01 de marzo de 2019

Entrega 01: Control

Robótica y Percepción Computacional

Alumnos:

Luciano García Giordano (150245)
Gonzalo Flórez Arias (150048)
Salvador González Gerpe (150044)

Universidad Politécnica de Madrid
ETSI Informáticos

Índice

1. Descripción del programa

Partiendo del programa Brain base del robot proporcionado por el profesorado de la asignatura, se han implementado una serie de cambios y mejoras, teniendo en mente 3 objetivos principales: lograr que el robot encuentre inicialmente la línea y sea capaz de recuperarla en caso de perderla; asegurar que el recorrido y velocidad del robot sobre la línea sean máximos; permitir que el robot esquive obstáculos mediante el uso de sus sónares (asegurando que deje y retome la línea correctamente al encontrarse con un obstáculo).

Por un lado, se planteó la necesidad de procurar que el robot encuentre la línea inicialmente. En este sentido, aunque es probable que se puedan encontrar soluciones mejores, nuestro código se limita a avanzar hacia adelante hasta que encuentre la línea, tras lo cual comienza el acercamiento progresivo (explicado en detalle en el siguiente apartado. Por otro lado, una vez que el robot pierda la línea, irá hacia atrás. En este sentido, nos dimos cuenta inicialmente de que a veces al volver atrás estaba girando demasiado, y le costaba volver a la línea de la que se escapó. Por tanto, está implementada una marcha atrás similar al aparcar en paralelo con un coche: si se ha salido hacia la izquierda de la línea, comienza dando marcha atrás hacia la derecha, para posteriormente dar marcha atrás a la izquierda y así reorientarse para estar lo más cerca posible de estar posicionado sobre la línea, con lo que pueda reanudar su marcha sin inconvenientes. De esa manera, volvemos recorriendo aquello que se asemejaría a una "S", de forma que el robot siempre encuentra la línea tras perderla. En curvas muy cerradas, eso es especialmente útil, aunque puede causar que el robot cambie el sentido de la marcha tras "encontrar la línea" por la que venía.

Respecto al propio código que afecta al acercamiento a la línea y su recorrido sobre ella, inicialmente se trató de implementar los distintos Kp, Kd y Ki vistos en clase, para luego entender qué valores para los mismos serían los idóneos para lograr que el recorrido sea óptimo. A este respecto, consideramos óptimo que cuanto más lejos esté de a línea, más rápida y pronunciadamente se acerque. Cuanto más se acerca, su velocidad y giro debe ir reduciéndose, ya que no es ideal que sobrepase la línea demasiado, pues acabaría haciendo simplemente zigzags en vez de ir lo más recto posible por la línea. Tras ir probando distintas opciones, pensamos que podríamos realizar un modelo cuadrático en Kd, pues parece tener sentido generar, a partir de un par de parámetros, una parábola que modelice bien la velocidad de movimiento según la distancia a la línea. Kp se queda haciendo referencia al ángulo de giro. Para poder encontrar los parámetros, se usó el software Geogebra, en el cuál generamos una parábola y mediante los parámetros .ªz "b"(luego introducidos en el código), fuimos probando hasta encontrar una curva que nos satisficiese. Hemos probado también el uso de Ki, pero dado que no tenemos un sistema en que la aceleración tiene demasiada importancia pierde el sentido su uso. De esa manera, preferimos no usarlo, ya que sería una complejidad añadida sobre un robot que recorre bien la línea sin necesitar ese parámetro.

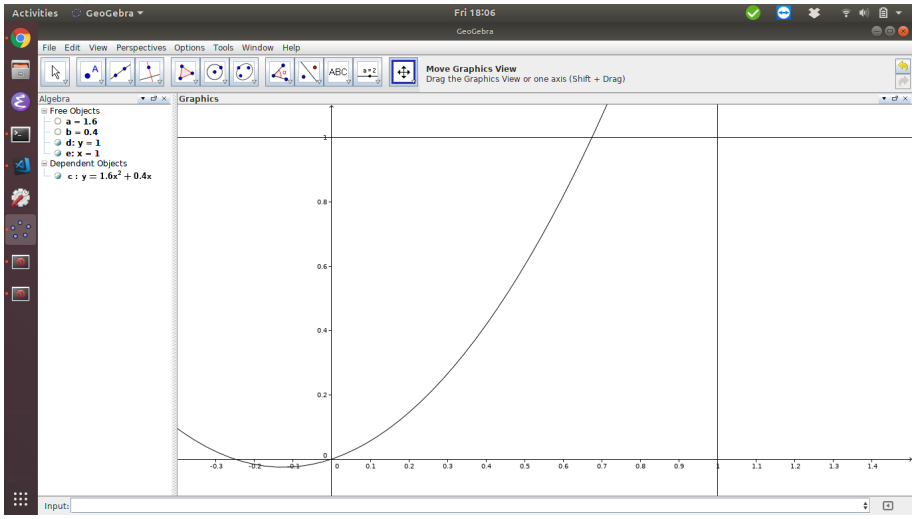

Figura 1: Observamos el modelo cuadrático en el software GeoGebra para determinar los parámetros iniciales para nuestra exploración. Con el simulador, los hemos cambiado según nuestras observaciones para que fuera más adecuado al funcionamiento del robot.

Basándonos tanto en el sistema de seguir paredes que ya desarrollamos para la primera tarea de la asignatura como en el estado de seguir líneas de esta entrega, implementamos la capacidad del robot de emplear sus sónares para encontrar y esquivar obstáculos que se interpongan en su camino. En este caso, calculamos la distancia respecto al objeto de tal manera que es positiva si está más lejos de cierta distancia d del objeto, y negativa si está más cerca de lo deseado. De esa manera, tenemos una trayectoria virtual, que podemos recorrer como si fuera una línea, alrededor del objeto. Para poder intentar desviar mejor los obstáculos, implementamos nuestra solución en dos casos posibles: que el obstáculo esté más cerca del lado izquierdo del robot, o del lado derecho. Si se da el primer caso, el robot tratará de virar hacia la derecha, siguiendo a su izquierda el obstáculo cual pared a recorrer, y bordeándola hasta que encuentre de nuevo la línea. Entonces, el robot olvidará su tarea de seguir el obstáculo, y retomará la línea. Si se da el segundo caso comentado el resultado es análogo. Se añadió además un limitador de la velocidad máxima mientras sigue la línea según la distancia de cualquier objeto para asegurar que el robot no avance demasiado antes de decidir que está en ruta de colisión con un objeto. Nuestro objetivo con eso era evitar choques con los obstáculos al virar cuando intenta rodearlo.

1.1. Modelo utilizado

En resúmen, hemos implementado un robot que tiene una serie de estados y transiciones entre estados, organizados de la siguiente manera:

- **initialSearchLine**: avanza hacia delante buscando una línea. Transita a followLine cuando la encuentra. Transita a circleObjectOnRight o a circleObjectOnLeft si encuentra un objeto por delante.
- **searchLine**: intenta encontrar la línea yendo hacia atrás en forma de "S". Transita a followLine cuando la encuentra.
- **approachLine**: sirve para aproximarse a una línea sin que otras variables le afecten. Transita a followLine cuando está cerca de la línea.
- **followLine**: sigue la línea implementando la consigna que lleva K_d y K_p en consideración. Transita a circleObjectOnLeft o a circleObjectOnRight cuando encuentra un obstáculo (dependiendo de por qué lado le encuentra).
- **circleObjectOnRight y circleObjectOnLeft**: circulan el objeto utilizando la consigna descrita anteriormente. Básicamente sique una línea virtual calculada a partir de la distancia al objeto. Transita a leaveObjectOnRight o a leaveObjectOnLeft (respectivamente) cuando encuentra una línea más cerca de 0.3 unidades.
- **leaveObjectOnRight y leaveObjectOnLeft**: dejan de seguir el objeto y hacen una curva en la dirección adecuada para seguir la línea anteriormente encontrada. Transita a approachLine cuando encuentra una línea a menos de 0.95 unidades.

2. Ejemplos ejecutados

Se muestran a continuación los tests con los que se ha probado el robot, incluyendo la traza en rojo del recorrido del mismo sobre el mapa.

Con el background 1 desarrollamos todo el modelo de funcionamiento. Llegamos a una solución para nosotros aceptable, que se muestra en la figura 2. La optimización de los parámetros también se decidieron en su mayoría en este mapa.

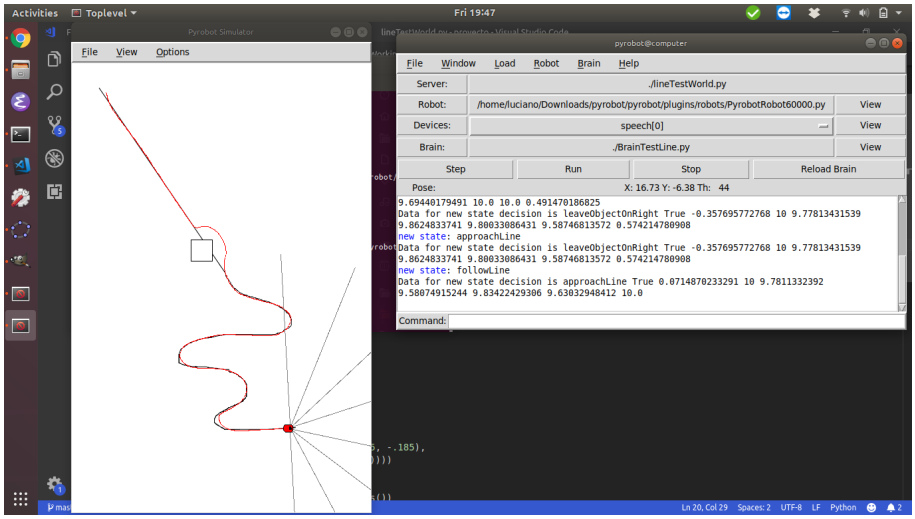

Figura 2: Uno de los resultados con el background 1.

Ya en el background 2, hicimos tanto cambios en los parámetros como desafíos más complejos con los objetos. Pudimos observar que faltaban estados para las salidas de los objetos, y los añadimos y probamos en este y en el anterior backgrounds.

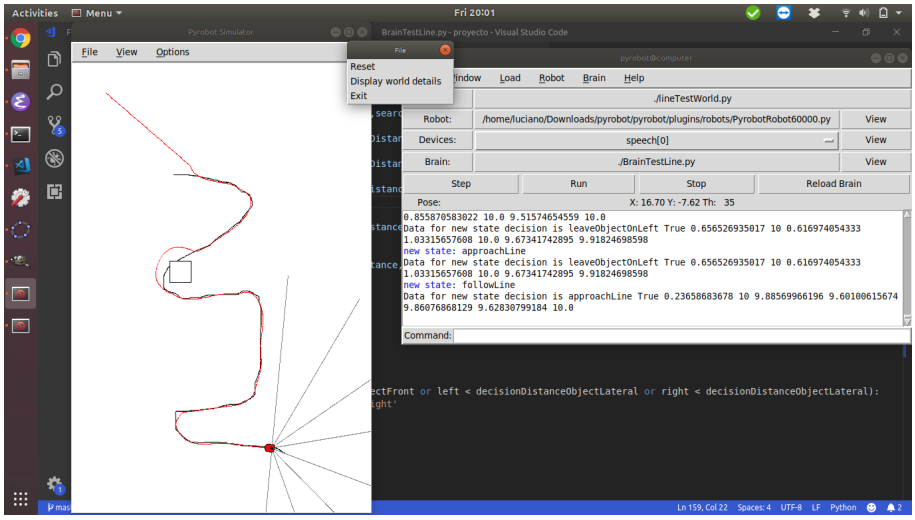

Figura 3: Con el background 2 probamos soluciones más complejas, y con más objetos.

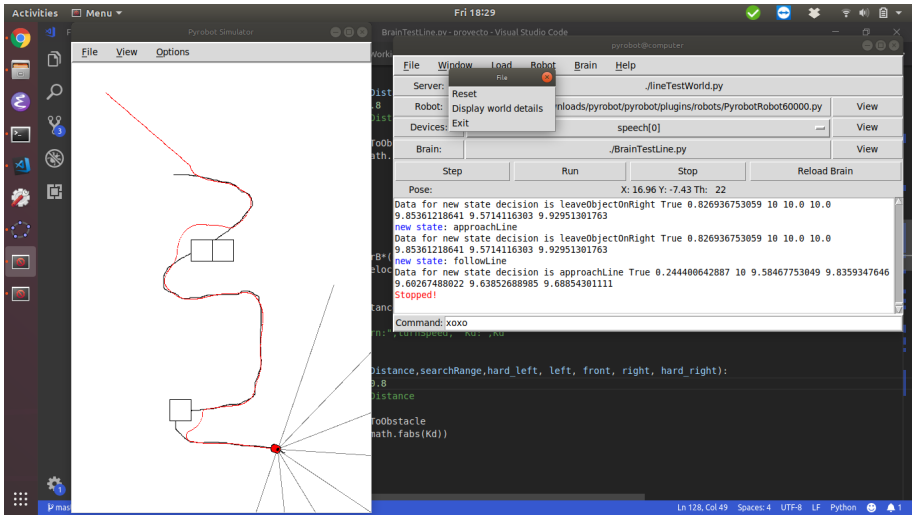

Figura 4: Otro resultado con el background 2.

En el background 3, de invención nuestra, hicimos un circuito cerrado. En diversas ocasiones, conseguimos hacer que el robot hiciera la vuelta entera (incluyendo las cerradas curvas de la derecha). Sin embargo, por un problema conocido del simulador, en muchos casos se queda atascado en un objeto porque el simulador no detecta la línea. En la salida por texto del programa en la figura 5, podemos observar varios "False", que corresponden justamente a no detectar la línea (variable hasLine).

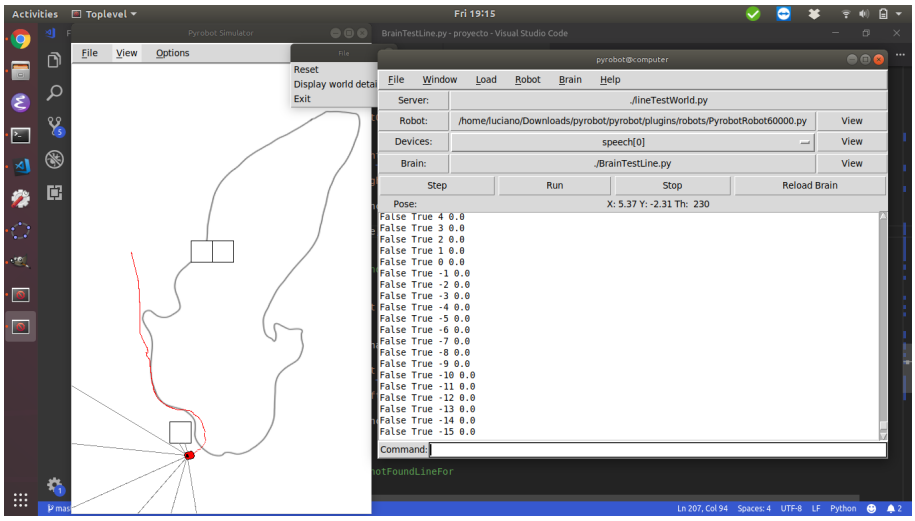

Figura 5: El simulador no detecta bien las líneas cuando están en perpendicular.

Ya en el background 4, quisimos comprobar la capacidad del robot de completar un circuito cerrado. Elegimos para ello el conocido circuito de Hockenheim, llamado Hockenheimring [1]. Pusimos un objeto en la curva más cerrada dado que era demasiado cerrada.

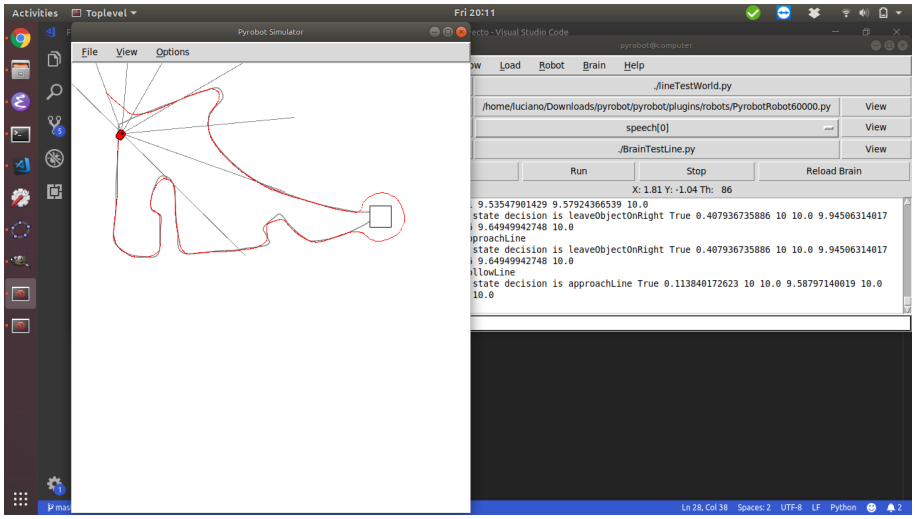

Figura 6: Hockenheimring. El robot es capaz de dar una vuelta completa en el circuito.

Como se puede observar, en algunos casos no se logra retomar la línea tras rodear un obstáculo debido al simulador, por el tema de que al encontrarse una línea perpendicular delante no encuentra correctamente la distancia del centro, y por lo tanto no avisa al robot de que deje de seguir el obstáculo y vuelva a reencontrarse con la línea.

Referencias

[1] Hockenheimring Official Website. (n.d.). Retrieved March 1, 2019, from https://www.hockenheimring.de/en