Universidad de Salamanca | Grado en Ingeniería Informática

i-Robot Roomba

Algoritmo de detección de caminos y simulación

López Sánchez, Javier
Mateos Pedraza, Alejandro

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Abstract: This essay is intended to explain the programming and simulation of a cleaning robot in terms of detection and tracking of routes.

Keywords: iRobot, Roomba, Roomba Create, Python

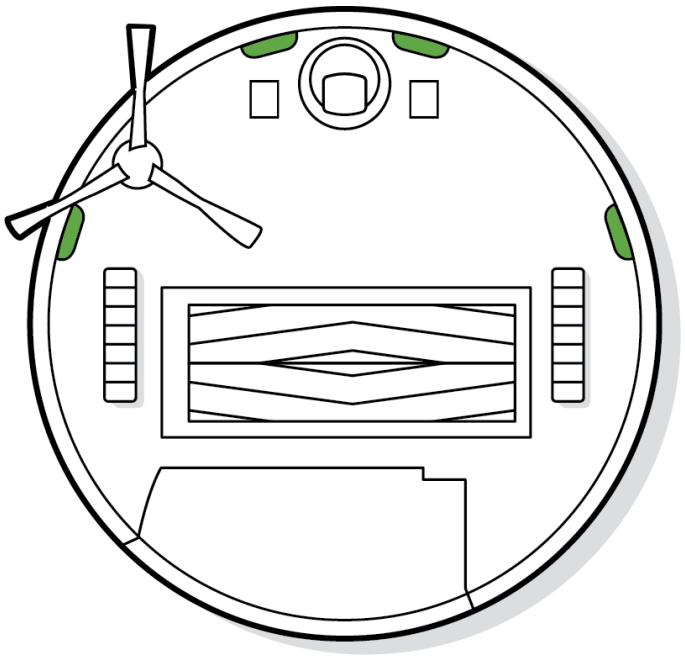
1. Explicación del algoritmo implementado

1.1. Nota preliminar

De entre los algoritmos propuestos, se ha decidido implementar, tal y como se ha presentado en el título, el algoritmo de búsqueda y seguimiento de caminos. La solución propuesta se basa en tres aspectos o procesos que se repiten en bucle infinito hasta la finalización de la ejecución de la simulación: búsqueda del camino, realización del recorrido y actualización de la odometría. Es importante notar que, aunque resulta indiferente hacer la implementación por señales de control de los sensores Cliff o en su defecto por valor lógico, se ha tomado la primera alternativa, donde, según la especificación, encontramos las siguientes correspondencias en cuanto a los valores devueltos por los sensores:

- Valor 0: el sensor detecta una superficie de color negro.
- Valor 4094 o superior (en ocasiones, 4095): el sensor detecta una superficie blanca.
- Valor 1000: el sensor detecta una superficie de color gris. Dada las características de nuestro escenario, este valor carece de interés, si bien conviene tenerlo en cuenta para futuras aplicaciones.

Sobre los sensores Cliff previamente citados, es preciso tener presente cuántos tenemos y en qué posición de la estructura se ubican:



Según se observa en la imagen anexa - que representa la cara inferior del equipo, vamos a contar con cuatro sensores Cliff, dos en la parte delantera y otros dos a cada lado del robot. El valor de las señales devueltas por cada sensor se puede recoger haciendo uso de las directivas:

a. CLIFF_FRONT_RIGHT_SIGNAL (sensor delantero-derecho)
b. CLIFF_FRONT_LEFT_SIGNAL (sensor delantero-izquierdo),
c. CLIFF_RIGHT (sensor derecho)
d. CLIFF_LEFT (sensor izquierdo)

En función del valor que devuelvan estos sensores se especifican unas acciones concretas a realizar, que se tratarán en los siguientes apartados.

1.2. Búsqueda del camino a seguir: findLine()

```
#Rotate until you detect the black line
def findLine():
    global aux
    aux += 1

    cliff_front_right_signal = robot.getSensor("CLIFF_FRONT_RIGHT_SIGNAL")
    cliff_front_left_signal = robot.getSensor("CLIFF_FRONT_LEFT_SIGNAL")
    cliff_right_signal = robot.getSensor("CLIFF_RIGHT_SIGNAL")
    cliff_left_signal = robot.getSensor("CLIFF_LEFT_SIGNAL")

    if aux < 36 and cliff_front_right_signal != 0 and cliff_front_left_signal != 0 and cliff_right_signal != 0 and cliff_left_signal != 0:
        robot.go(0,10)
        time.sleep(1)
        findLine()

    elif aux >= 36:
        if cliff_front_right_signal != 0 and cliff_front_left_signal != 0 and cliff_right_signal != 0 and cliff_left_signal != 0:
            robot.go(10,0)
            time.sleep(1)
            findLine()
        else:
            aux = 0
```

La función findLine() será el método inicial que utilizaremos para que el robot realice una exploración inicial del entorno en el que se encuentra. La idea que planteamos es que el robot comienza a girar sobre sí mismo (mediante llamadas recursivas a la función) hasta que alguno de los sensores dé un aviso de: "superficie negra encontrada" o bien hasta haber dado un giro completo de 360º. Llegado ese último caso, solo puede significar que el robot se encuentra *«en tierra de nadie»* (bajo una superficie completamente blanca), por lo que debe avanzar hasta hallar una superficie negra que pueda recorrer. Para ello, la opción óptima en términos de conseguir un balance entre velocidad y detección es comprobar a cada paso si el estado de algún sensor ha cambiado. Se probó una configuración en la cual el robot daba otra vuelta completa de 360º con cada nuevo paso que, si bien nos garantiza completamente que el robot hallará el camino, resultaba computacionalmente demasiado pesada y, por lo tanto, lenta. Deberemos suponer que a medida que avanza en línea recta en algún momento hallará el camino. También habremos de suponer que el robot no encontrará una pared o un obstáculo antes de llegar al camino, pues ese algoritmo excede al propuesto. Una vez encuentre la superficie negra que va buscando, la ejecución del algoritmo prosigue con la llamada a la siguiente función.

1.3. Siguiendo el camino: followTheWay()

```
#Follow the way
def followTheWay():
    cliff_front_right_signal = robot.getSensor("CLIFF_FRONT_RIGHT_SIGNAL")
    cliff_front_left_signal = robot.getSensor("CLIFF_FRONT_LEFT_SIGNAL")
    cliff_right_signal = robot.getSensor("CLIFF_RIGHT_SIGNAL")
    cliff_left_signal = robot.getSensor("CLIFF_LEFT_SIGNAL")

    if (cliff_front_right_signal == 0):
        robot.go(10,10)
    elif (cliff_front_left_signal == 0):
        robot.go(10,10)
    elif (cliff_right_signal == 0):
        robot.go(0,10)
    elif (cliff_left_signal == 0):
        robot.go(0,10)
    else:
        findLine()
```

Una vez encontrado el camino, el siguiente paso es recorrerlo. La manera de hacerlo es bastante intuitiva, y se divide según las características de cada sensor. En caso de que sea un sensor lateral (izquierdo o derecho) el que ha detectado la línea negra, se le obliga a seguir rotando, con objeto de que alguno de los sensores frontales sean los que despierten, en cuyo caso significa efectivamente que tenemos el robot orientado en la ruta a seguir. Una vez que un sensor delantero nos devuelve la detección de negro, se hace avanzar ligeramente al robot y se le pone un determinado ángulo de giro sacándolo de la trayectoria con el fin de que no se de la vuelta -o tratando de minimizar este efecto a los mínimos casos posibles- en la medida de su recorrido. Tal y como se han configurado los ángulos, puede notarse que la lógica del robot está ligeramente sesgada en cuanto a la realización del recorrido exclusivamente en sentido antihorario, pero como apenas se solicitaba seguir el recorrido, se consideró una solución válida. En caso de que el robot esté orientado en sentido horario, las pruebas demuestran que finalmente se reorienta y termina haciendo el trayecto en el sentido contrario. Esto lo hace, nuevamente mediante llamadas a la función recursiva findLine(), explicada anteriormente.

1.4. Actualización automática y odometría: DynamicUpdate()

```
def __call__(self):
    global xdata
    global ydata
    global x
    global y
    global xIncrement
    global yIncrement
    global last_angle
    global xfinal
    global yfinal
    global aux

    self.on_launch()

    while 1:
        angle = robot.getSensor("ANGLE") * pi/180
        distance = robot.getSensor("DISTANCE") /1000

        xIncrement = distance * np.cos(last_angle + angle)
        yIncrement = distance * np.sin(last_angle + angle)

        last_angle += angle
        last_angle_degrees = last_angle*180/pi

        if last_angle_degrees >= 360:
            last_angle_degrees = last_angle_degrees - 360
            last_angle = last_angle - (360*pi/180)

        xfinal = xdata[-1] + xIncrement
        yfinal = ydata[-1] + yIncrement

        xdata.append(xfinal)
        ydata.append(yfinal)

        self.on_running(xdata, ydata)

        aux = 0
        findLine()
        followTheWay()

        time.sleep(1)
    return xdata, ydata
```

El método más interesante de esta función de actualización automática es el que se ejecuta automáticamente al instanciar la clase en cuestión. Es aquí donde calcularemos primero la odometría (con intervalos de tiempo de 1 segundo para asegurar la correcta medición de los diferentes parámetros -un tiempo de estudio muy pequeño hace que los sensores devuelvan valores nulos o cero que carecen de significado-) y después llamaremos a los métodos de cálculo de rutas anteriores. Para el cálculo de la odometría, simplemente obtendremos los valores de ángulo y distancia -con las directivas ANGLE y DISTANCE-, que convertimos a las unidades correspondientes, y realizamos los cálculos que se exponen en la teoría. Los resultados se acumulan en diferentes vectores para ser graficados mediante las funciones ofrecidas en el propio enunciado del proyecto.

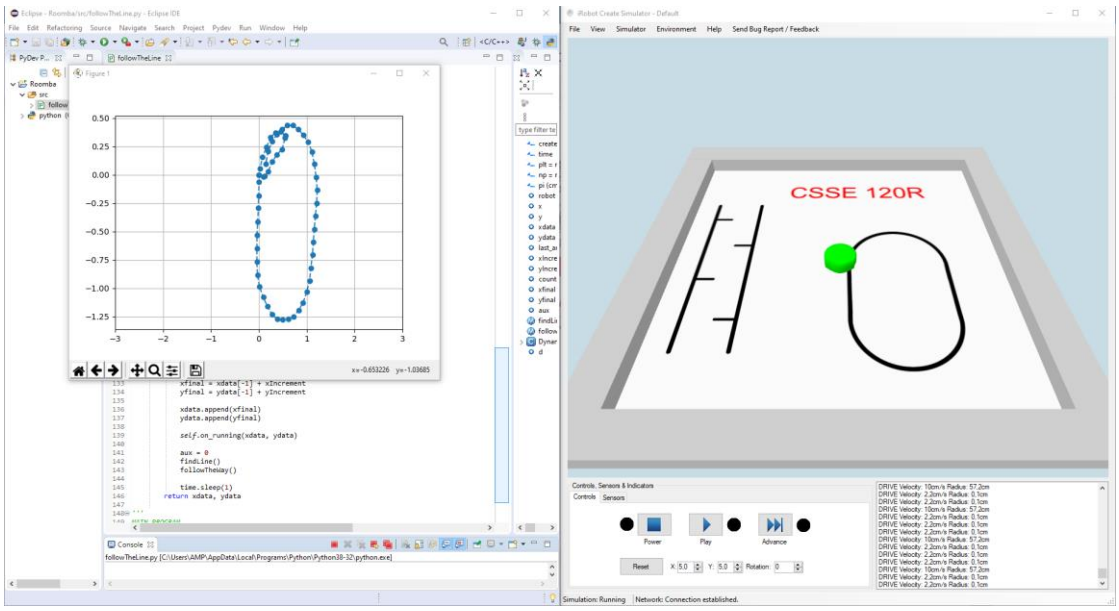


Ilustración 1. Simulación y cálculo odométrico para una configuración inicial (x, y, theta) = (5,5,0)