Laboratorio de Principios de Mecatrónica

16 de mayo de 2021

Proyecto Final

Trayectorias cooperativas

Grupo:

L002

Estudiante:

- Oceguera Urquiza Juan Manuel
- Jáuregui Tapia Jesús Enrique
- Reyes Badilla Ángel Gabriel

Profesor:

Benito Granados–Rojas

Índice

1. Introducción

El sistema de operación robótica (ROS por sus siglas en inglés) es un marco de trabajo para escribir software a robots. Es una colección de herramientas, librerías y convenciones que buscan facilitar la tarea de crear comportamientos robóticos complejos a través de distintas plataformas. Debido a su capacidad de conectarse a distintas fuentes permite el desarrollo colaborativo de software robótico. [1]

Turtlesim es una herramienta de ROS desarrollada por Josh Faust y Dirk Thomas que provee un entorno especializado para la simulación de agentes móviles. En este caso, los agentes móviles se representan por tortugas que se desplazan en un escenario cuadrado. Para desplazar dichos agentes, se envía a la tortuga la velocidad en línea recta y angular. Es común utilizar la plataforma para entender el funcionamiento de ROS y sus distintas librerías (y -posteriormente- aplicar el conocimiento en un entorno físico). [2]

ROS permite la interacción de distintos agentes a través de un sistema central al que se conectan nodos. Después, cada nodo puede publicar y suscribirse a información que pasa por este sistema; en el caso de Turtlesim, los distintos agentes publican sus coordenadas e inclinación. Con estos datos es posible desarrollar estrategias para la conducción de una tortuga evitando chocar con el resto. [3]

Se pretende planificar y ejecutar una trayectoria de intercambio de posición simultánea en Turtlesim evadiendo a las demás tortugas.

2. Experimentos y Simulaciones

2.1. Generación del código

Para el código se decidió tomar como base la última versión del algoritmo “Go To Goal” provisto por ROS. Dicho código funciona a través de un controlador PID: la velocidad lineal se calcula como el producto de una constante por la distancia euclidiana (entre la posición actual y la meta); la velocidad angular como una constante multiplicada por la arcotangente de la división de la distancia vertical entre la distancia horizontal. Por último, se define una tolerancia que indica la separación máxima admisible entre el punto deseado ideal y el punto final de la trayectoria del agente. [4]

Lo anterior nos permite pasar de un punto ’A’ a un punto ’B’ del entorno de Turtlesim; sin embargo, falta resolver el sistema que nos permita evadir a los demás agentes (para no chocar con ellos). Para esto se definieron distintas maniobras evasivas según la proximidad con otras tortugas: en caso de que algún agente se encuentre dentro del semicírculo superior de radio 2 disminuimos la velocidad linear a la mitad; en caso de que algún agente se encuentre dentro del semicírculo superior de radio 1 redirigimos nuestra ruta hacia el punto cercano (a una distancia de 1.5) que nos aleje más del resto de las tortugas (y no choque con ninguna en su trayectoria al nuevo punto).

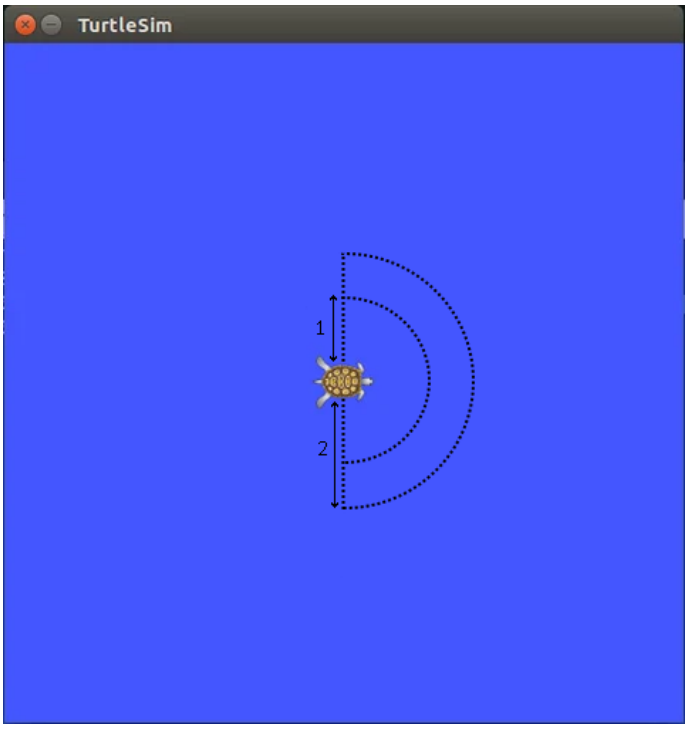

Figura 1: Semicirculos de detección de peligro.

Se decidió utilizar un semicírculo debido a que no utilizamos velocidades negativas (no vamos en reversa), por lo que no corremos el riesgo de chocar con agentes que se encuentren a nuestra espalda. Para lograr esto se calcula el producto punto entre la diferencia de las coordenadas de otra tortuga con nuestra posición actual, con el vector compuesto por el coseno y el seno de nuestra inclinación actual; luego, discriminamos las tortugas con producto punto negativo (dado que están a nuestras espaldas), y nos quedamos con la distancia euclidiana de la tortuga más cercana con producto punto positivo.

$$ProductoPunto = (x_{otro} - x_{propio}) * cos(\theta_{propio}) + (y_{otro} - y_{propio}) * sin(\theta_{propio}) \quad (1)$$

Si dicha distancia se encuentra dentro de alguno de los radios definidos tomamos la acción evasiva correspondiente. Para la selección de la trayectoria óptima (el punto cercano que nos aleje más del resto de los agentes) consideramos 28 trayectorias de radio 1.5 uniformemente distribuidas alrededor de un círculo con centro en las coordenadas actuales; luego, se selecciona la trayectoria que maximice la distancia con el resto de las tortugas y cumpla con las siguientes condiciones: la distancia con la tortuga más cercana sea por lo menos de 1, y la distancia mínima entre la tortuga más cercana y el punto medio de la trayectoria sea, igualmente, de al menos 1 (lo que nos permite asegurar que no chocaremos con otra tortuga al tomar esta trayectoria).

2.2. Simulaciones previas

Primeramente, se realizó una simulación sencilla de dos agentes con nuestro código, a los que se les asignaron trayectorias que se intersectaban en varios puntos (con el objetivo de ver cómo reaccionaban para evadir las colisiones). El resultado fue satisfactorio dado que no chocaron.

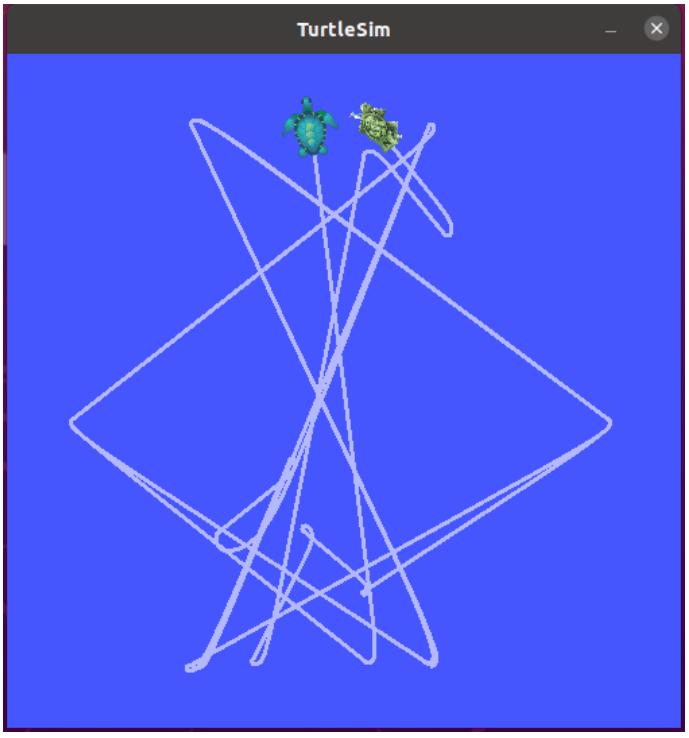

Figura 2: Simulación de 2 tortugas.

Después, se fue agregando una tortuga a la vez (con el mismo código) con las trayectorias que se utilizarían en la demostración final (Figura 3). En ningún caso existieron colisiones.

	x0	y0	x1	y1	x2	y2	x3	y3
T1	1.5	1.0	3.0	10.0	10.0	3.0	7.5	7.0
T2	3.0	1.0	1.5	10.0	10.0	4.0	5.5	9.0
T3	4.5	1.0	6.5	10.0	1.0	2.0	1.5	5.0
T4	6.5	1.0	4.5	10.0	10.0	2.0	9.5	5.0
T5	8.0	1.0	10.0	10.0	1.0	4.0	3.5	7.0
T6	10.0	1.0	8.0	10.0	1.0	3.0	5.5	5.0

Figura 3: Rutas de las tortugas.

A continuación, se muestran algunos resultados de las simulaciones descritas anteriormente; si bien no existieron colisiones, existían momentos en los que las tortugas tardaban mucho tiempo en encontrar una ruta con la que evitarían choques. La velocidad no se considera prioritaria para este proyecto, pero terminar rápido permite evitar posibles situaciones complicadas en las que las tortugas que ya terminaron bloquean el trayecto que se desea tomar.

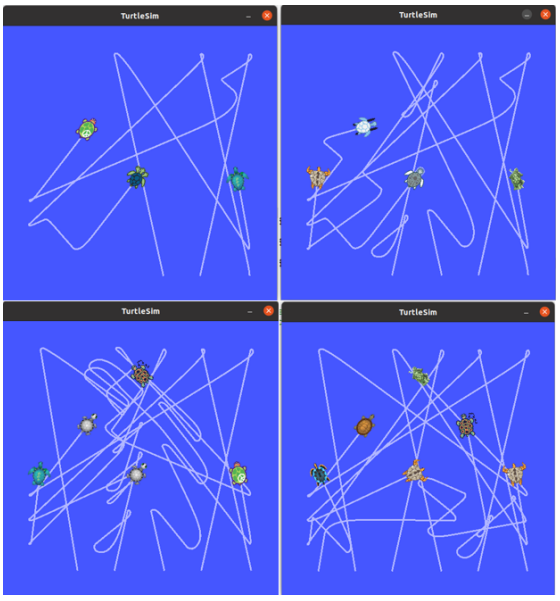

Figura 4: Simulaciones previas.

2.3. Demostración final

Para la demostración cada equipo cargó el código de su tortuga; en nuestro caso fue la tortuga 6 (con la trayectoria definida en la Figura 3). Se realizó varias veces la simulación y nunca chocó nuestra tortuga; además, fue la más rápida en todas las pruebas (lo que nos permitió evitar situaciones complicadas, como la enfrentada por las tortugas en la parte inferior derecha de la Figura 5).

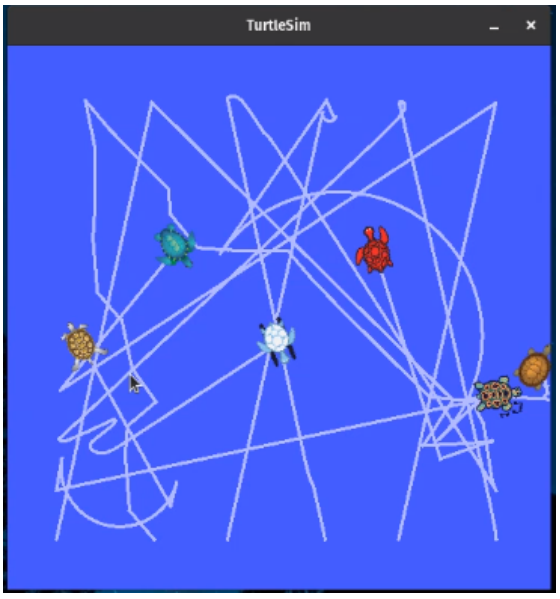

Figura 5: Demostración final.

3. Conclusiones

La implementación de Turtlesim en ROS es sumamente útil, pues nos permite simular las trayectorias de múltiples agentes móviles intercomunicados a través del sistema publicación/suscripción de la plataforma. Esto, además de ayudar a generar una base de conocimientos y destrezas en el uso de plataformas profesionales de diseño y simulación mecatrónica (como ROS), posibilita la simulación de robots de trayectorias cooperativas previo a su construcción en físico.

En cuanto al sistema de evasión de colisiones utilizado, notamos -a través de la comparación con el comportamiento del resto de agentes- que ignorar el semicírculo de la parte trasera de nuestra tortuga fue un gran acierto. Mientras que la mayoría de tortugas bajaban la velocidad si detectaban a otra tortuga cercana a su espalda (debido a que consideraban un círculo completo de seguridad), la nuestra mantenía la velocidad y huía de las situaciones complicadas. Esto nos permitió terminar la trayectoria completa en un menor tiempo que el resto de los agentes, y -en consecuencia- disminuir el número de evasiones necesarias para llegar a la meta.

Al observar el resto de la simulación, era común que otros agentes quedaran bloqueados en situaciones comprometedoras: por ejemplo, en la parte inferior derecha de la Figura 5 se puede observar que una tortuga está bloqueando por completo el punto meta de la otra. Si bien en nuestro caso evadimos estas situaciones por la velocidad con la que recorrimos los puntos, sería importante añadir al código una condición que detecte esta situación y se aleje mucho de la tortuga obstructora para permitirle que continúe su trayecto (nuestro agente sí se alejaría, pero no sabemos si la longitud sería lo suficientemente grande para abandonar el radio de seguridad de la tortuga obstructora).

Por último, sería muy útil contar con una base de nodos de demostraciones anteriores para realizar las simulaciones previas a la demostración final. En nuestro caso, todas las simulaciones previas fueron con tortugas con el mismo código (el nuestro), y funcionaban bien. El problema es que esto reduce de sobremanera la variedad de códigos posibles para la conducción de los agentes; por ejemplo: nos fue evidente que la tortuga 2 tenía un algoritmo de selección de trayectorias muy diferente al resto de las tortugas. Si bien no chocamos con ella, sí notamos que fue la tortuga con la que más nos complicamos. Esto se pudo evitar si hubiéramos tenido algún nodo que actuara de forma similar en las simulaciones previas, pues hubiéramos planificado estrategias para lidiar con este tipo de conducción.

4. Enlaces externos

`https://github.com/JuanOceguera/PrincipiosDeMecatronica`

`https://github.com/Jesus669/PrincipiosMecatronica`

`https://github.com/Angelreyes99`

Referencias

[1] Robot Operating System, About ROS, ROS, 2021. [En línea]. Disponible en: https://www.ros.org/about-ros/. [Visitado el: 23- Abr -2021].

[2] J. Faust and D. Thomas, Turtlesim, ROS Wiki, 2020. [En línea]. Disponible en: http://wiki.ros.org/turtlesim. [Visitado el: 23- Abr -2021].

[3] Granados, B. (2021). Principios de Mecatrónica. Tema: *Robot Operating System*. Instituto Tecnológico Autónomo de México. Ciudad de México. 16 de abril del 2021.

[4] Robot Operating System, Go To Goal, ROS wiki, 2021. [En línea]. Disponible en: http://wiki.ros.org/turtlesim/Tutorials/Go%20to%20Goal. [Visitado el: 14- May -2021].