Taller 1: Robótica

Camilo Andrés Morillo Cervantes (*c.morillo@uniandes.edu.co*), Jeremy Mateo Hernández Jaramillo (*jm.hernandezj1@uniandes.edu.co*), Juan David Riveros Pérez (*j.riverosp@uniandes.edu.co*), David Santiago Rincón Reyes (*d.rinconr@uniandes.edu.co*),
Departamento de Ingeniería Eléctrica y Electrónica
Universidad de Los Andes

I. Resultados

*I-A. **Control del robot.***

El objetivo de este punto fue el de construir un nodo que permitiera controlar el robot TurtleBot2 simulado en CoppeliaSim utilizando el teclado de la computadora. Este nodo se llamo `/turtle_bot_teleop`.

*I-A1. **Creación del nodo turtle_bot_teleop**:* Para desarrollar el nodo se utilizo el lenguaje de programación de Pyhton. Las librerías utilizadas fueron:

- ***curses:*** Esta librería permitió tomar las entradas del teclado. Las teclas que se utilizaron para los cuatro movimientos fueron *w*, *a*, *s*, *d*, siendo *w* y *s* para manejar el robot hacia adelante y hacia atrás, mientras *a* y *a* son para rotar el robot hacia los lados.
- ***time:*** Permite generar *delays* en la ejecución del código con tal de crear tiempos de espera segun fuera necesario.
- ***argparse:*** Permite ingresar los valores de la velocidad lineal y angular por consola, **al momento de iniciar el nodo**. Por ejemplo, si el usuario quiere que la velocidad lineal del robot sea de 50 y la angular 30, se debe ingresar por consola lo siguiente al iniciar el nodo: `ros2 run turtle_bot turtle_bot_teleop -l 50 -a 30`.
- ***geometry_msgs.msg:*** Esta librería permitió definir el tipo de mensaje que se envía al tópico correspondiente mediante la arquitectura subscriptor-publicador. En este caso se utilizo un mensaje tipo `Twist` que permite definir las velocidades lineales y angulares en los ejes x, y, z.

Además se contó con ayuda de la librería `rclpy` siguiendo las convenciones de ROS 2 para Python. En el método `__init__` se realiza la configuración e inicialización necesaria:

- Inicializar nodo padre.
- Crear publicador en el tópico `/turtlebot_cmdVel` para mensajes `geometry_msgs/Twist`.
- Imprimir en el log instrucciones de uso de teclas.
- Crear timers para solicitar teclas y publicar velocidades.

También se tienen métodos `wrapper` y `set_cmdVel` que contienen la lógica de solicitar velocidades iniciales al usuario y publicar los mensajes de velocidad en cada iteración, respectivamente.

*I-A2. **Funcionamiento del nodo**:* Al ejecutarse, el nodo le solicita al usuario establecer una velocidad lineal y una velocidad angular para el robot simulado. Luego, en cada iteración se solicita la tecla presionada por el usuario y según esta se publica en el tópico `/turtlebot_cmdVel` los valores de velocidad apropiados dentro de los límites ingresados inicialmente.

Las teclas se asignaron de la siguiente forma:

- `w`: velocidad lineal positiva.
- `s`: velocidad lineal negativa.
- `a`: velocidad angular negativa.
- `d`: velocidad angular positiva.
- Sin tecla: velocidades en cero.

De este modo se logra un control preciso del TurtleBot2 con la computadora. El nodo funciona de forma estable y cumple con las especificaciones solicitadas para esta funcionalidad. En la figura 1 se nuestra la ejecución del nodo `/turtle_bot_teleop` donde se observa los mensajes de inicio del nodo y las correspondientes instrucciones para manejar el robot mediante este nodo.

```
camilo@camilolinux:~$ cd ros2_ws/
camilo@camilolinux:~/ros2_ws$ ros2 run turtle_bot turtle_bot_teleop
[INFO] [1709264854.099917369] [turtle_bot_teleop]: turtle bot teleop started
[INFO] [1709264854.100222842] [turtle_bot_teleop]: Use A for left, D for right,
W for up and S for down
```

Figura 1. Inicio del nodo para el control del robot (`turtle_bot_teleop`)

En la figura 2 se muestra el correspondiente grafo generado por `rqt_graph` cuando se esta ejecutando el nodo `turtle_bot_teleop`. Aquí se observa como el nodo `turtle_bot_teleop` publica en el tópico `turtlebot_cmdVel` al cual esta suscrito el nodo `sim_ros2_interface` el cual corresponde a la simulación de CoppeliaSim.

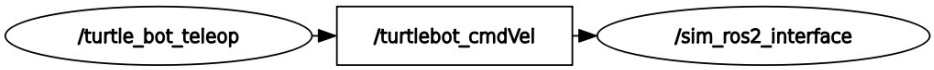

Figura 2. Grafo para el control del robot (`turtle_bot_teleop`)

I-B. Visualización en tiempo real de la posición del robot.

Inicialmente, se creo un nodo llamado `/turtle_bot_interface` que permitía mostrar la posición en tiempo real del *Turtlebot2*. Sin embargo, dado que este nodo también debía poder almacenar el recorrido del robot y más adelante se debía comunicar con otro nodo para reproducir una ruta guardada haciendo uso de la arquitectura cliente servidor, se convirtió entonces en el nodo central. Esto quiere decir que a partir de este nodo se pueden acceder a las funcionalidades de los demás nodos. En relación al almacenamiento del recorrido del robot, esto se realizo de dos formas: guardando la trayectoria en un archivo de texto y guardando la trayectoria en una imagen.

I-B1. ***Creación del nodo turtle_bot_interface:*** Este nodo es el que utiliza la mayor cantidad de librerías dado que presenta múltiples funcionalidades. Al igual que todos los demás nodos desarrollados, este se realizo en Python con las siguientes librerías:

- **tkinter:** Se utiliza para todo lo relacionado con la interfaz gráfica e interacción con el usuario permitiendo establecer botones, entradas de texto y demás con el objetivo de interactuar con el usuario.
- **matplotlib:** Esta librería se utiliza principalmente para dibujar la trayectoria del robot y poder guardarla posteriormente como una imagen. Además, esta librería permite crear la ventana donde se gráfica la trayectoria y poder introducirla dentro de la ventana de la interfaz gráfica creada con *tkinter*.
- **threading:** Esta librería ayudo a crear múltiples hilos de forma que se pueda tener una solución concurrente. Se aplico de forma que se tuvieran hilos independientes para graficar la posición del robot, para guardar el archivo y para llamar al servicio de reproducir el recorrido del robot.
- **os:** Esta librería se utiliza para crear terminales y ejecutar comandos en esta, en este caso se utilizo para crear una nueva terminal y ejecutar los nodos del control del robot y del servicio para reproducir la ruta del robot.
- **geometry_msgs.msg:** Define el tipo de mensaje que se publica en el tópico de la velocidad del robot siendo este de tipo `Twist` de forma que se pueda guardar dichas velocidades mediante la funcionalidad de guardar el recorrido del robot.
- **nav_msgs.srv:** Define los mensajes tipo `LoadMap` de forma que se tiene un estándar para los mensajes que se envían al servicio encargado de reproducir el recorrido del robot. Este tipo de mensajes `LoadMap` contienen el url del archivo que se pretende reproducir.

Este nodo, al ser el nodo central de toda la solución, contiene múltiples métodos y clases que permiten manejar todo de la forma mas fácil posible. Principalmente se tienen dos clases `gui_class` y `turtle_bot_interface` los cuales corresponden a la interfaz gráfica y al nodo de la interfaz respectivamente. La ejecución comienza creando un objeto de la clase `gui_class` el cual en su construcción se encarga de crear todos los gráficos de la interfaz (botones, texto, etc.) y además de crear un *thread* encargado de la ejecución del nodo. Mediante los eventos generados al presionar un botón se procede a ejecutar la respectiva funcionalidad dando así una interfaz completamente funcional con todas las características requeridas.

I-B2. ***Funcionamiento del nodo****:* Tal como se menciono anteriormente, el funcionamiento del nodo se presenta en la figura 3 donde se muestran todos los botones y campos de texto correspondientes a las funcionalidades planteadas. Aquí se observa, en primer lugar, el botón correspondiente al graficador de la posición del robot. Posteriormente, se observa el campo de texto y botones para guardar la ruta. Luego, se tiene el botón para llamar al servicio del seguidor de ruta y finalmente dos botones para iniciar los nodos para controlar el robot y el servicio del seguidor de ruta.

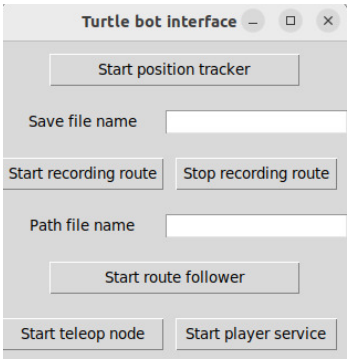

Figura 3. Interfaz desplegada por el nodo `turtle_bot_interface`

En la figura 4 se observa el dibujo de la posición del robot y como se va mostrando en tiempo real el movimiento que sigue el robot. Aquí se observa que la gráfica generada por *matplotlib* presenta coordenadas desde -250 a 250 tanto para el eje vertical como el horizontal con el punto central $[0, 0]$ siendo la posición inicial por defecto del robot.

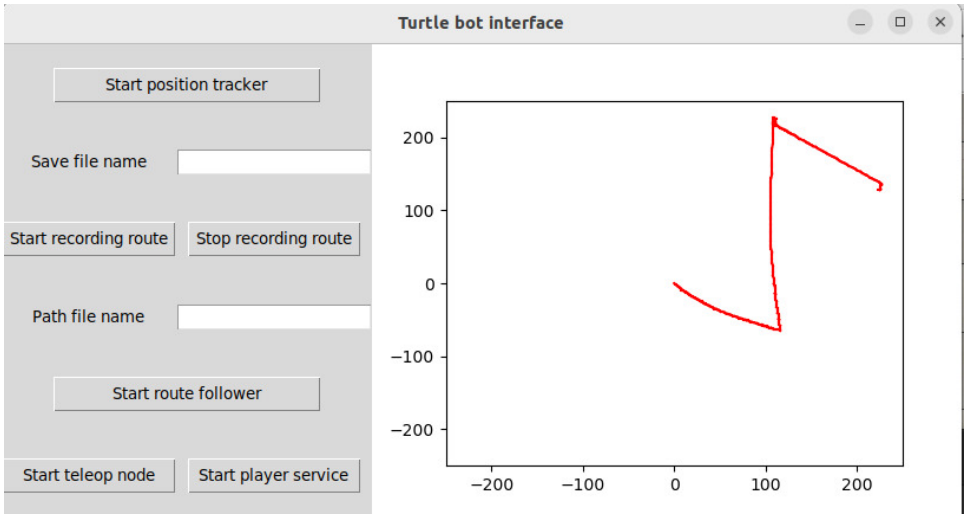

Figura 4. Seguimiento de la posición del robot desplegada por el nodo `turtle_bot_interface`.

Cuando se presiona nuevamente el botón correspondiente al graficador de la posición del robot se muestra la opción para guardar la figura tal como se presenta en la figura 5. En esta nueva ventana se puede ingresar el nombre y ubicación donde se guardará la figura o directamente no guardar la figura.

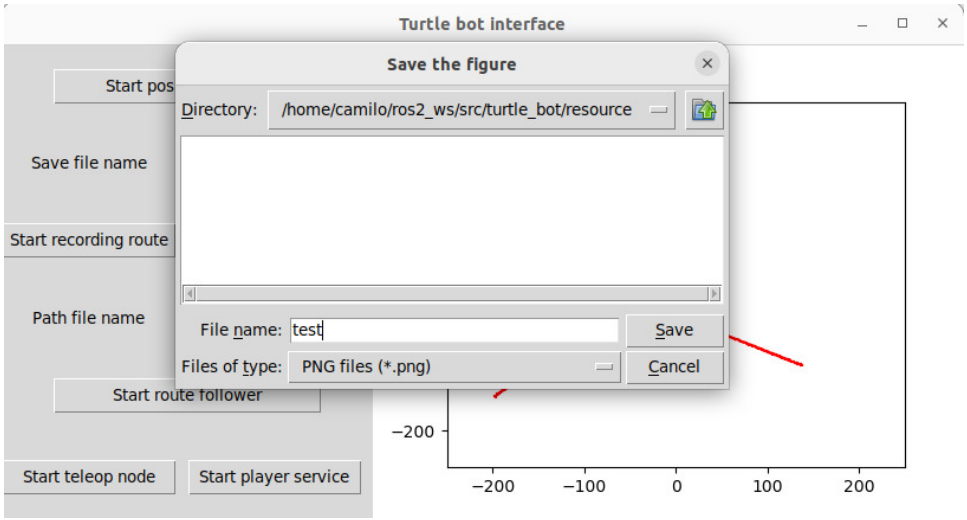

Figura 5. Interfaz para guardar la figura generada por la trayectoria del robot

En caso de elegir guardar la figura, está se guardará en una imagen como la mostrada en la figura 6.

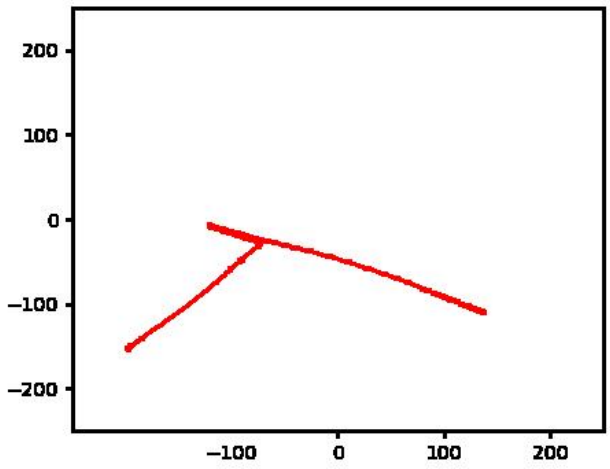

Figura 6. Imagen guardada de la trayectoria realizada por el robot.

En el grafo presentado en la figura 7 se muestran las comunicaciones entre nodos cuando se esta ejecutando el graficador de la posición del robot. Aquí se muestra como `sim_ros2_interface` publica en el topico `turtlebot_position` su posición actual al cual esta suscrito el nodo `turtle_bot_interface`.

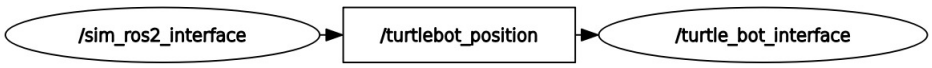

Figura 7. Grafo para el seguimiento de la posición del robot (`turtle_bot_interface`).

1-C. Guardar recorrido del robot.

Para el caso de la funcionalidad de guardar el recorrido del robot, está se implemento en el mismo nodo `turtle_bot_interface` y presenta el grafo mostrado en la figura 8. Aquí se muestra como el nodo se suscribe al tópico que publica la velocidad del robot `turtlebot_cmdVel` de forma que pueda guardar dichas velocidades con tal de replicarlas posteriormente.

Cabe aclarar que en la caja de texto de la interfaz solamente se ingresa el nombre con el que se quiere guardar el archivo el cual es posteriormente **guardado en la carpeta resource** del paquete.

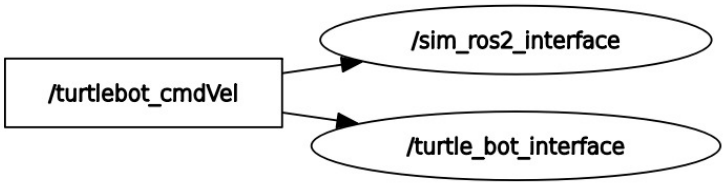

Figura 8. Grafo para guardar un recorrido del robot.

I-D. Reproducción de la trayectoria del robot.

Con el objetivo de reproducir una secuencia de acciones realizadas previamente por el robot y guardadas en un archivo de texto, se creo el nodo `turtle_bot_player`. Este nodo busca en la carpeta **resource** el archivo de texto correspondiente a una trayectoria previa del robot y la reproduce.

*I-D1. **Creación del nodo turtle_bot_player**:* Para la creación de este nodo se hizo uso de nuevo del lenguaje de programación Python. Las librerías utilizadas en este caso fueron:

- **rclpy:** Con esta librería se habilitan todas las funcionalidades de *ROS2* desde Python.
- **time:** Esta librería permite que el tiempo con el que se ejecutaron las acciones anteriormente sea el mismo con el que se reproduzcan. Esto quiere decir, por ejemplo, que cuando el usuario esta generando la ruta al oprimir las teclas *W*, *A*, *S*, *D*, se tiene un *delay* entre las acciones. Por lo que, para que la reproducción de la ruta sea lo más fiel posible a la original, esos tiempos entre acciones deben ser los mismo y esta librería lo permite.
- **threading:** De forma similar al uso que se le dio en la creación del nodo `turtle_bot_interface`, esta librería permite crear múltiples hilos que permiten tener una solución concurrente. En este caso esa concurrencia se refiere a la reproducción de diferentes rutas cada vez que se le haga una llamada al servicio.
- ***geometry_msgs.msg:*** El uso de esta librería es exactamente el mismo que el que se le dio en la creación del nodo `turtle_bot_interface`. Permite definir el tipo de mensajes, siendo en este caso tipo `Twist`.
- **nav_msgs.srv:** De manera muy similar a su uso en la creación del nodo `turtle_bot_interface`, esta librería define los mensajes tipo `LoadMap` de forma que se tiene un estándar para los mensajes, pero que esta vez se **reciben** al servicio encargado de reproducir el recorrido del robot.

*I-D2. **Funcionamiento del nodo**:* En la figura 9 se aprecia la inicialización del nodo en la consola. Luego, la figura 10 muestra el correspondiente grafo generado por `rqt_graph` cuando se esta ejecutando el nodo `turtle_bot_player`. Aquí se observa como el nodo `turtle_bot_player` publica en el tópico `turtlebot_cmdVel` al cual esta suscrito el nodo `sim_ros2_interface` que cual corresponde a la simulación de CoppeliaSim. Por último, la figura 11 muestra como lo siguiente. Primero, estando en el nodo `turtle_bot_interface` se oprime el botón *start route follower* y se envía un mensaje tipo `LoadMap`, el cual contiene el nombre del archivo, al nodo `turtle_bot_player`. Estando en último nodo, se proceden a publicar las velocidades en el tópico *turtlebot_cmdVel* y estas velocidades son leídas finalmente por el software de *CoppeliaSim*.

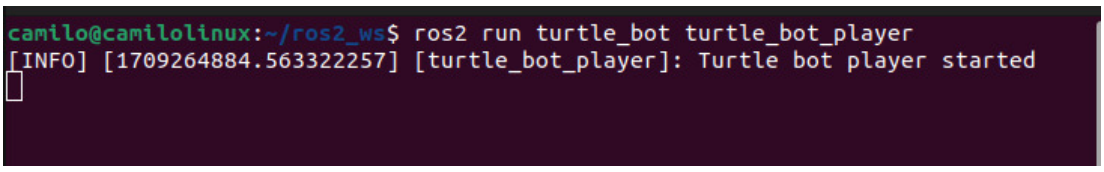

Figura 9. Inicio del nodo para para reproducir un recorrido del robot (`turtle_bot_player`)

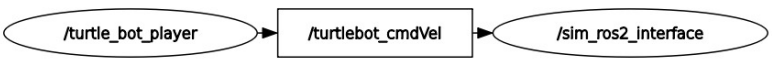

Figura 10. Grafo para reproducir un recorrido del robot (`turtle_bot_player`)

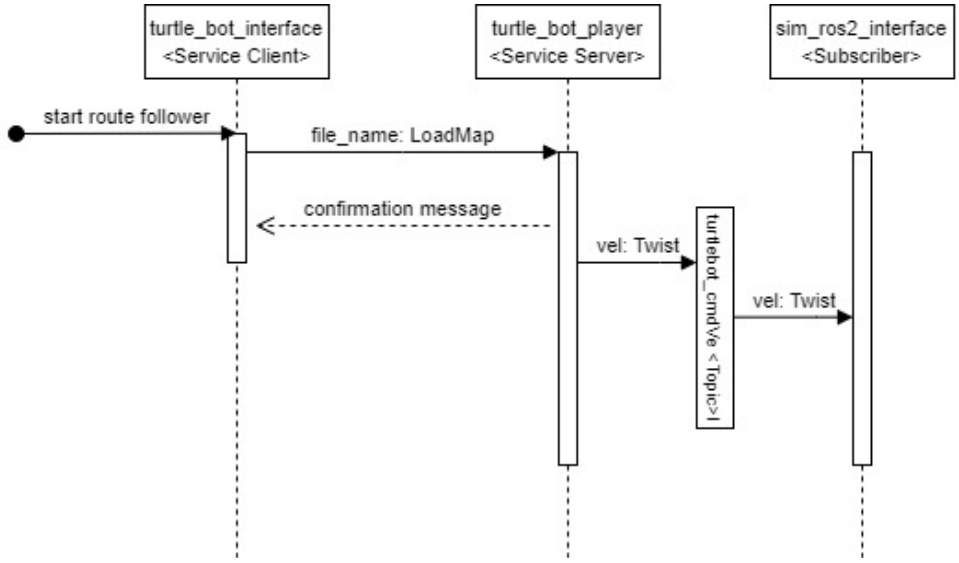

Figura 11. Diagrama de ejecución para la ejecución del servicio de `turtle_bot_player`

I-E. Solución general

El grafo de la figura 12 presenta la solución general desplegada con los 3 nodos funcionando normalmente. Aquí se puede ver un flujo de la información secuencial comenzado desde los nodos `turtle_bot_player` y `turtle_bot_teleop` los cuales publican en el tópico de la velocidad del robot `turtlebot_cmdVel`. Dichas velocidades las recibe el nodo de CoppeliaSim (`sim_ros2_interface`) con tal de controlar el robot de acuerdo a las velocidades publicadas en ese tópico. El nodo correspondiente al robot publica su posición en el tópico `turtlebot_position` de forma que el nodo `turtle_bot_interface` se subscriba a este y puede graficar la información correspondiente a la posición del robot.
El vídeo con la explicación del funcionamiento de todos los nodos y la aplicación en general se encuentra en el siguiente **enlace**.

Figura 12. Grafo de los nodos y topicos activos mediante rqt_graph

II. Conclusiones

- La principal dificultad encontrada fue a la hora de tomar las entradas del usuario para el nodo que controla el robot dadas las diferentes opciones que se presentaban y las funcionalidades de cada una de estas. Al final la opción escogida (librería curses) fue debido a que otras librerías podrían producir problemas a la hora de manejar múltiples hilos, especialmente debido al hilos de la suscripción del nodo.
- Dado que puede que se manejen– múltiples funcionalidades al mismo tiempo, como por ejemplo graficar la ruta del robot y seguir una ruta automáticamente al mismo tiempo, fue necesaria la implementación concurrente de la solución y por tanto se utilizaron hilos (mediante la librería threading) que permitieron realizar múltiples acciones al mismo tiempo.
- La comunicación entre nodos mediante tópicos con los cuales los nodos pueden suscribir o publicar proporciona una forma asincronica de enviar información que es muy útil a la hora de desacoplar los nodos y tener una aplicación modular.
- Los servicios en ROS permiten establecer una arquitectura cliente-servidor que permite a un nodo ofrecer funcionalidades a los cuales los demás nodos pueden acceder de forma sencilla y así poder tener un flujo de ejecución sencillo y limpio.

Referencias

[1] ROS2 Documentation, Open Robotics, 2024.
[2] Graphical User Interfaces with Tk, Python Software Fundation, 2024.