
        left_motor_rect.left = 175
        left_motor_rect.centery = 380
        right_motor_rect = right_motor_text.get_rect()
        right_motor_rect.left = 175
        right_motor_rect.centery = 410
        self.screen.blit(left_motor_text, left_motor_rect)
        self.screen.blit(right_motor_text, right_motor_rect)

        status_text = self.font.render(str(speed[2]), True, (0, 0, 0))
        status_rect = status_text.get_rect()
        status_rect.left = 50
        status_rect.centery = 440
        self.screen.blit(status_text, status_rect)

    def update_env (self, readings, speed):
        if self.queue is None: return
        self.queue.put((readings, speed))

import robotica
from multiprocessing import Process, Queue
import pygame as pg
import numpy as np

WIDTH: int = 10; INICIO: bool = True
def avoid(readings):
    global INICIO
    lspeed = 0; rspeed = 0; status = "Parado"
    izq, izq_fro, front, der_fro, der, atra, izq_back = (
            np.mean([readings[0], readings[15]]),
            np.mean([readings[1], readings[2]]),
            np.mean([readings[3], readings[4]]),
```

```
            np.mean([readings[5], readings[6]]),
            np.mean([readings[7], readings[8]]),
            np.mean([readings[11], readings[12]]),
            np.mean([readings[13], readings[14]]))

    if INICIO:
        if front < 0.4:
            lspeed, rspeed, status = 1, 0, "curva inicio"
        elif izq < 0.3:
            INICIO = False
        else:
            lspeed, rspeed, status = 0.5, 0.5, "inicio"
    elif front < 0.4:
        if izq < 0.3:
            lspeed, rspeed, status = 1, 0, "curva"
    elif izq_fro < 0.4 and izq > 0.5:
        lspeed, rspeed, status = 0.3, 0.1, "terminando vuelta extremo"

    else:
        if (readings[0] > 0.9 and readings[15] < 0.4):
            lspeed, rspeed, status = +0.4, 0.4, "inicio extremo"
        elif izq < 0.2:
            if readings[2] == 1:
                lspeed, rspeed, status = +0.4, 0.4, "inicio curva"
            elif izq < 0.1:
                lspeed, rspeed, status = +0.3, 0.2, "muy cerca de la pared"
            else:
                if np.isclose(izq_fro, izq_back, atol = 0.05):
                    lspeed, rspeed, status = +0.5, +0.5, "cabecea largo"
                elif izq_fro < izq_back:
                    lspeed, rspeed, status = +0.2 + (izq_back - izq_fro), +0.2, "cabecea lejos"
                else:
                    lspeed, rspeed, status = +0.2, +0.2 + (izq_fro - izq_back), "cabecea cerca"
        elif  izq_fro < 0.5:
            lspeed, rspeed, status = +0.3, +0.3, "acercandose pared"
        else:
            lspeed, rspeed, status = +0.1, +0.45, "lejos de todo"
    return lspeed, rspeed, status


def main(args=None):
    display = DebugDisplay()
    coppelia = robotica.Coppelia()
    robot = robotica.P3DX(coppelia.sim, 'PioneerP3DX')
    coppelia.start_simulation()
    while coppelia.is_running():
        readings = robot.get_sonar()
        readings_use = (
            np.mean([readings[0], readings[15]]),
            np.mean([readings[1], readings[2]]),
            np.mean([readings[3], readings[4]]),
            np.mean([readings[5], readings[6]]),
            np.mean([readings[7], readings[8]]),
            np.mean([readings[11], readings[12]]),
            np.mean([readings[13], readings[14]]))
        lspeed, rspeed, status = avoid(readings)
        display.update_env(readings_use, (lspeed, rspeed, status))
        robot.set_speed(lspeed, rspeed)

    coppelia.stop_simulation()

if __name__ == '__main__':
    main()
```