Universidad de los Andes Colombia | Departamento de Ingeniería Eléctrica y Electrónica

Taller 1: Controlar un robot diferencial en simulación (ROS + CoppeliaSim)

Mateo Chilito Avella
Universidad de los Andes
a.chilitoa@uniandes.edu.co
202214992

Cristihan Meza Poveda
Universidad de los Andes
c.mezap@uniandes.edu.co
202215455

Samuel Mora Carrizosa
Universidad de los Andes
s.morac2@uniandes.edu.co
202123048

Brayan Joya Herrera
Universidad de los Andes
b.joya@uniandes.edu.co
202215312

***Resumen*—En el taller 1 se desarrolla la implementación de un sistema de control y monitoreo para un robot diferencial en un entorno de simulación utilizando ROS2 y CoppeliaSim. Se creó una serie de nodos en ROS2 para la teleoperación del robot, la visualización de su trayectoria y la reproducción de recorridos almacenados. La implementación se lleva a cabo siguiendo principios de programación orientada a objetos (POO) para garantizar modularidad y escalabilidad en el desarrollo del software. CoppeliaSim se emplea como plataforma de simulación, permitiendo validar la cinemática del robot y la interacción con el entorno.**

***Index Terms*—CoppeliaSim, estructura de datos, nodos, Robot diferencial, ROS2, topics, trayectorias.**

I. Introducción

El avance en la robótica ha impulsado el desarrollo de herramientas que permiten el diseño, simulación y control de robots. ROS2 se ha consolidado como un framework ampliamente utilizado para la implementación de sistemas robóticos modulares y escalables, facilitando la comunicación entre nodos y la integración con diversos simuladores. ROS2 es un conjunto de librerías orientadas a la construcción de aplicaciones de robots; este permite la creación y monitoreo de diferentes tipos de ”nodos”, los cuales funcionan como robots virtuales capaces de recibir inputs para realizar diferentes acciones. Estos nodos pueden trabajar en conjunto para realizar tareas complejas.

Uno de los modelos más usados en robótica es el robot de configuración diferencial, donde su cinemática se basa en la variación de velocidades de las ruedas para generar movimiento y cambios de dirección. La simulación de este tipo de robots permite evaluar su comportamiento en distintos entornos antes de su implementación en hardware, reduciendo costos y optimizando el desarrollo de algoritmos de control.

En este laboratorio, tuvimos que crear 3 nodos en ROS2, que fueran capaces de cooperar para lograr que un robot diferencial fuera capaz de moverse en base a una serie de inputs dados por el teclado, exportar su ruta y recibir una ruta preexistente como input, para posteriormente recrear esta ruta. También debíamos crear un nodo capaz de visualizar la ruta y movimiento del robot en tiempo real.

II. Trabajo realizado

II-A. Punto 1

Para solucionar el punto 1, se creó un nodo en ROS llamado *”turtle_bot_teleop”* que toma 2 inputs: la velocidad lineal y angular, que serán tomadas como constantes para el desplazamiento del robot. Posteriormente, el nodo recibe ininterrumpidamente el input del teclado, para controlar el robot, con las teclas *'w', 'a', 's', 'd'*, o ’↑’, ’←’, ’↓’ ’→’ para moverse hacia adelante, girar a la izquierda, moverse hacia atrás o girar a la derecha respectivamente.

La magnitud de movimiento que ocurría con cada pulso de una tecla es proporcional a los valores de velocidad usados como input al inicializar el nodo. Un valor de velocidad más alto significa que el robot se mueve/gira una distancia mayor con cada pulso de una tecla. Para lograr este movimiento en tiempo real en el nodo se crea un publisher para el topic *”\turtlebot_cmdVel”* donde se publica una estructura de datos tipo Twist donde solo varía la velocidad lineal en x y velocidad angular en z. Para cerrar el nodo de forma adecuada se debe presionar la tecla ’q’ seguida de ’Ctrl’ + ’C’.

II-B. Punto 2

Para solucionar el punto 2 se creó un nodo ros, con el nombre de ”turtle_bot_interface”, el cual crea una interfaz capaz de visualizar el movimiento y camino del robot. La interfaz hace uso de un pequeño avatar (una foto de una

tortuga) para visualizar el movimiento y trayectoria.

Para visualizar correctamente la trayectoria, el nodo de interfaz se suscribe al topic al cual el nodo del punto 1 está publicando información, y registra la información del topic. A partir de estos mensajes, la interfaz traza un camino y mueve el avatar del robot de acuerdo al mensaje recibido. La interfaz también es capaz de borrar el trazo de recorrido existente si el usuario lo desea.

II-C. Punto 3

Para implementar el punto 3, se añadió la funcionalidad de salvar el recorrido al nodo de interfaz. Esta funcionalidad se implementó como un botón de 'Guardar' en la interfaz. Este botón guarda los datos de velocidad lineal y angular (guardados como un dato tipo Twist) y los guarda en un archivo tipo .json. Los datos se guardan en este formato debido a que es más fácil para el nodo leer estos más adelante, puesto que se guardan en forma matricial de una vez. Esta implementación también será útil para el punto 4. Finalmente, el nodo solicita al usuario escoger el nombre y ruta del archivo creado.

II-D. Punto 4

Para este punto se implemento un nodo llamado turtle_bot_player encargado de reproducir la trayectoria previamente almacenada ene l archivo .json. Este nodo utiliza un servicio del paquete pcl_msgs.srv para recibir la ruta del archivo y posteriormente publica los comandos de velocidad en el topico turtlebot_cmdVel. el nodo player opera mendiante un servicio que recibe la ruta del archivo seccionado y carga los datos en una lista. Para lograrlo, la función handle_file_reques se encarga de abrir el archivo y leer los datos de velocidad lineal y angular, los cuales serán publicados en secuencia. La interfaz gráfica desarrollada en pyQt5 permite al usuario seleccionar la trayectoria mediante una ventana emergente. Una vez seccionado el archivo, la ruta se envía al nodo player. La confirmación de recepción asegura que el nodo esta listo para iniciar la reproducción. El nodo recorre la lista de comandos leídos del archivo y publica cada uno en el tópico de turtlebot_cmdVel utlizando mensajes de tipo Twist. Los valores se asignan a los campos linear.x y angular.z, reproduciendo de esta manera el recorrido registrado.

Para ejecutar este requerimiento de forma adecuada es necesario detener el requerimiento de teleop, posteriormente ejecutar el nodo de player y en la interfaz cargar el archivo correspondiente a la ruta, por último reiniciar la simulación en coppelia.

III. Resultados y Análisis de Resultados

Se observa como el robot diferencial TurtleBot2 creado y simulado en conjunto con ROS y CooppeliaSlim presenta un funcionamiento sobresaliente en las diversas pruebas realizadas. Comenzando por la teleoperación, mediante el nodo *"turtle_bot_teleop"* permitió desplazar al robot en las direcciones esperadas utilizando las teclas w, a, s y d. La asignación de velocidades lineales y angulares proporcionadas al inicio del nodo influyó directamente en la magnitud del desplazamiento evidenciando la correcta transmisión de comandos mediante el tópico *"turtlebot_cmdVel"*. Durante las pruebas, el robot respondió de manera inmediata a las pulsaciones de teclas y la tecla de escape "q"para terminar el nodo, se ejecutó según lo esperado.

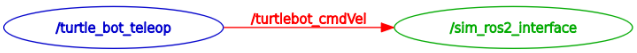

Figura 1. rqt_graph de req1

En la figura 1 se observa como se creó el nodo correctamente, y este publica hacía el topic *"turtlebot_cmdVel"* al cuál está suscrito el nodo perteneciente a la simulación de CoppeliaSim *"\sim_ros2_interface"*.
Al momento de implementar este requerimiento se presentaron algunas dificultades principalmente asociadas al input ininterrumpido del teclado. Para solventar esto se hizo uso de las librerias sys, os, termios y select con las cuales podemos modificar las propiedades de la terminal haciendo que no sea necesario dar un input para seguir el programa, ni dar enter para poder digitar un input, así de esta forma publicar un valor diferente cuando un input no es recibido. Sin embargo, al destruir el nodo se presentaba el problema que no cerraba de forma adecuada y posteriormente la terminal quedaba con las propiedades aplicadas en el nodo haciendo que el texto escrito en la terminal no fuera visible.

Para solventar eso se creó una función cleanup_terminal(), la cuál ejecuta el comando **ssty sane** que devuelve la terminal a su normalidad. Arreglando el error y pudiendo utilizar la terminal de forma adecuada luego de usar y cerrar el nodo.

En paralelo, la interfaz gráfica implementada a través del nodo *"turtle_bot_interface"* mostró la trayectoria del robot en tiempo real mediante un avatar representado por una imagen de tortuga. La posición y orientación del avatar se actualizaron de acuerdo con los mensajes recibidos del tópico **"turtlebot_position"**, mientras que la trayectoria se trazó sobre un lienzo blanco, centrado en el punto de inicio. La opción de borrar la trayectoria permitió restablecer la visualización sin interferencias. La función de guardar el recorrido en un archivo .json (el cual es mucho más optimo para esta practica por su formato en texto plano comparado con un archivo txt) registró las velocidad lineales y angulares publicadas, así como el tiempo de simulación, lo que facilito su uso posterior en la reproducción de trayectorias.

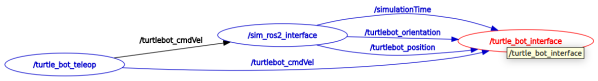

Figura 2. rqt_graph req2 y 3

Como se puede ver en la imagen 2, aparecen 3 nodos, correspondientes al teleop, la simulación en Coppelia y la interfaz. El nodo de la interfaz se suscribe a los topicos de tiempo de simulación, orientación del turtle_bot y la posición. A partir de esto traza la gráfica en la interfaz que posteriormente se puede guardar como una imagen, a su vez se suscribe al topic de cmdVel al cuál publica el nodo de teleop esto con el fin de guardar las acciones que se están dando en el teclado y crear un archivo json para recrear la trayectoria.

Durante la implementación del requerimiento 2, la principal dificultad que se enfrento fue garantizar la actualización en tiempo real de la interfaz gráfica sin afectar su rendimiento. Para resolverlo se realizaron con acciones a funciones que procesan los daros recibidos de los tópicos turtlebot_position, turtlebot_orinettion y simulationTime, lo que permitió tener una interfaz fluida. Por otro lado, en el requerimiento 3, el desafío principal consistió en la sincronización del almacenamiento de las velocidades y el tiempo de simulación ara generar el archivo JSON de manera correcta. Esto se logró resolver mediante un diccionario donde las llaves corresponden al tiempo de simulación recibido del tópico simulationTime y los valores almacenados son las velocidad lineales y angulares publicadas en el turtlebot_cmdVel.

Con respecto al nodo *"turtle_bot_player"*, este logro cargar los archivos de trayectorias aleccionados desde la interfaz gráfica mediante el servicio turtlebot_player_service. La confirmación de recepción fue útil para iniciar la reproducción de la trayectoria replicando los movimientos del robot en el simulador. La publicación secuencial de los comandos de velocidad en el tópico turtlebot_cmdVel se uso para reproducir el recorrido registrado, lo que garantizo la comunicación eficiente entre los nodos que se desarrollaron. Se invita al lector a confirmar estos resultados mediante la simulación disponible en el repositorio adjunto.

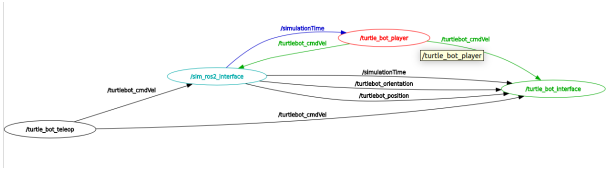

Figura 3. rqt_graph req4

Como se observa en el diagrama 3, el nodo de player ejecuta un servicio que actúa como intermediario entre la interfaz y la simulación, recibiendo un archivo json cargado desde la interfaz y publicando datos hacía el topic de cmdVel de tal forma que se reconstruya la trayectoria, para poder correr este servicio de forma correcta es indispensable que el nodo de teleop se haya detenido.

IV. Conclusiones

La practica evidencio la solidez de un enfoque distribuido en la construcción de sistemas al separar los componentes de teleoperación, interfaz de usuario y reproducción de trayectorias en nodos de ROS2 independientes. Este método garantizó alta coherencia en la transmisión de datos y facilitó la integración de nuevas funcionalidades sin afectar la estabilidad general de la aplicación. El uso de CoppeliaSim como entorno de simulación fue una hermanita valiosa para confirmar de manera realista la cinemática del robot diferencial y ensayar múltiples configuraciones de velocidad como desplazamiento. Al usar programación concurrente junto a programación orientada a objetos, se logró que el sistema sea escalable como estable, creando posibilidades de ejecución de varias tareas en simultaneo sin provocar bloqueos o retardos no deseados. Así mismo, la interfaz gráfica diseñada con ayuda de PyQt5 garantizó un medio intuitivo para observar el comportamiento del robot y el almacenamiento de los recorridos en el formato JSON. En términos generales, este proyecto generó competencias relevantes para el equipo, desde el manejo de tópicos, servicios y suscriptores en ROS2, hasta el desarrollo de habilidades de documentación y validación de sistemas robóticos con el uso de herramientas como rqt_graph. Este conjunto de herramientas se vuelven invaluables en pleno siglo XXI cuando es vital para un ingeniero electrónico desarrollar habilidades a la vanguardia de tecnologías como la robótica o el machine learning que se trabajará después en el curso.

Referencias

[1] Oracle Corporation: Download VirtualBox, 2025 [Online].
[2] Ubuntu 22.04 LTS, 2022 [Online]
[3] Open Robotics: Installing ROS 2 Iron on Ubuntu, ROS Documentation, 2023 [Online]
[4] Coppelia Robotics AG, ÇoppeliaSim User Manual,"2023. [Online]
[5] Robotics Back-End, ROS2 Tutorials - ROS2 HUMBLE for Beginners"2022. [https://www.youtube.com/playlist?list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy]