Taller 1
Introducción a ROS TurtleBot2

Laboratorio de Robótica, 2 de marzo de 2024

1st Ángel J. Pulido G.
Dpto. Ingeniería Eléctrica y Electrónica
Universidad de los Andes
Bogotá DC, Colombia
a.pulidog@uniandes.edu.co
202015284

2nd Leffer A. Trochez
Dpto. Ingeniería Eléctrica y Electrónica
Universidad de los Andes
Bogotá DC, Colombia
l.trochez@uniandes.edu.co
202013764

3th Cesar E. Antonio H.
Dpto. Ingeniería Eléctrica y Electrónica
Universidad de los Andes
Bogotá DC, Colombia
c.escobarh@uniandes.edu.co
202014294

Keywords* - *ROS, nodos, tópicos, paquetes, servicios, CoppeliaSim, Python, TurtleBot, velocidad lineal y angular.

I. Materiales y Metodología

Para este informe tuvimos que hacer uso de una maquina virtual la cuál nos permite ingresar al sistema operativo de Linux, ya que el programa que vamos a usar funciona en un ambiente de Ubuntu. Una vez en linux, el siguiente paso es descargar los programas necesarios, que en este caso fueron CoppeliaSim y ROS2, adicionalmente para este taller tuvimos que instalar la libreria pygame y sshkeyboard, poniendo en un terminal ”sudo apt pip3 install el nombre”, una vez instalados los programas y dependencias anteriores, se empezó a desarrollar el taller que consta de 4 partes. La primera parte, tiene como objetivo pedir al usuario la velocidad angular y linear a la que queremos que viaje el robot, además debemos poder controlar el robot con las teclas del teclado. La segunda parte, consiste en guardar la posición donde el robot ha estado y a partir de esos datos crear una matriz gráfica. La tercera parte, consistía en preguntar al usuario si desea guardar los datos y que el usuario tenga la posibilidad de dar nombre al archivo. Finalmente, la ultima parte consistía en darle un recorrido por medio de un servicio y que el robot lo replique.

II. Resultados y Análisis de resultados

II-A. PUNTO 1

En este primer punto el objetivo es consultar al usuario a que velocidad desea que se mueva el robot, y mediante el uso de las teclas ”W”,”S”,”A”y”D” poder desplazar el robot simulado en direcciones deseadas. En consecuencia la tecla ”W”permite el movimiento hacia al adelante, la tecla ”S”hacia atrás, la tecla .ª”gire a la izquierda y la tecla ”D”permite el giro a la derecha

Debido a la necesidad de controlar el movimiento del robot mediante las teclas ”W”, ”S”, ”A” y ”D”, se ha implementado una estrategia de código que se basa principalmente en la función listen_keyboard de la biblioteca sshkeyboard. Esta función facilita la identificación de las teclas que han sido presionadas. Inicialmente, se definen variables para representar la velocidad radial y angular del robot. Posteriormente, el programa solicita al usuario que ingrese los valores deseados para estas velocidades.

A través de funciones específicas, se establece el comportamiento tanto al presionar como al liberar una tecla. Dentro de estas funciones, se utilizan estructuras condicionales if para asignar a las teclas mencionadas la tarea de incrementar o reducir sus respectivos valores de velocidad angular o lineal. El algoritmo completo se visualiza de manera clara en el siguiente diagrama de flujo:

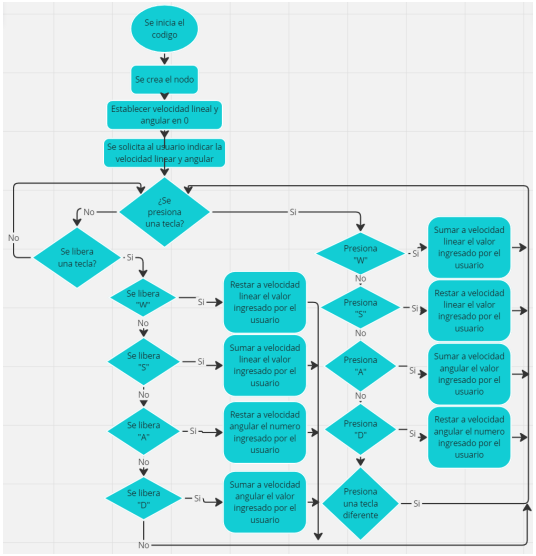

Figura 1. Diagrama de flujo del control del robot con las teclas ”W”,”S”,”A”y”D”.

Para reflejar las acciones deseadas en el entorno simulado de Coppelia, se hace uso de la librería rclpy, con la cual se crea el nodo /turtle_bot_teleop y el publicador (publisher). Este publicador se utiliza para enviar la información de las velocidades como un mensaje del tipo Twist al tópico "turtle-bot_cmdVel". De esta manera, se establece la comunicación efectiva entre el código y el entorno simulado, permitiendo controlar el movimiento del robot según las teclas presionadas.

Al ejecutar el código se obtiene la siguiente imagen:

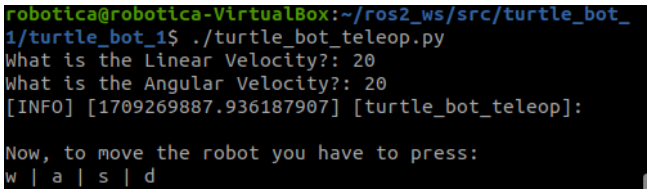

Figura 2. Solicitud al usuario de la velocidad lineal y radial deseada

En esta se puede evidenciar como el código le pide al usuario ingresar la velocidad lineal y radial deseada, para posteriormente esperar las instrucción dependiendo de las teclas "W","S","A"y"D".

Se procede a oprimir las teclas ya mencionadas, con lo cual se obtiene la siguiente imagen:

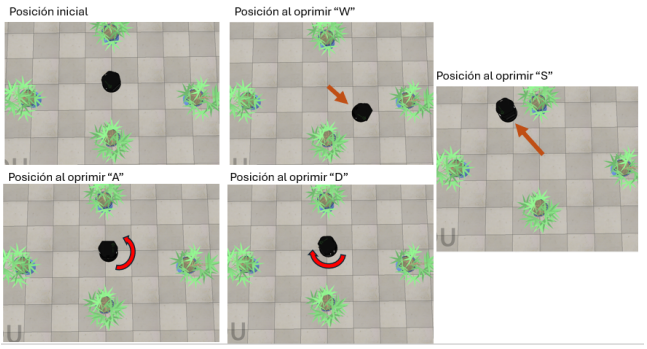

Figura 3. Movimiento del robot al presionar las teclas "W","S","A"y"D".

Sin embargo, en caso de oprimir una tecla no definida se obtiene:

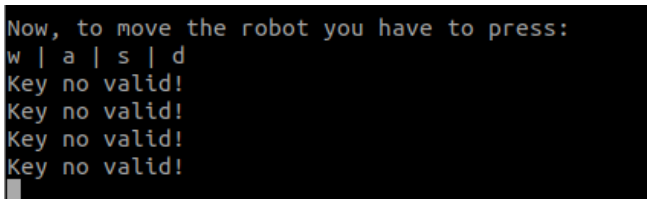

Figura 4. Mensaje obtenido al presionar una tecla diferente a las establecidas.

De esta forma se demuestra el funcionamiento adecuado del código, con el cual es posible controlar el robot con las teclas "W","S","A"y"D". Sin embargo con el objetivo de comprender la interacción entre el código y el simulador Coppelia, se presenta a continuación el grafo de ROS presente durante la ejecución del código:

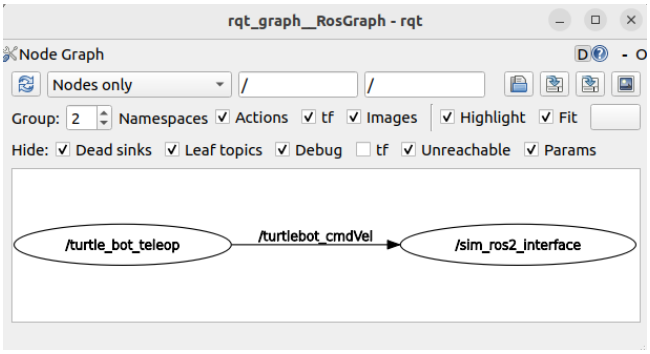

Figura 5. Grafo de Ros obtenido durante la ejecución del código de movimiento con las teclas "W","S","A"y"D".

De esta forma es evidente como el nodo /turtle_bot_teleop mediante un publisher, publica los valores de la velocidad por medio del topico /turtlebot_cmdVel

II-B. PUNTO 2

En esta segunda fase, el objetivo principal es graficar los resultados obtenidos en la fase anterior. Esto implica la implementación de la visualización en tiempo real de la posición y el camino recorrido por el robot Coppelia en una interfaz gráfica. Además, se busca la capacidad de asignar un nombre a la gráfica y guardarla en un directorio específico.

Para lograr esto, se utiliza la librería rclpy, que permite la suscripción a los datos de posición y velocidad del robot, esenciales para construir dinámicamente la gráfica de posición. La biblioteca pygame se importa para facilitar el diseño de la interfaz gráfica, incluyendo elementos como botones y la actualización constante de la ubicación del robot.

Se incorpora el tipo de mensaje Twist de geometry_msgs.msg para manejar las velocidades radiales y angulares del robot, proporcionando una estructura de datos adecuada para representar la información necesaria en la gráfica.

La funcionalidad de seleccionar un archivo del sistema y especificar un directorio para guardar la imagen de la ruta se logra mediante el uso de filedialog de tkinter, permitiendo una interacción sencilla con el sistema de archivos.

Finalmente, la librería os se emplea para realizar modificaciones en el archivo de la ruta obtenida. Esto incluye la capacidad de borrar el archivo existente para permitir la creación y guardado de nuevas gráficas de ruta, contribuyendo así a una implementación completa y eficiente del sistema.

Utilizando tanto Twist como rclpy, se realiza la suscripción a los tópicos de posición y velocidad '/turtlebot_position' y '/turtlebot_cmdVel'. Posteriormente, se escalan estos valores con el objetivo de que los cambios de posición sean más evidentes. Se crea una interfaz con un título y un botón de guardar, lo cual es posible gracias a pygame. Esta interfaz se actualiza mediante pygame.display.update(), lo que permite graficar las posiciones de la circunferencia en cada momento.

Además, gracias a pygame.MOUSEBUTTONDOWN, es posible detectar cuando se realiza un clic en el botón. En

caso de suceder, se toma una captura de la imagen de la ruta, permitiendo definir el nombre del archivo para guardarlo. El algoritmo de este funcionamiento se presenta en el siguiente diagrama de flujo:

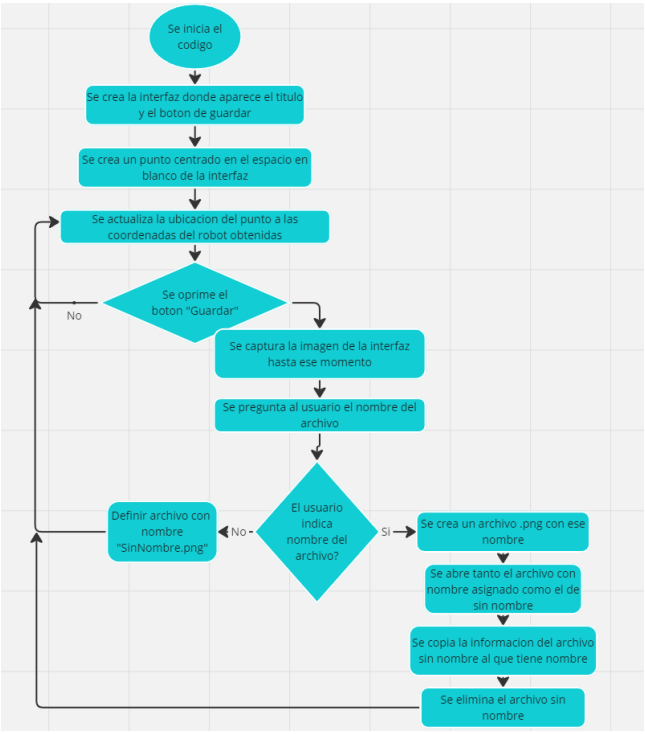

Figura 6. Diagrama de flujo sobre la creacion y almacenamineto de una hoja de ruta del robot.

Es así que al ejecutar el código junto al del punto 1 se obtiene las siguientes imágenes para el uso de las diferentes teclas validas:

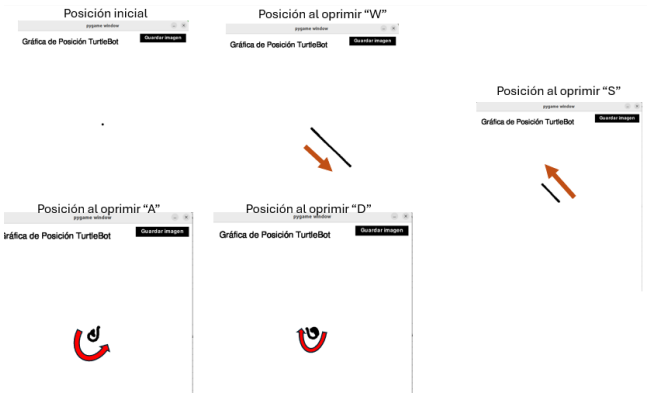

Figura 7. Imagen de ruta obtenida para el uso de cada una de las teclas validas

De igual forma el proceso de guardar se presenta a continuación:

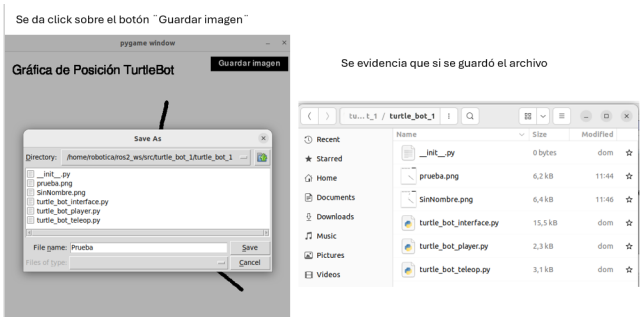

Figura 8. Proceso de guardado de la imagen de ruta

Así como en el punto 1 con el objetivo de comprender la interacción entre el código y el simulador Coppelia, se presenta a continuación el grafo de ROS presente durante la ejecución del código:

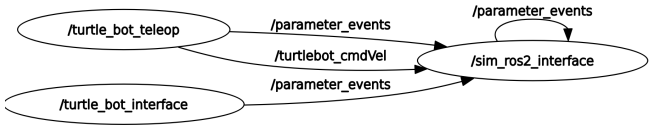

Figura 9. Grafo de Ros obtenido durante la ejecución del código de movimiento con el código de generación y guardado de imagen de ruta.

De esta forma se evidencia la exitosa creación del nodo /turtle_bot_teleop el cual solicita información de la simulación

II-C. PUNTO 3

La tercera sección de este informe aborda la interacción con el usuario para determinar si desea almacenar la trayectoria del robot. Con este fin, se han introducido nuevas funciones relacionadas con la toma de decisiones y se ha realizado una ligera modificación en el código de la interfaz gráfica. En este contexto, se han creado dos funciones para la toma de decisiones y otras dos funciones específicas para guardar la trayectoria, además de las funciones asociadas a los botones.

Para comenzar, se define una función de evento encargada de detectar la posición del mouse y habilitar los botones generados mediante Pygame. Posteriormente, se ingresa a la función del botón construido con la ayuda de Pygame, la cual devuelve los botones operativos en la interfaz gráfica.

Se introduce una variable global establecida como ”True”para iniciar la modificación mencionada anteriormente. Esta modificación implica la creación de una interfaz previa a la mencionada en el punto 2, donde se consulta al usuario sobre su deseo de guardar los datos de la trayectoria. El usuario puede responder haciendo clic en alguno de los dos botones creados: ”Sí, quiero.º ”No, no quiero”.

En caso de aceptar la opción de guardar la trayectoria, se extraen los valores de velocidad lineal y angular. Sin embargo, al solicitar esta información, se obtiene una mezcla de la velocidad con la posición. Para abordar esto, se implementa un filtro donde los valores con un número de dígitos menor

a 7 se consideran velocidades. Una vez discriminados estos valores, se procede a almacenarlos en un archivo en forma de cadenas separadas por comas.

Simultáneamente, mediante Pygame y siguiendo la lógica del punto 2, se crea una interfaz con título y botones que muestra el movimiento de la circunferencia representativa del robot de Coppelia. A diferencia de la interfaz anterior, se incluye un botón adicional para guardar la trayectoria. Gracias a la funcionalidad de filedialog, se consulta al usuario sobre el nombre que desea asignar al archivo al guardarlo, con ”SinNombre.txtçomo predeterminado.

El algoritmo se presenta en las figuras 10 y 12 donde es importante mencionar que si se escoge la opción de ”no”, el algoritmo va realizar las mismas acciones del punto anterior, por otro lado si se elige la opción de ”si”se procede a realizar el proceso antes descrito.

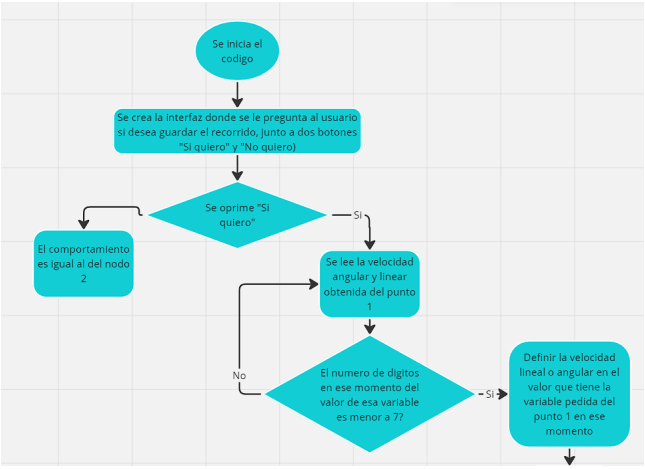

Figura 10. Imagen de diagrama de flujo sobre guardar o no el recorrido

Figura 11. Imagen de diagrama de flujo sobre guardar o no el recorrido

El resultado de todo eso es que cuando en el terminal se corra el nodo ”tultle_bot_interface”salga la siguiente interfaz con los dos botones, entonces si se dice que no se sigue a la interfaz del punto anterior, si se dice que ”si”, se pasa a la interfaz de la figura[**?**] este caso sale la opción de guardar recorrido como se ve en la figura 12.

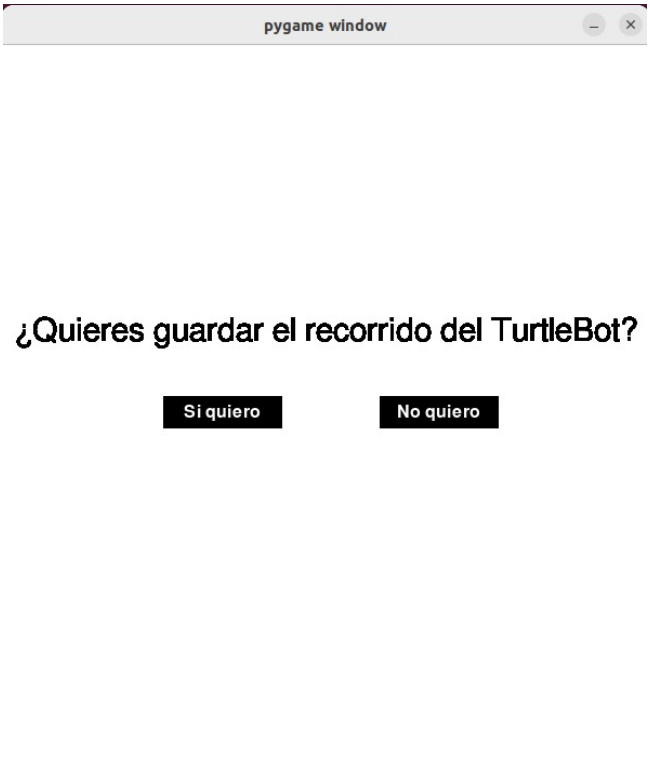

Figura 12. Imagen de diagrama de flujo sobre guardar o no el recorrido

Figura 13. Interfaz cuando se presiona el botón si quiero

Cuando se guarda el recorrido se tiene la opción de guardarlo con el nombre deseado figura 14

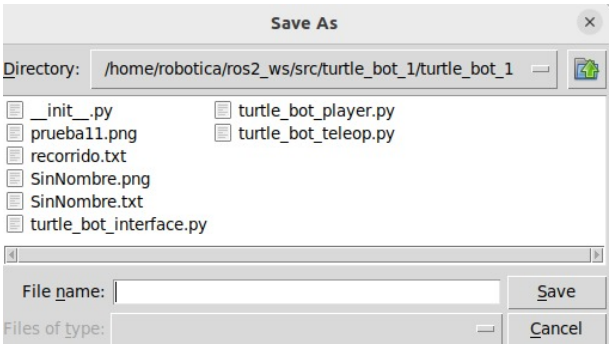

Figura 14. Cuando se oprime el botón de guardar recorrido

Finalmente mencionar que el grafo de los nodos es igual al del punto anterior debido a que solo es una modificación sobre lo que ya se había realizado, dicha modificación se hace sobre el nodo de interface debido a eso no se afectan los tópicos que se suscriben a los nodos antes definidos.

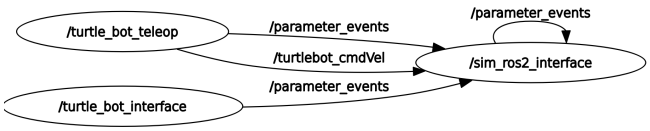

Figura 15. Grafo de Ros obtenido durante la ejecución del código guardar el recorrido

II-D. PUNTO 4

Para esta parte igual que las anteriores lo primero que se hace es exportar las librerias, para este caso usamos rclpy , para crear el nodo , usamos Twist de geometry_msgs.msg por el tipo de mensaje del topico, la libreria os para trabajar con archivos y Setbool de la libreria example_interface.srv que es el mensaje con el que se va a comunicar el servicio, con todas esas librerías ya importadas, lo primero en hacer es definir la función init que va a iniciar el publisher, el timer y el cliente.Después se crea una función que llamamos recorrido , que espera a que le llegue un servicio, a penas le llegue una solicitud se comunica con otra función que le pusimos de nombre función la cual tiene como objetivo obtener el nombre del archivo que se va a replicar con ayuda de la libreria os antes importada, o en caso de no encontrarse informa que no se encontró ningún archivo con ese nombre.
Después de que se realizará lo anterior y en caso de no presentar fallos es donde se empieza a analizar el archivo con las coordenadas en lineales y angulares, como estos datos están unidos con ";"tenemos que usar split para separar el array y una vez separados ya podemos definir las variables asociadas a la velocidad lineal o angular, se publicaran esas variables en el terminal para que se pueda saber en todo momento en que parte del recorrido esta , finalmente se aumenta en 1 el contador para que se sepa en que fila del archivo se encuentra.

Guardando el recorrido del punto anterior en un archivo txt con las especificaciones antes mencionadas , se procede a correr el nodo /turtle_bot_player en el terminal dándonos como resultado la siguiente imagen:

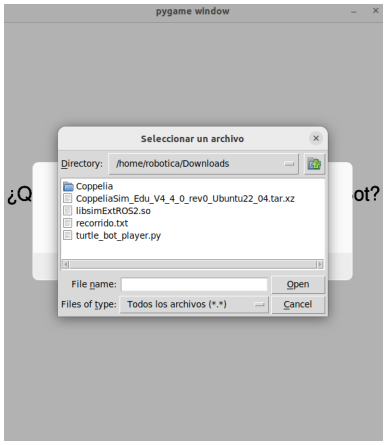

Figura 16. Solicitud para elegir que recorrido recrear

En la cual se selecciona el archivo recorrido que se busca recrear, una vez seleccionado, el robot procede a movemerce siguiendo las intrucciones del archivo por lo cual al terminar se puede apreciar como el recorrido que hace el robot que es muy similar al que se guardó anteriormente.

Figura 17. Prueba de que se replica el recorrido

Finalmente este nodo cierra con la función main , que se encarga de inicializar el código, evita que se cierre con spin y ya cuando se quiera cerrar el terminal cierra todo. El diagrama de flujo se puede ver en la siguiente imagen.

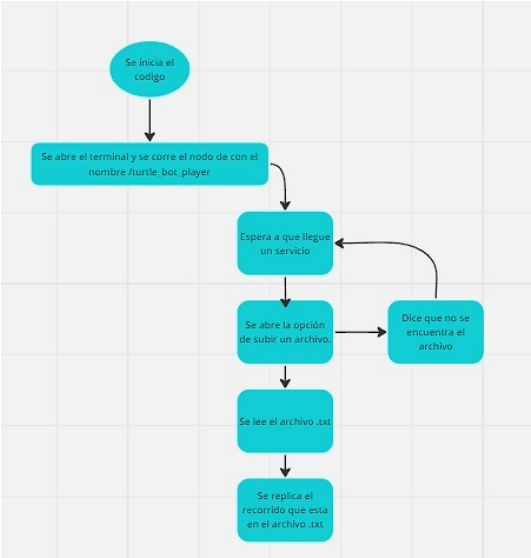

Figura 18. Diagrama flujo punto 4

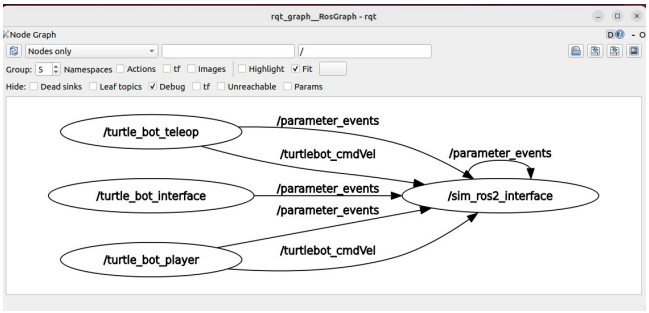

Figura 19. Grafo de Ros obtenido durante la ejecución del código replicar recorrido

En el nodo de ROS se ve como se aumenta el nodo de /turtle_bot_player al diagrama de los puntos anteriores y este nodo lo que hace es suscribirse al nodo de /Sim_ros2_interface con los tópicos relacionados a las velocidades angulares y lineales.

III. Limitaciones

En esta sección describiremos las limitaciones que tiene nuestro código, la primera es que a la hora de gráficar se hizo un filtro que deja pasar los datos de posición ya que al momento de tomar los datos nuestro código toma los datos tanto de las velocidades, y eso crea gráficas erróneas.Otra limitación que el usuario debe poner un nombre especifico para que lo reconozca recorrido", además este archivo se debe encontrar en descargas entonces cuando se guarde , toca revisar donde queda guardado para moverlo hasta descargas en caso de que no se pueda escoger la carpeta para guardar el archivo.Otra limitación es que cada vez que se crea el archivo recorrido , se crea otro que se llama "SinNombre", entonces debemos borrar ese archivo cada vez que se desee crear un nuevo recorrido o cada vez que se desee replicar un recorrido. Finalmente para el almacenamiento de un recorrido adecuado en el punto 3, y para que sea replícable en el punto 4, el usuario debe controlar el robot oprimiendo las teclas repetidamente, es decir no mantener presionado el botón si busca un movimiento continuo, en cambio tan pronto como lo presiona, soltarlo y volverlo a presionar

IV. Conclusiones

- En este proceso de simulación utilizando el simulador Coppeliasim y ROS 2 se implemento un código capaz de pedir tanto la velocidad lineal como la angular, para poder que el robot se mueva a la velocidad indicada anteriormente en la dirección de la tecla que se oprima y en caso de no oprimir nada el robot se quede quieto, permitiendo que el robot pueda moverse controladamente por un entorno permitiendo al usuario esquivar obstáculos, ó parar en caso que sea necesario, lo anterior es importante ya estos son los primero pasos para lograr que un robot en la vida real pueda realizar tareas en cualquier entorno y este no se choque.

- Gracias a el simulador CoppeliaSim se pudieron implementar los conceptos de nodos, tópicos servicios, nodo publisher, entre otros con un robot que aunque fue simulado funciona de manera similar a la realidad.El procedimiento llevado a cabo deja en claro las capacidades que tiene ROS como herramienta de trabajo , además de todas las ventajas que este programa nos brinda a la hora de programar funciones que debe realizar un robot tales como avanzar en la dirección que deseamos o ir guardando los datos de posición por donde pasa, finalmente por medio de ROS se puede replicar un recorrido que se le de al robot, abriendo la puerta a un gran numero de posibilidades que se pueden implementar en un robot con ayuda de este programa.