UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingenieros Informáticos

MADRID

Máster Universitario en Inteligencia Artificial

Práctica 1 – Robots Autónomos

ALEJANDRO MUÑOZ NAVARRO

FERNANDO PLANES RUIZ

27/10/2020

Capítulo 1
Código Proporcionado

En este capítulo se describe el material proporcionado por el profesor para el perfecto desarrollo de la práctica.

Como estado inicial, partimos de una escena "*muia-2020.ttt*", un cliente "*muia-2020-client*" y un entorno de simulación "*CoppeliaSim Edu*".

A continuación, se desarrollará un análisis del código proporcionado en el cliente.

1. Robot

Una vez realizada la conexión, dado el "*clientID*", se realiza la inicialización del robot. Para ello, la función "*getRobotHandles(clientID)*" inicializa los motores, la cámara, y los sensores.

1..1. Motores

El robot consta de dos motores a cada lado. Los cuales pueden moverse hacia delante o hacia atrás con velocidad variable. De tal forma que podremos realizar giros dándole a un motor una velocidad mayor que la del otro, podremos ir hacia delante con una velocidad positiva o marcha atrás usando una velocidad negativa.

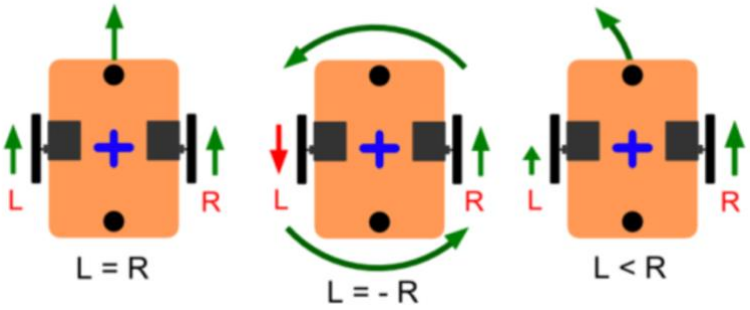


Figura 1. Funcionamiento de los motores

1..2. Cámara

El robot consta de una cámara situada en la parte delantera. Esta cámara únicamente detecta la bola roja que deberemos seguir. Por lo tanto, no es capaz de detectar ningún otro obstáculo en la escena.

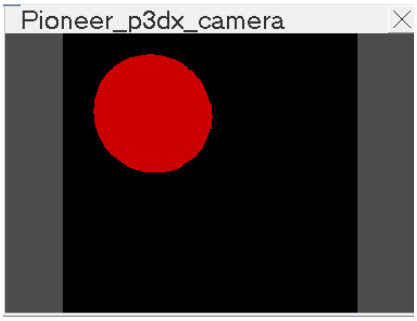


Figura 2. Cámara

1..3. Sensores

El robot consta de dieciséis sensores distribuidos por los 360º del robot. De tal forma que ocho de ellos ocupan la parte delantera del robot y los otros ocho la parte trasera.

Estos sensores detectan todos los objetos a su alrededor, ya sean un obstáculo (pared) o la bola. Con estos obtenemos la distancia a la que se encuentra el robot de aquellos objetos, siendo 1 la distancia máxima de detección del sensor. De tal forma que podamos movernos para impedir chocarnos.

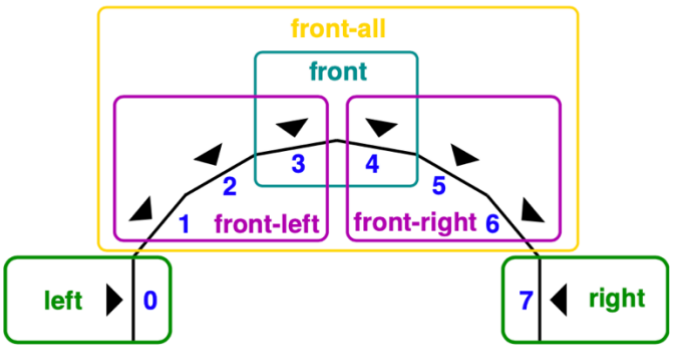


Figura 3. Sensores delanteros

Capítulo 2
Comportamientos desarrollados

En este capítulo se describe el material desarrollado por los alumnos para el desarrollo de las tareas.

A continuación, se realizará el análisis de los diferentes comportamientos desarrollados.

2. Controladores

Para controlar el robot, se han identificado las diferentes funciones que tendrá que realizar el robot, de esta forma podemos aplicar diferentes algoritmos en diferentes situaciones.

A continuación, desarrollaremos cada una de estas funciones:

2.1. Comportamiento de búsqueda de la bola

Este comportamiento se encarga de la búsqueda de la bola. El robot realiza un giro sobre sí mismo durante un número de pasos suficientes como para girar más de 360 grados, tras los cuales intentará avanzar otros tantos pasos en línea recta, aunque si se encuentra con obstáculos, los irá esquivando (gracias a otro comportamiento), incluso en ocasiones cambiará el sentido de la marcha, generando así un comportamiento suficientemente aleatorio como para explorar razonablemente bien la escena.

Se han desarrollado escenas con zonas de difícil acceso para comprobar que el robot es capaz de entrar en ellas, explorarlas y posteriormente salir.

Para ello, hemos optado por el giro del robot en la dirección al error de la bola en el eje X. Este error es calculado dentro del rango [-1,1] siendo -1 si la bola se desplaza a la izquierda y 1 si la bola se desplaza a la derecha.

Por lo tanto, si el error de la bola es negativo, el robot girará sobre si mismo hacia la izquierda y en caso contrario girará hacia la derecha.

2.2. Comportamiento para evitar objetos

El robot puede encontrarse con obstáculos por el camino. Es por esto por lo que deberá de evitarlos a través del siguiente comportamiento. Este comportamiento tiene la mayor prioridad y si el robot se aproxima a un obstáculo se activará.

Para ello, calculamos previamente el índice del sensor con valor mínimo en el sonar, de esta manera podemos localizar donde se encuentra el obstáculo más preocupante. Una vez obtenida la posición, podemos tratarla a través de condiciones:

- Si el índice del sensor se encuentra en [1, 2, 3, 9, 10, 11], el obstáculo se encuentra en la diagonal izquierda de la parte delantera o la derecha de la parte trasera.

- Si el índice del sensor se encuentra en [4, 5, 6, 12, 13, 14], el obstáculo se encuentra en la diagonal derecha de la parte delantera o izquierda de la parte trasera.

- En caso contrario, sabemos que el obstáculo se encuentra en alguno de los dos laterales y realizamos las siguientes operaciones:

    - Calculamos un giro para el lado contrario al que se encuentra el objeto.
    - Si hay obstáculos a ambos lados y el robot se encontrase más pegado al lateral izquierdo que al lateral derecho, daremos un giro más fuerte hacia la derecha.
    - Si hay obstáculos a ambos lados y el robot se encontrase más pegado al lateral derecho que al lateral izquierdo, daremos un fuerte giro hacia la izquierda.
    - Si además tenemos algo delante, seguramente estemos ante una esquina, por lo que haremos un giro a mucha velocidad en la dirección de giro que calculada previamente. Si tenemos algo detrás, aplicamos esto mismo, pero con la dirección contraria.
    - Con esto tendríamos el giro deseado, pero además haremos otra comprobación y según la cercanía del obstáculo ajustaremos la velocidad lineal del robot para ir proporcional a la distancia.
    - Por último, si estamos yendo hacia delante y tenemos algo muy cerca, cambiaremos el sentido e iremos marcha atrás y si estábamos yendo ya hacia atrás y nos acercamos demasiado a un obstáculo, cambiaremos el sentido hacia delante.

Se intentó desarrollar un controlador borroso, pero quizás por nuestra poca experiencia con ellos, no conseguimos que funcionase mejor que el que tenemos, que se desenvuelve relativamente bien entrando y paseando por pasillos estrechos.

Este comportamiento además incluye una distinción sobre si la pelota aparece dentro del campo de visión de la cámara. El comportamiento de esquivar objetos cuando se está siguiendo a la pelota de cerca se detallará en otro apartado.

2.3. Comportamiento de acercamiento a la bola

Una vez encontrada la bola, el robot se acercará a una velocidad relativamente rápida a ella. Para controlar el giro del robot mientras se va acercando, comprobamos la coordenada horizontal de la bola en la imagen, que será reescalada de [0, 1] a [-1,1] de manera que valores positivos indiquen que la bola se ubica a la derecha y los negativos indicarán izquierda.

Empleamos un controlador PID con un “mecanismo de clamping” para ajustar la parte integral. En general hemos realizado los cálculos en términos de velocidad lineal y velocidad de giro pues nos es más cómodo trabajar así y luego convertirlos en las señales correspondientes a cada motor.

2.4. Comportamiento de escolta de la bola

Una vez nos encontramos cerca de la bola, deberemos reducir la velocidad y quedarnos a una velocidad aproximada a la de esta. Para ello usaremos el mismo controlador PID para el giro que en la fase de acercamiento, pero emplearemos una velocidad lineal mucho menor.

A su vez, tenemos un segundo controlador PID dentro del código relativo a evitar colisiones, puesto que el robot querrá ir en la dirección de la bola, pero a la vez tendrá que adaptar su movimiento para esquivar los obstáculos.

En ocasiones, cuando la bola realiza un giro, el robot termina acercándose demasiado a ella con peligro de colisionar, por ello si la bola se acerca demasiado al robot, este realizará un pequeño movimiento marcha atrás.

Hemos implementado la marcha atrás sin bola pues pensamos que podría darse el caso en que la pelota entrase a un callejón sin salida y diese la vuelta para pasar a perseguir al robot desde delante. Aunque no fuimos capaces de combinar del todo la marcha atrás con giros y persecución de la pelota, es probable que el robot sea capaz de resolver algunos casos por la combinación de comportamientos implementados.

Capítulo 3
Pruebas y resultados obtenidos

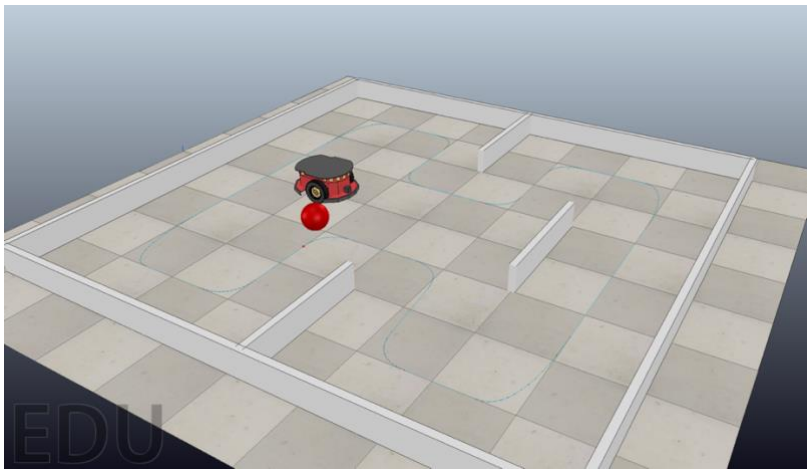
En este capítulo se describen las pruebas desarrolladas por los alumnos para el testeo de los comportamientos implementados.

A continuación, se realizará el análisis de cada uno de los escenarios propuestos.

3. Escenario 1

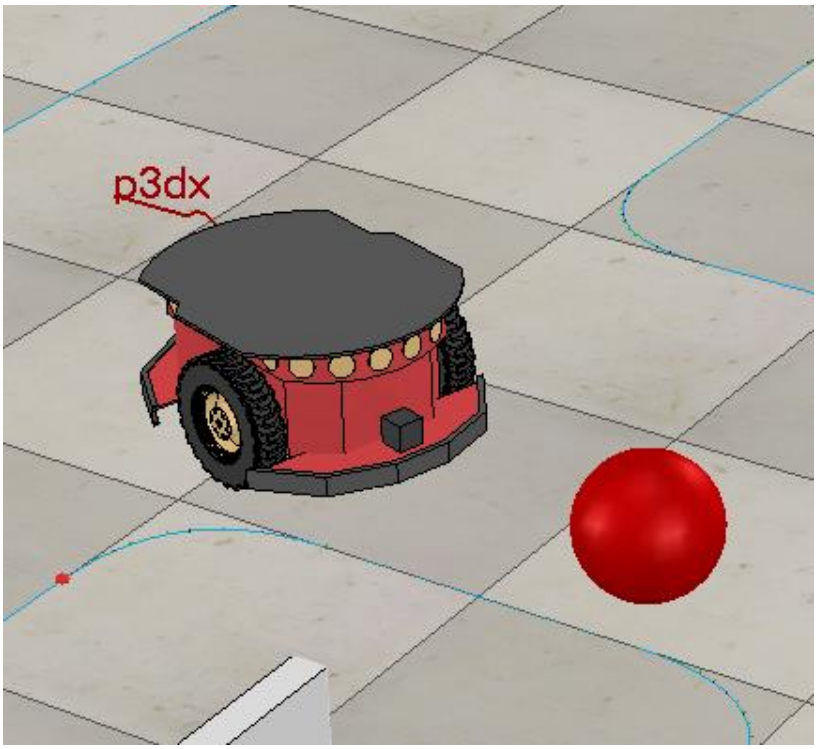
Esta primera escena es la que se nos entregó en clase. A partir de ella desarrollamos el resto y también nos sirvió para hacer algunas de nuestras primeras pruebas o para comprobar que el robot tuviese un movimiento más o menos fluido en una escena relativamente sencilla.

3.1. Estado inicial

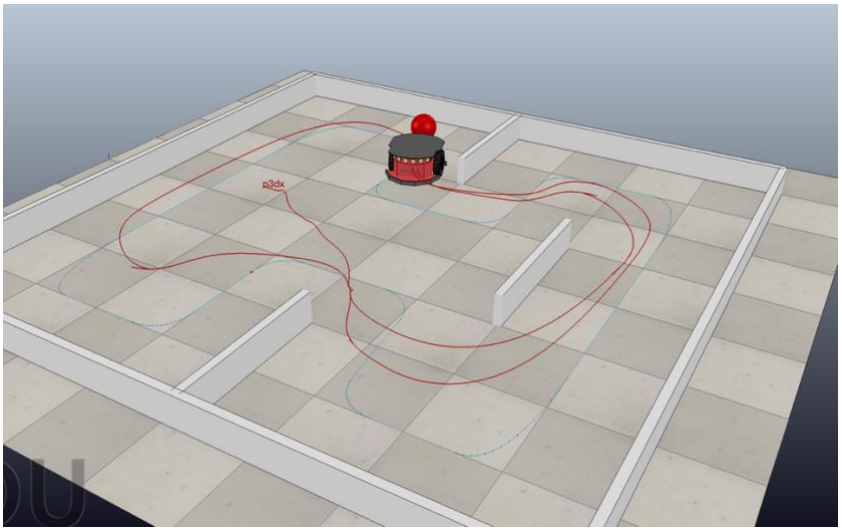


3.2. Encontrar y seguir la bola

Nuestro robot arranca girando sobre sí mismo hasta encontrar la pelota, momento en el cual se aproxima a ella. Rápidamente aparece un primer obstáculo a su derecha, por lo que el robot debe corregir ligeramente su trayectoria. Tras esto la pelota gira y el robot se encuentra un nuevo obstáculo a su izquierda, además de recorrer un primer pasillo (amplio). El resto de la escena es más sencilla que estos primeros pasos.



3.3. Estado final

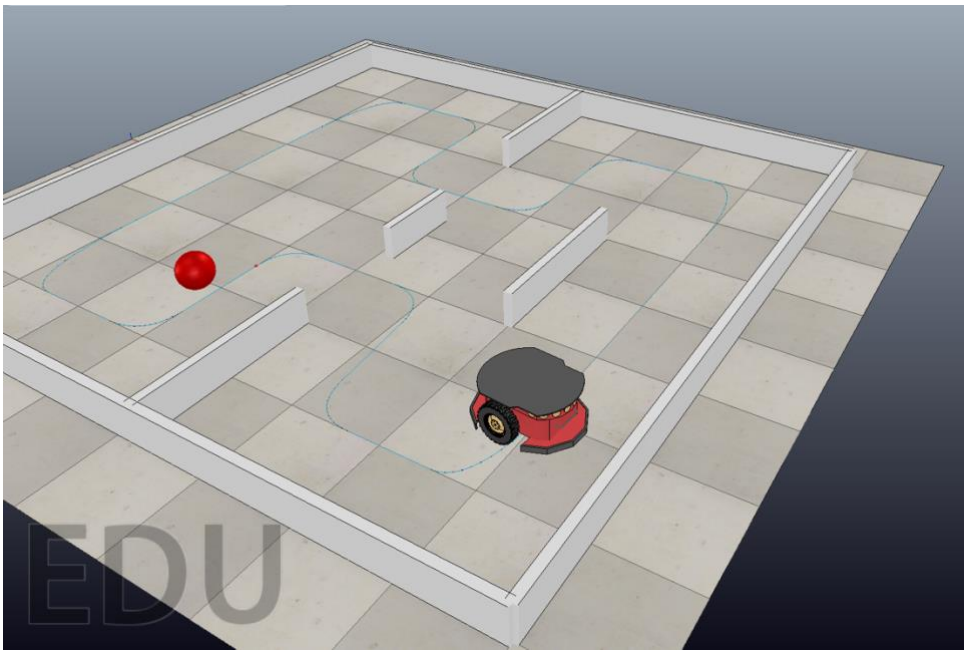


4. Escenario 2

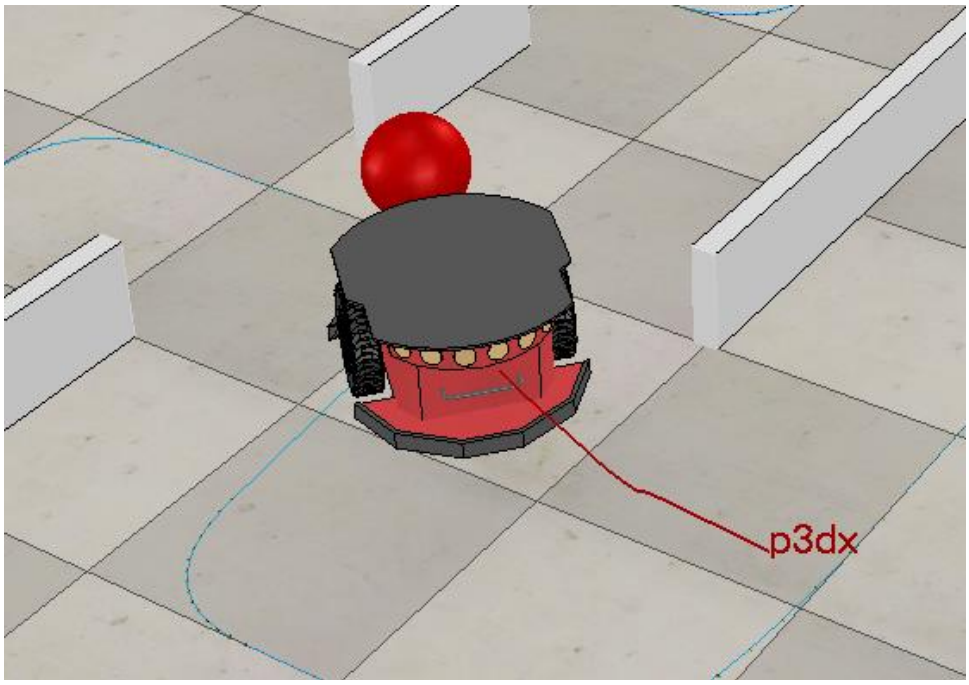
En esta escena añadimos una pieza para hacer que el robot tuviese que pasar entre puertas más estrechas y que si no lo realizase correctamente podría perder a la bola de vista.

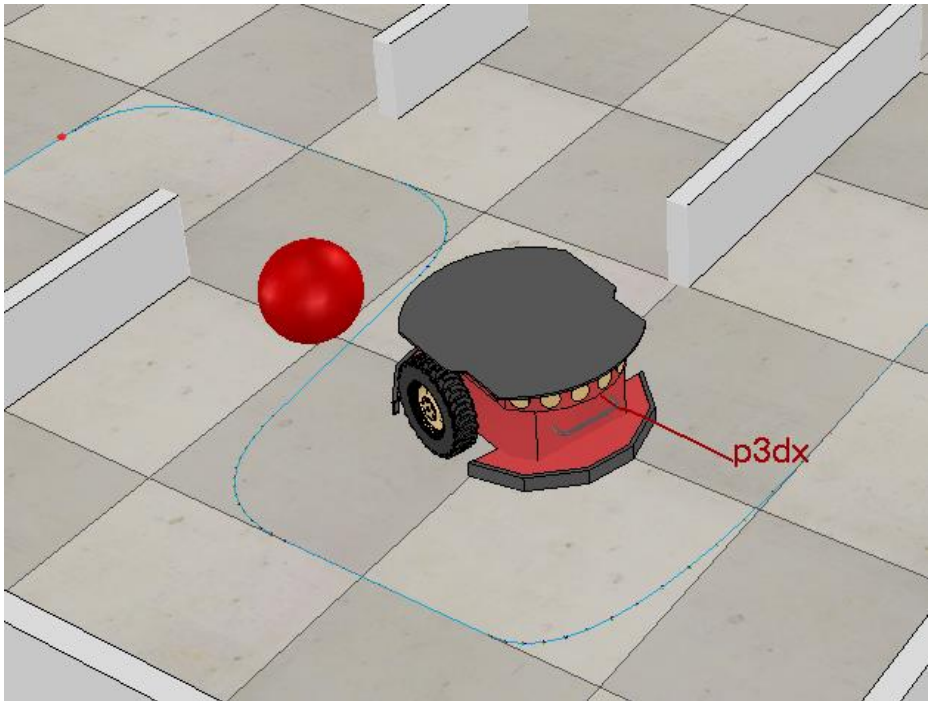
Es ligeramente más difícil que la primera escena por la posible perdida de visibilidad y porque la disposición de los obstáculos hará que el camino del robot sea más oscilante.

4.1. Estado inicial

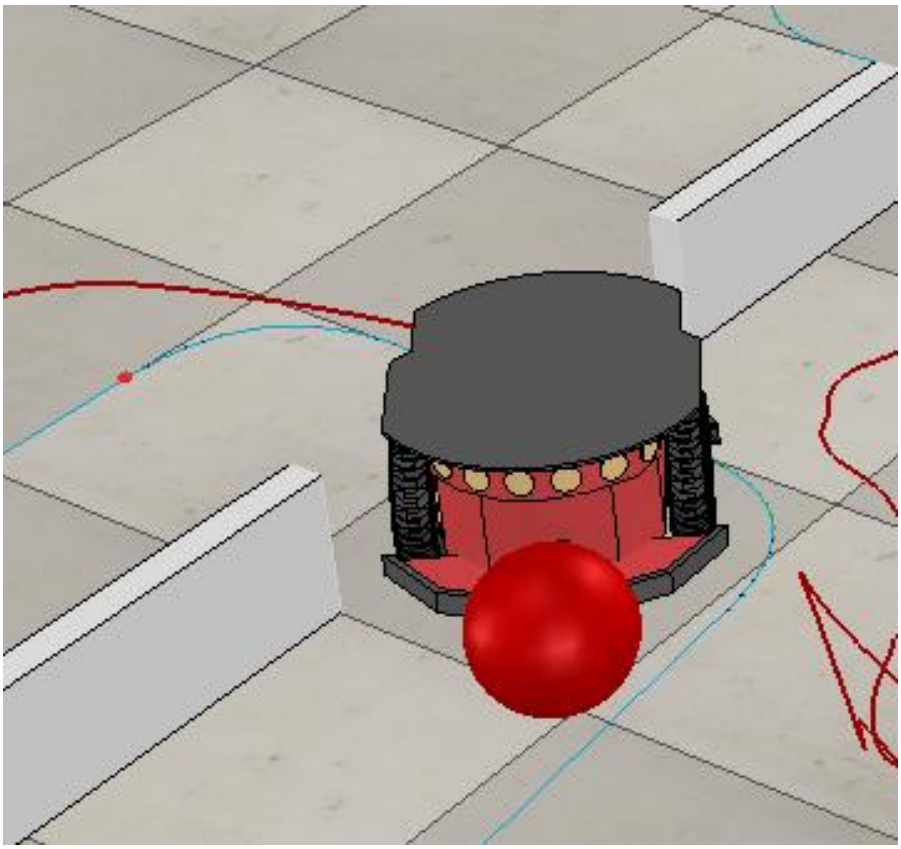


4.2. Esquivar bola

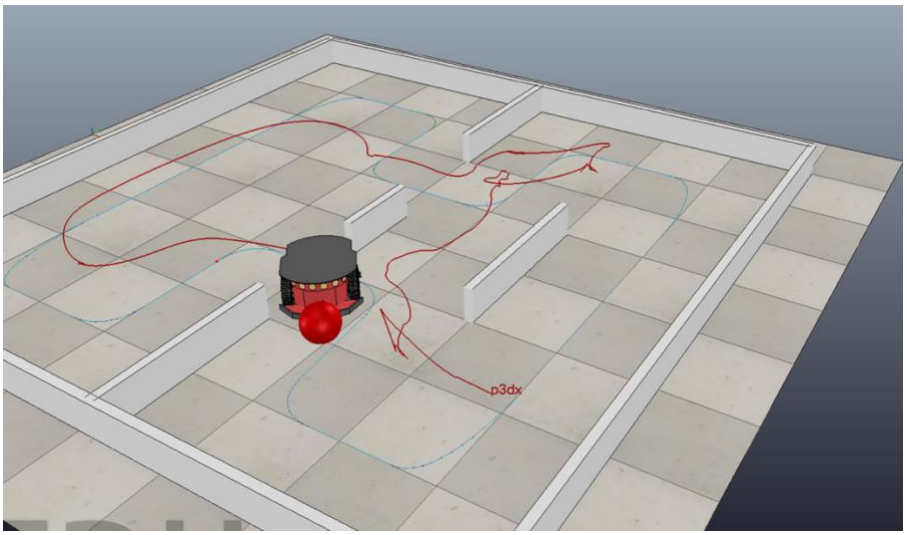




4.3. Pasar por lugares estrechos



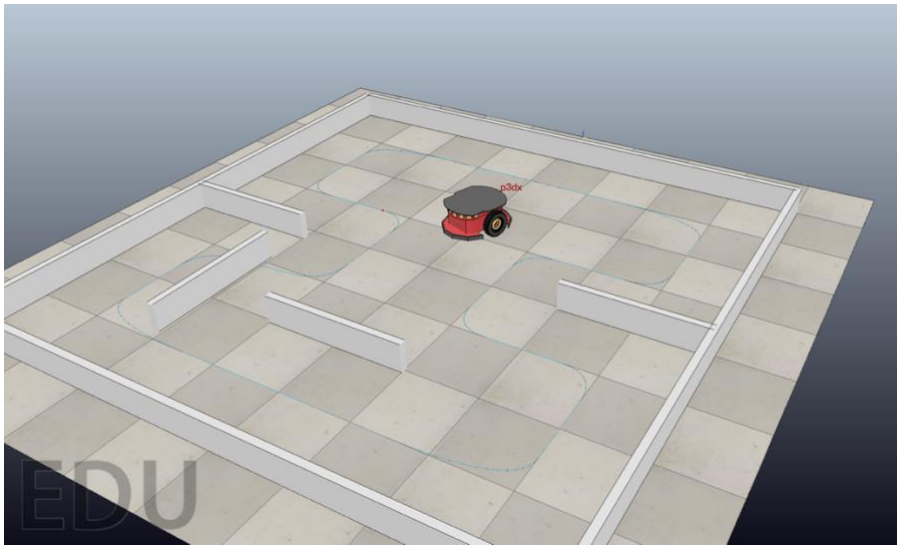
4.4. Estado final



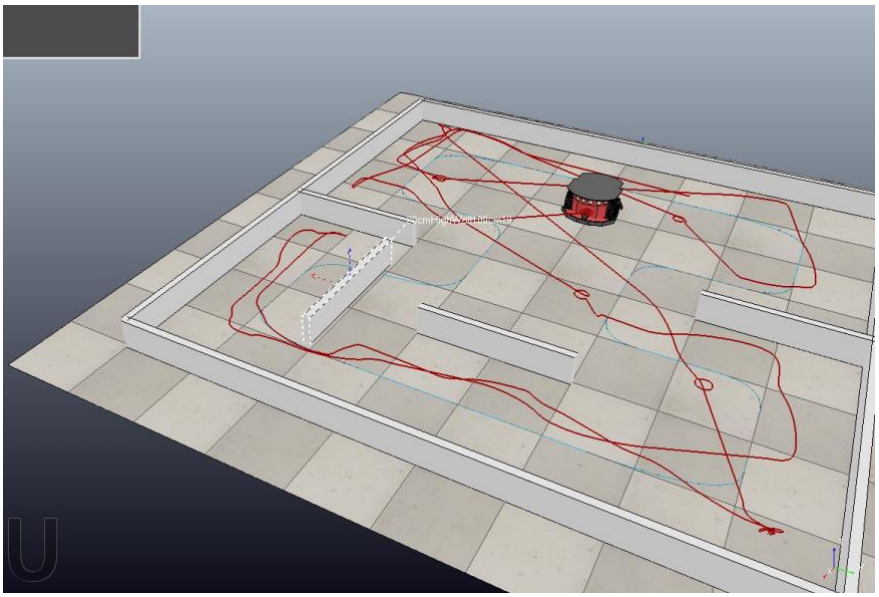
5. Escenario 3

Desarrollamos una escena sin bola y con unas entradas especialmente estrechas para comprobar que el robot fuese capaz de entrar y salir de ellas, además de explorar de manera razonable el espacio, sin quedarse demasiado tiempo en una misma zona.

5.1. Estado inicial



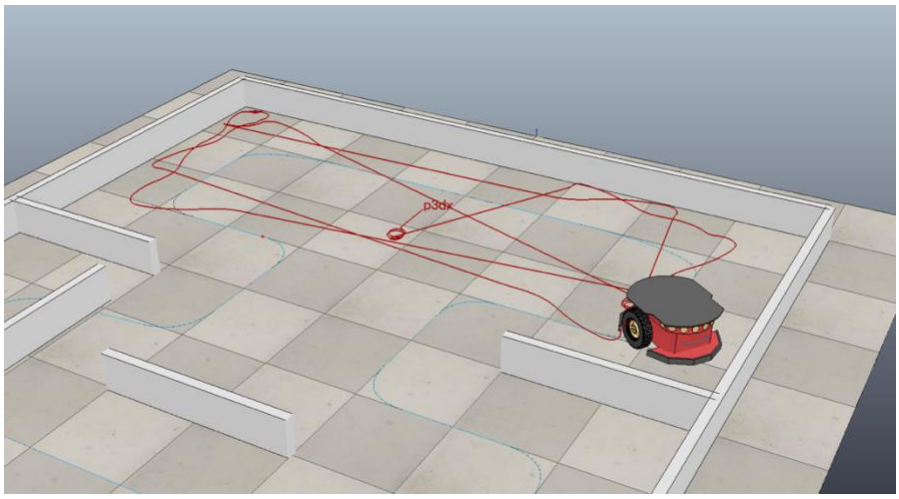
5.2. Estado final

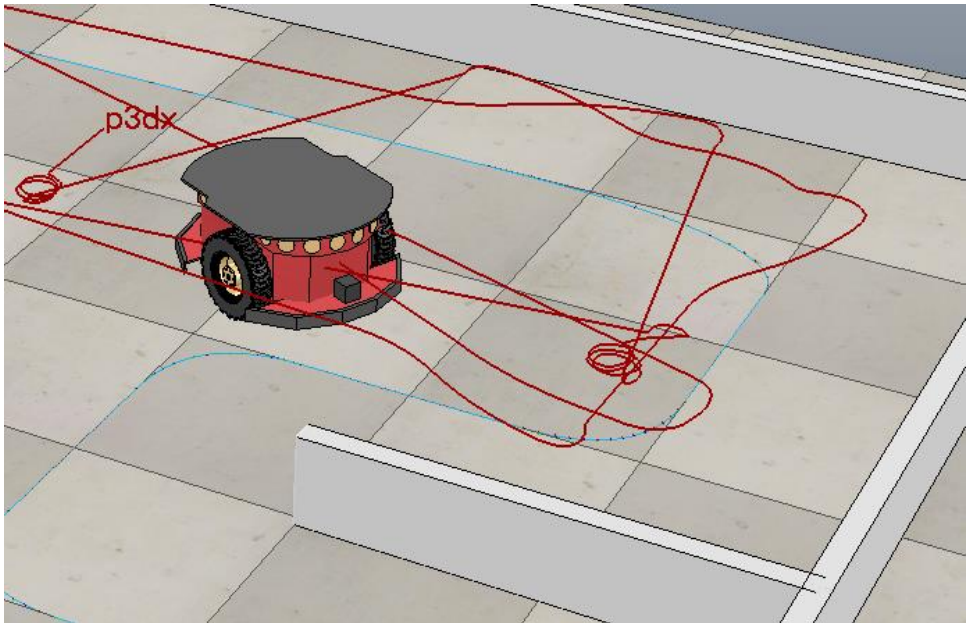


5.3. Estado marcha atrás

A la par que desarrollamos el movimiento de búsqueda de la pelota, desarrollamos la mayoría del comportamiento para evitar obstáculos. En especial, se añadieron modificaciones más allá de alejarse de los obstáculos para los casos en que se encontrase en una esquina y también los casos en que se acercase demasiado a un obstáculo, momento en el cual el robot cambia el sentido de la marcha y la parte trasera pasa a actuar de manera simétrica a la delantera en términos de evitar los obstáculos.

Marcha atrás

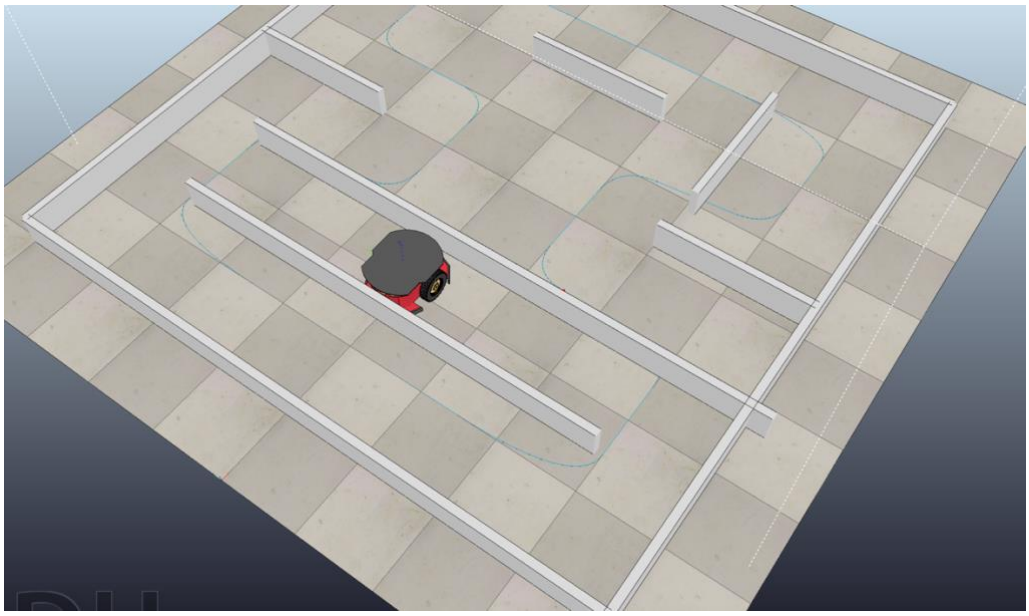




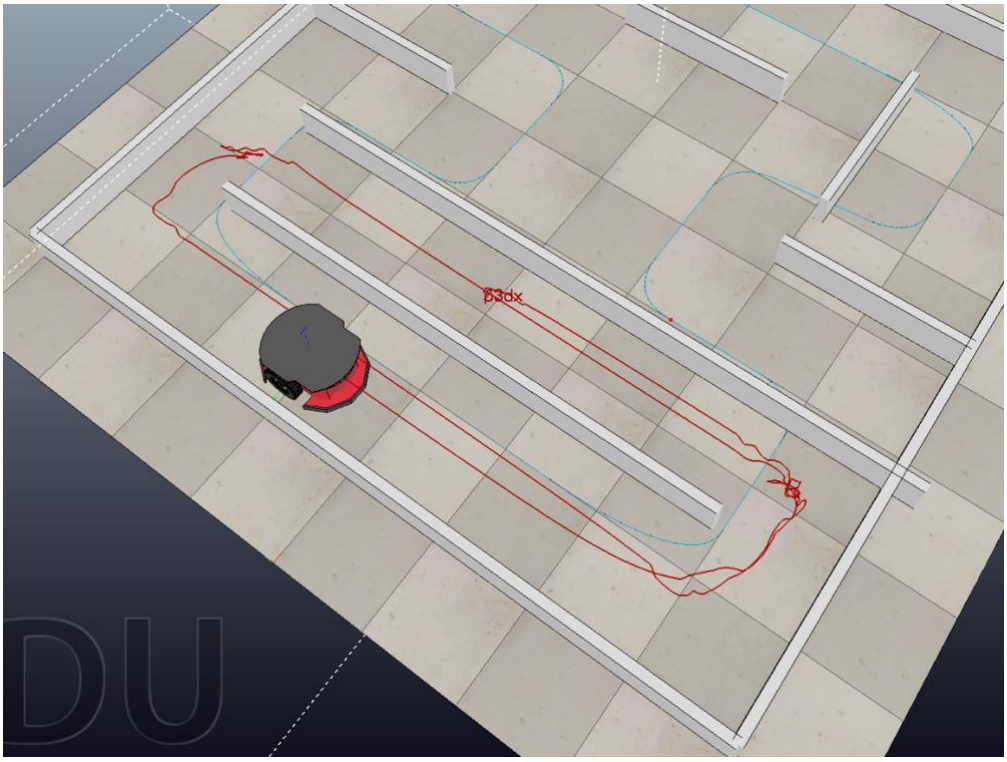
6. Escenario 4

Desarrollamos un escenario especialmente estrecho para comprobar como se podría desenvolver nuestro robot. Se consiguió desarrollarlo de una forma en que era capaz de recorrerlo rápidamente, pero al añadir más comportamientos, su interacción hizo que fuese algo menos eficiente. No obstante, asumimos que en general no se encontrará en situaciones tan complicadas y en cualquier caso si fuese así, el robot puede resolverlas.

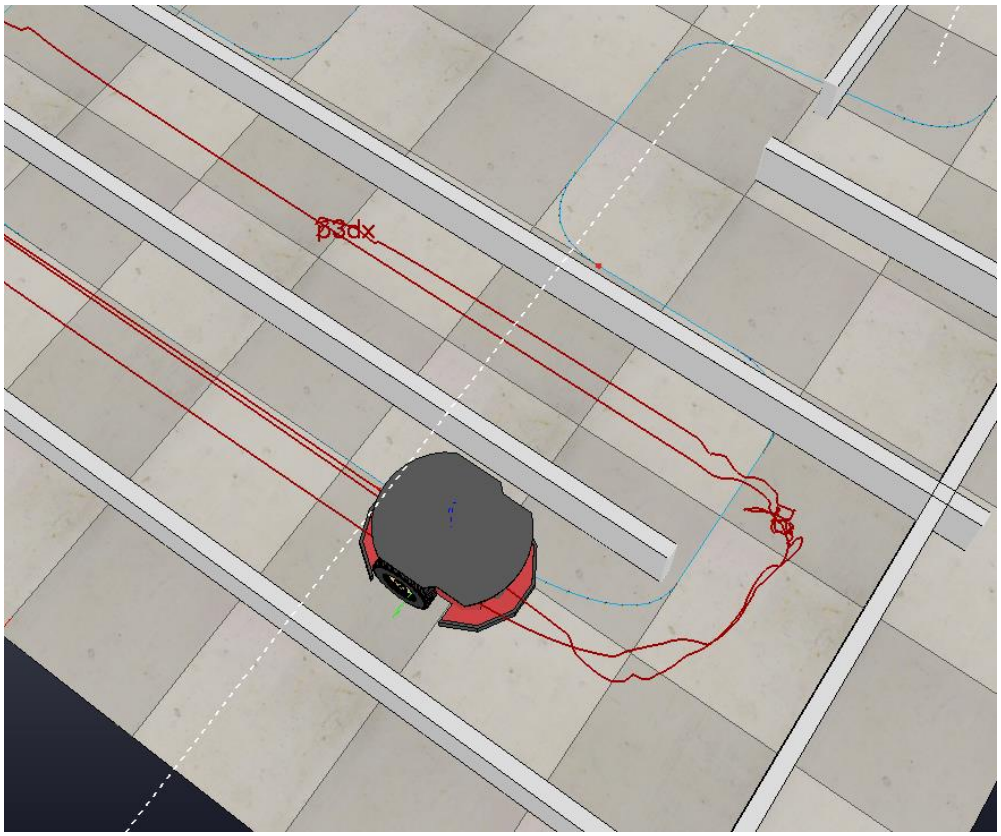
6.1. Estado inicial



6.2. Estado final



6.3. Estado marcha atrás lugares estrechos



Anexo I
Código Cliente

En este Anexo se recopila el código del cliente desarrollado en el lenguaje Python 3 y probado en la plataforma "*CoppeliaSim Edu*"

```
#!/usr/bin/python3

# ------------------------------------------------------------------

print('### Script:', __file__)

# ------------------------------------------------------------------

import math
import sys
import time

# import cv2 as cv
```

```
# import numpy as np
import sim

# ------------------------------------------------------------------

def getRobotHandles(clientID):
    # Motor handles
    _,lmh = sim.simxGetObjectHandle(clientID, 'Pioneer_p3dx_leftMotor',
                                    sim.simx_opmode_blocking)
    _,rmh = sim.simxGetObjectHandle(clientID, 'Pioneer_p3dx_rightMotor',
                                    sim.simx_opmode_blocking)

    # Sonar handles
    str = 'Pioneer_p3dx_ultrasonicSensor%d'
    sonar = [0] * 16
    for i in range(16):
        _,h = sim.simxGetObjectHandle(clientID, str % (i+1),
                                      sim.simx_opmode_blocking)
        sonar[i] = h
        sim.simxReadProximitySensor(clientID, h, sim.simx_opmode_streaming)

    # Camera handles
    _,cam = sim.simxGetObjectHandle(clientID, 'Pioneer_p3dx_camera',
                                    sim.simx_opmode_oneshot_wait)
    sim.simxGetVisionSensorImage(clientID, cam, 0, sim.simx_opmode_streaming)
    sim.simxReadVisionSensor(clientID, cam, sim.simx_opmode_streaming)

    return [lmh, rmh], sonar, cam

# ------------------------------------------------------------------
```

```
def setSpeed(clientID, hRobot, lspeed, rspeed):
    sim.simxSetJointTargetVelocity(clientID, hRobot[0][0], lspeed,
                                   sim.simx_opmode_oneshot)
    sim.simxSetJointTargetVelocity(clientID, hRobot[0][1], rspeed,
                                   sim.simx_opmode_oneshot)

# ------------------------------------------------------------------

def getSonar(clientID, hRobot):
    r = [1.0] * 16
    for i in range(16):
        handle = hRobot[1][i]
        e,s,p,_,_ = sim.simxReadProximitySensor(clientID, handle,
                                        sim.simx_opmode_buffer)
        if e == sim.simx_return_ok and s:
            r[i] = math.sqrt(p[0]*p[0]+p[1]*p[1]+p[2]*p[2])

    return r

# ------------------------------------------------------------------

# def getImage(clientID, hRobot):
#     img = []
#     err,r,i = sim.simxGetVisionSensorImage(clientID, hRobot[2], 0,
#                                            sim.simx_opmode_buffer)

#     if err == sim.simx_return_ok:
#         img = np.array(i, dtype=np.uint8)
#         img.resize([r[1],r[0],3])
```

```
#         img = np.flipud(img)
#         img = cv.cvtColor(img, cv.COLOR_RGB2BGR)

#     return err, img

# ------------------------------------------------------------------

def getImageBlob(clientID, hRobot):
    rc,ds,pk = sim.simxReadVisionSensor(clientID, hRobot[2],
                                        sim.simx_opmode_buffer)
    blobs = 0
    coord = []
    if rc == sim.simx_return_ok and pk[1][0]:
        blobs = int(pk[1][0])
        offset = int(pk[1][1])
        for i in range(blobs):
            coord.append(pk[1][4+offset*i])
            coord.append(pk[1][5+offset*i])

    return blobs, coord

# ------------------------------------------------------------------

def avoid(sonar, direction, integral, P_anterior, blobs, error):
    Kp = 0.5
    Ki = 0.3
    Kd = 0.1
    velocidad = 1
    steer = 0
    minIdx = sonar.index(min(sonar))
```

```
P = 0


if blobs:
    if (sonar[3] > 0.12 and error < 0) or (sonar[4] > 0.12 and error > 0):
        sensor_loc=[-4,-3,-2,-1,1,2,3,4]


        if (direction == 0):
            for i in range(0,8):
                P += sensor_loc[i] * sonar[i]
            D = P - P_anterior
            U = Kp * P - Ki * integral + Kd * D
        else:
            for i in range(0,8):
                P+=sensor_loc[i] * sonar[i+8]
            D = P - P_anterior
            U = Kp * P + Ki * integral + Kd * D
            velocidad = -1 * velocidad


        lspeed, rspeed = velocidad + U, velocidad - U
    else:
        lspeed = -2
        rspeed = -2



else:
    if minIdx in [1, 2, 3, 9, 10, 11]:
        steer = math.pi * (-1) ** direction
    elif minIdx in [4, 5, 6, 12, 13, 14]:
        steer = -math.pi * (-1) ** direction
    else:
```

```
        steer = (sonar[15] + sonar[7] - sonar[0] - sonar[8]) * math.pi * (-1) ** direction
        if sonar[15] + sonar[0] > 0.8 and min(sonar[7:9]) < 0.5:
            steer = -math.pi / 2
        elif sonar[7] + sonar[8] > 0.8 and min([sonar[15],sonar[0]]) < 0.5:
            steer = math.pi / 2
        elif (direction == 0 and sum(sonar[2:6]) < 2) or (direction == 1 and
sum(sonar[10:14]) < 2):
            if steer > 0:
                steer = math.pi
            else:
                steer = -math.pi


    if direction == 0:
        v = min(sonar[2:6])
        if v < .1:
            v = min(sonar[10:14])
            direction = 1

    else:
        v = min(sonar[10:14])
        if v < .1:
            v = min(sonar[2:6])
            direction = 0

    v *= 4

    lspeed = (v + Kp * steer) * (-1) ** direction
    rspeed = (v - Kp * steer) * (-1) ** direction
```

```
    return lspeed, rspeed, direction, integral, P

# ----------------------------------------------------------------------

def main():
    print('### Program started')

    print('### Number of arguments:', len(sys.argv), 'arguments.')
    print('### Argument List:', str(sys.argv))

    sim.simxFinish(-1) # just in case, close all opened connections

    port = int(sys.argv[1])
    clientID = sim.simxStart('127.0.0.1', port, True, True, 2000, 5)

    if clientID == -1:
        print('### Failed connecting to remote API server')

    else:
        print('### Connected to remote API server')
        hRobot = getRobotHandles(clientID)

        # Definimos 4 estados posibles
        AVOID = 0
        SEARCH = 1 # Busca la pelota. Gira 360º y después avanza buscando la pelota
        APPROACH = 2 # Acercarse a la pelota hasta que el sonar la detecte cerca
        ESCORT = 3 # Seguir a la pelota sin acercarse ni alejarse demasiado

        FWD = 0
        BWD = 1
```

```
# Parámetros obtenidos de manera experimental:
steps_360 = 120
Kc = 3.5
Pc = 10


# Inicialización
cur_state = SEARCH
step_count = 0
error = 0
integral = 0
d = 0
prevError = 0
direction = FWD
new_state = SEARCH

PIDKp = 0.6 * Kc
PIDKi = 2 * PIDKp / Pc
PIDKd = PIDKp * Pc / 8


while sim.simxGetConnectionId(clientID) != -1:


    # Perception
    sonar = getSonar(clientID, hRobot)
    # print '### s', sonar
    #print(sonar)
```

```
            blobs, coord = getImageBlob(clientID, hRobot)
            #print('###  ', blobs, coord)

            if blobs:
                error = (coord[0] - 0.5) # Paso coord x de [0, 1] a [-1, 1]

            # Update internal state
            if min(sonar) < .3 and not (blobs and ((sonar[3] > 0.12 and error < 0) or
(sonar[4] > 0.12 and error > 0))):
                if new_state != AVOID:
                    new_state = AVOID
                else:
                    integral = 0
            else:
                if blobs:
                    if error < 0:
                        if sonar[3] < 1:
                            new_state = ESCORT
                            direction = FWD
                        else:
                            new_state = APPROACH
                            direction = FWD
                    else:
                        if sonar[4] < 1:
                            new_state = ESCORT
                            direction = FWD
                        else:
                            new_state = APPROACH
                            direction = FWD
                else:
```

```
            new_state = SEARCH

        if new_state != cur_state:
            step_count = 0
            integral = 0
            d = 0
            cur_state = new_state

    if new_state == AVOID:
        lspeed, rspeed, direction, integral, d = avoid(sonar, direction, integral, d,
blobs, error)


    elif cur_state == SEARCH:
        # En primer lugar giramos sin movernos del sitio buscando la pelota y
también cada 200 pasos hacemos un giro
        if step_count % 200 < steps_360:
            # Pelota a la derecha
            if error >= 0:
                lspeed = 1
                rspeed = -1
            else:
                lspeed = -1
                rspeed = 1
        else:
            lspeed = 4 * (-1) ** direction
            rspeed = 4 * (-1) ** direction

        step_count += 1
```

```
    else:
        prevError = error

        PIDtv = PIDKp * error + PIDKi * integral + PIDKd * (error - prevError)
        tv = max(-1, min(1, PIDtv)) # Limit the command that will be sent

        if cur_state == APPROACH:
            if abs(error) < 0.1:
                fv = 1.5 # Máxima velocidad
            else:
                fv = abs(1.5 * error)
        else: # ESCORTING
            fv = 1.2



        # Integrator anti-windup (clamping method)
        if PIDtv == tv or PIDtv * error < 0:
            integral += error

        lspeed = tv + fv
        rspeed = fv - tv




    # Action
    setSpeed(clientID, hRobot, lspeed, rspeed)
    time.sleep(0.1)

print('### Finishing...')
```

```
        sim.simxFinish(clientID)

    print('### Program ended')

# ----------------------------------------------------------------------

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