Informe Taller 2

Robótica - IELE 3338.
2022-1.

1st Dylan Mateo Zambrano Salamanca
Departamento de Ingeniería Eléctrica y Electrónica
Universidad de los Andes
Bogotá, Colombia
dm.zambrano@uniandes.edu.co

2nd Andrés Felipe Manrique Moreno
Departamento de Ingeniería Eléctrica y Electrónica
Universidad de los Andes
Bogotá, Colombia
af.manrique@uniandes.edu.co

3rd Laura Milena Gamba Sánchez
Departamento de Ingeniería Eléctrica y Electrónica
Universidad de los Andes
Bogotá, Colombia
lm.gamba@uniandes.edu.co

4rd Camilo Andrés Arango Navarrete
Departamento de Ingeniería Eléctrica y Electrónica
Universidad de los Andes
Bogotá, Colombia
ca.arango@uniandes.edu.co

***Resumen*—En el segundo taller se diseñó el modelo mecánico y electrónico de un robot diferencial, al cual se aplicaron los nodos realizados en el primer taller. De forma tal que fuese capaz de manejarse a través del teclado, y que pudiera localizarse en tiempo real.**

***Index Terms*—Diseño, Ensamblaje, Nodo, Plano, Servicio, Threading**

I. Introducción

Las simulaciones permiten un primer acercamiento al funcionamiento de la robótica, sin embargo, para crear un robot se requiere de un diseño mecánico y electrónico ideado a partir de las pruebas realizadas en simulación. En el presente informe se expone el desarrollo de dichos diseños por medio de planos y diagramas esquemáticos. De igual manera, se presentan los resultados obtenidos al aplicar los requerimientos de funcionamiento del primer taller al robot físico.

II. Objetivos

- Cree un nodo en ROS que permita a un usuario controlar por teclado el robot físico. Es decir, que las velocidades que son publicadas al robot de forma lineal y angular (en los tópicos respectivos) coincidan de forma natural con la distribución de teclas básicas, teniendo así cuatro movimientos diferentes.
- Crear un nodo de ROS que permita visualizar la posición en tiempo real del robot, a través de una interfaz gráfica.
- Complementar los nodos anteriores para preguntar al usuario en la interfaz si desea guardar el recorrido del robot y posteriormente guardar en un archivo .txt la secuencia de acciones que realizó el usuario durante el recorrido del robot.
- Crear un nodo de ROS que a partir de un archivo .txt de un recorrido guardado, reproduzca la secuencia de acciones del robot. La partida a reproducir debe ser solicitada a partir de un servicio y dicha llamada al servicio ser realizada directamente desde la interfaz creada anteriormente.

III. Materiales

En primer lugar, se realizó un listado de los materiales y componentes que podrían necesitarse para el montaje del robot. A partir de los requerimientos de funcionamiento se añadieron nuevos componentes, tales como las ruedas de castor que proporcionaran mayor estabilidad, hasta obtener la lista completa que se observa en la tabla I.

Componente (Cantidad)	Referencia	Link de Referencia
Raspberry Pi	4 Model B	https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/
Arduino	UNO	https://store-usa.arduino.cc/products/arduino-uno-rev3/
Puente H	L298N	https://naylampmechatronics.com/drivers/11-driver-puente-h-l298n.html
Motor DC (2)	1:48	https://opencircuit.es/producto/Motorreductor-DC-1-48
Ruedas (2)	65mm	https://www.amazon.com/-/es/ruedas-pl%C3%A1stico-neum%C3%A1tico-Arduino-paquete/dp/B077M8C2XN
Encoder (2)	HC-020K	https://opencircuit.es/producto/codificador-de-velocidad-hc-020k
Regulador	LM2596	https://electronilab.co/tienda/modulo-lm2596-convertidor-de-vol\taje-dc-dc-buck-1-25v-35v/
Ruedas de Castor (2)	25mm	https://naylampmechatronics.com/drivers/11-driver-puente-h-l298n.html
Baterías recargables	Litio 7,5V	https://articulo.mercadolibre.com.mx/MLM-661104489-bateria-recarga\ble-litio-lipo-74v-1500-mah-arduin\o-robot-_JM

Tabla I
Lista de Materiales con Referencia

IV. Desarrollo

IV-A. Diseño Mecánico:

En primer lugar, se tomaron en cuenta las dimensiones de los materiales de la Tabla I y las restricciones de diseño dadas en el enunciado del proyecto final para poder comenzar el diseño de la base y cómo se posicionaría cada componente. Con esto en mente se diseñó un chasis de 20x21cm de 2 niveles, entre los cuales se encontrarán los motores, y con espacios a los costados para las ruedas de 3cm de ancho. El primer nivel tiene todos los agujeros para asegurar los componentes de la lista, mientras el subnivel tiene solo los huecos de los espaciadores y de los encoders.

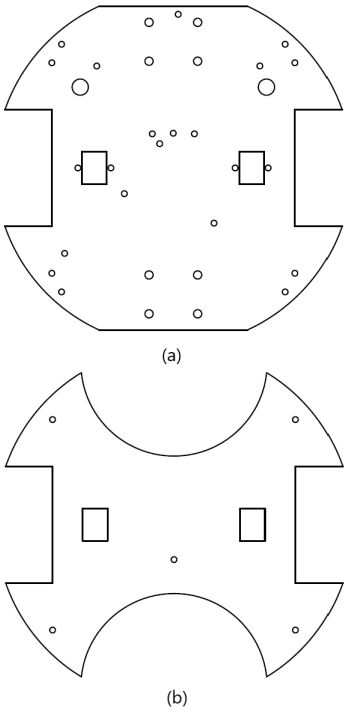

Figura 1. Plano de las bases (a) del primer nivel y (b) del subnivel.

Para conocer la ubicación de cada perforación se realizó una disposición de componentes de forma virtual, lo cual requirió la toma de medidas de cada componente para su posterior modelamiento en el programa Inventor, tal como se muestra en la figura 2.

IV-B. Diseño Electrónico:

Teniendo en cuenta los criterios de funcionamiento del robot se hizo un diseño general de las conexiones entre dispositivos. En un principio se contempló la idea de conectar el puente H directamente a la Raspberry y no utilizar un Arduino, sin embargo, por recomendación de los profesores y para mayor protección del circuito se decidió utilizar un Arduino y un regulador de voltaje para la Raspberry, resultando en el diseño que se observa en la Figura 3.

Cabe mencionar que el protocolo de comunicación utilizado entre la Raspberry y el Arduino fue rosserial, el cual permite

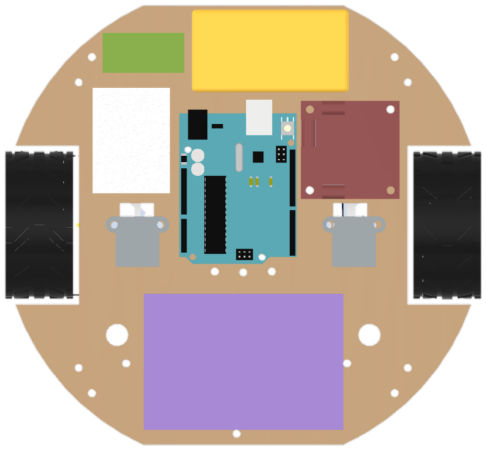

Figura 2. Disposición de los componentes en Inventor

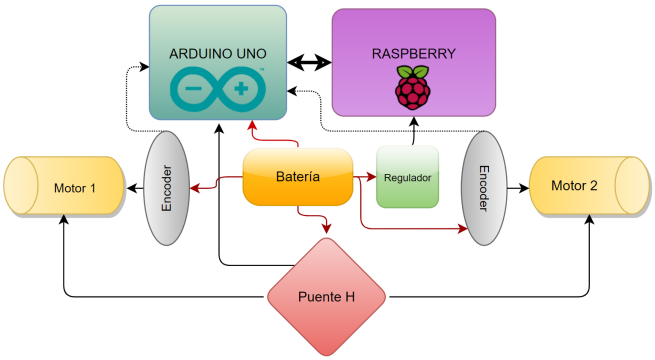

Figura 3. Diagrama general del diseño electrónico

crear nodos desde Arduino por el cual se podrán publicar mensajes a ROS o suscribirse a ellos. Para que este protocolo funcione se debe descargar el paquete *rosserial*[21] disponible para Arduino, que a su vez necesita de la librería *ros_lib* para hacer posible la interacción entre los programas de Arduino y ROS. Teniendo en cuenta lo anterior, se hizo un subscriber en Arduino que leyera los pulsos PMW que se envían desde la la raspberry al indicar las velocidad lineal y angular en la consola. En la dirección contraria un publisher estará enviando a ROS la información referente a la velocidad de las ruedas (obtenida con los encoders) para así calcular la posición del robot. Tanto el publisher como el subscriber mencionados anteriormente se pueden apreciar en las consolas del lado la izquierdo en la Figura 4.

IV-C. Punto 1:

Como se venía explicando en la seccion de diseo electróni-co, el punto 1 se implementó mediante rosserial entre ROS y el arduino. En donde ROS envia la información de las velocidades y los comandos de teclas mediante un nodo a arduino, mientras este último se encarga de activar los motores depediendo de las ordenes del usuario.

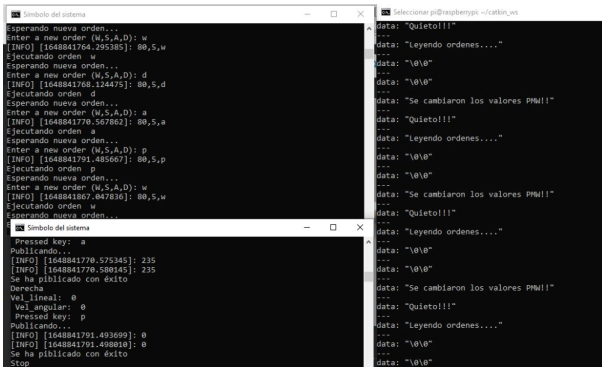

Figura 4. Publisher y subscriber del Arduino en funcionamiento

Para el desarrollo de este punto se realiza un nodo que permite al usuario mover al Robot en 4 direcciones utilizando 4 teclas por medio del tópico *teturtlebot_cmdVel*, en el cual se publican las velocidades lineales y angulares que el usuario especifica previamente en la consola (respetando los límites del robot).

En primer lugar, se considera un valor de tipo twist en el cual se tienen a su vez tres datos en linear y angular de manera correspondiente en los ejes x, y, z. Para el movimiento hacia adelante y hacia atrás se modificarán los datos linear.x, mientras que para los movimientos rotacionales hacia la derecha y la izquierda se modifican los datos angular.z.

En segundo lugar, la comunicación se establece publicando en el *teturtlebot_cmdVel* el valor twist, siendo este modificado por las funciones on_press y on_release para determinar los movimientos deseados y seguir los requerimientos. En particular, este emplea mediante la función on_press el módulo *keyboard* de la librería *pynput* para obtener un valor key y compararlo con las teclas definidas para el movimiento, teniendo así los siguientes comandos: arriba (w), abajo (s), izquierda (a), derecha (d). En cada caso se le asigna el valor de velocidad linear para (w) y (s), y la velocidad linear para (a) y (d). Es válido resaltar que tal decisión de la selección de las teclas tuvo en consideración la disposición de los teclados en ASCII para evitar posibles fallas de error en la replicación. Así mismo, la función de on_release a mayor detalle devuelve a cero los valores de velocidad lineal y angular, es decir, para retornar al estado inicial del robot y evitar que se mueva sin haber presionado el botón. Por lo que primero se publica el valor del twist en on_prees y luego el valor dado por on_release.

El funcionamiento y relación entre el nodo y el tópico se puede apreciar en la figura 5.

Figura 5. Grafo de ROS de CoppeliaSim y el nodo teleop

Este nodo sigue el proceso que se observa en las figuras 6 y 7, en las cuales se hizo la distinción entre el proceso de publicación y el de subscripción.

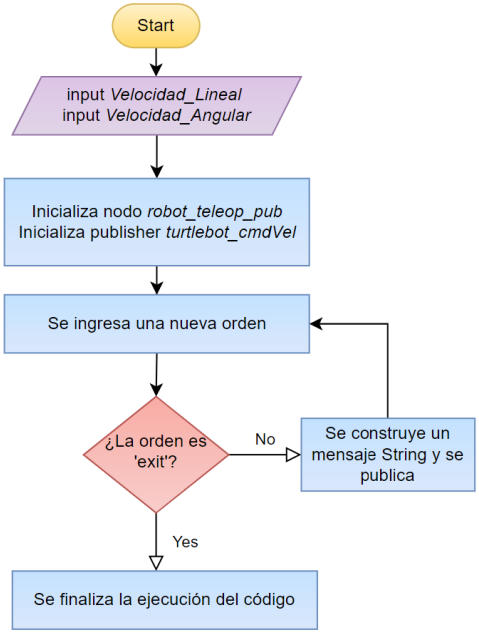

Figura 6. Diagrama de flujo del Publisher del punto 1

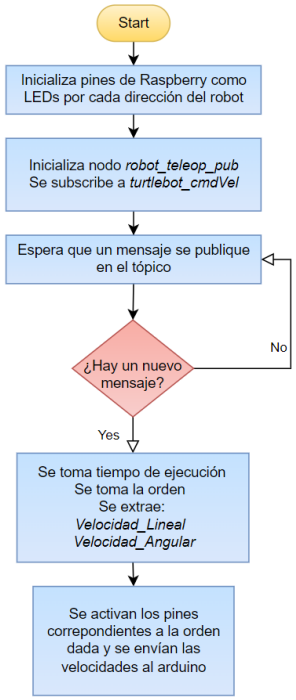

Figura 7. Diagrama de flujo del Subscriber del punto 1

IV-D. Punto 2:

En esta ocasión, se hace uso de la librería *matplotlib* con la cual es posible realizar gráficas en tiempo real utilizando las funciones *pyplot* y *FuncAnimation*. La posición es calculada a partir de la velocidad linear y angular de las ruedas, los datos (rpm) son adquiridos por medio de los encoder y transmitidos del arduino a la raspberry mediante el protocolo

de comunicación. Posteriormente, se crean dos arreglos de datos vacíos que se van llenando a medida que se registran los datos en las posiciones (x) y (y) en el marco inercial, la cual se calcula a partir de las ecuaciones que se presentan a continuación:

$$x(t) = x_o + \frac{1}{2}\int_0^t [V_r(t) + V_l(t)]\cos\big(\theta(t)\big)\,dt$$

$$y(t) = y_o + \frac{1}{2}\int_0^t [V_r(t) + V_l(t)]\sin\big(\theta(t)\big)\,dt$$

$$\theta(t) = \theta_o + \frac{1}{l}\int_0^t [V_r(t) - V_l(t)]dt$$

Esta posición, al igual que el recorrido del robot, se puede observar en la gráfica de la interfaz en tiempo real creando un esquema de la trayectoria recorrida mientras el robot se mueve. De igual manera, es válido resaltar que el comando autoscale_view() nos permite ir definiendo los contornos de la gráfica dependiendo la magnitud de los datos. Así mismo, se puede realizar la captura de la imagen presionando el botón del cassete abajo de la imagen, seleccionando el destino de este.

Finalmente, se presenta el diagrama de flujo correspondiente a este punto en la figura 8. En el cual se pueden observar dos métodos para el cálculo de la posición: el primero utiliza las velocidades que publica el nodo teleop, y el segundo utiliza las velocidades medidas por los encoders.

IV-E. Punto 3:

En el nodo *turtle_bot_interface_2.py* se inicializa el string del nombre del archivo para guardar la trayectoria y una bandera que indica que se desea realizar esta acción, estando esta conectada con el botón. Al ser presionado el mencionado botón, se le da el valor a este string con el texto ingresado en la caja de texto. Así mismo, se establece una comunicación basada en la publicación de una bandera indicativa de la acción en el tópico *turtlebot_Guardar_flag.py* y la publicación del nombre archivo deseado para guardar en el tópico *turtlebot_Guardar_name.py*.. La interfaz se muestra en la imagen de la Figura 9.

Así mismo, se modificó el nodo *turtle_bot_teleop.py* haciendo que este se subscriba a los tópicos mencionados anteriormente. Adicionalmente, se ubicó un condicional para abrir el documento con el nombre recibido de acuerdo con la bandera de la realización de la acción. Por último, se modificó on_press con un condicional adicional de la bandera indicativa para escribir en una nueva línea con el comando presionado asociado al movimiento.

A continuación, se puede evidenciar como al correr el nodo *turtle_bot_teleop.py* y el nodo *turtle_bot_interface_2.py*, el primero se subscribe a *turtlebot_Guardar_flag.py* y a *turtlebot_Guardar_name.py* para obtener los datos publicados por el segundo nodo. Por otro lado, se puede ver como el primero

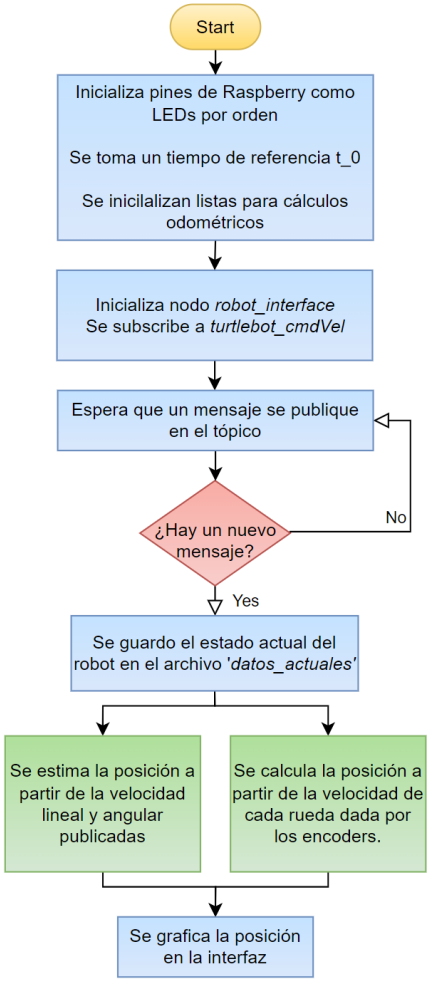

Figura 8. Diagrama de flujo del la interfaz

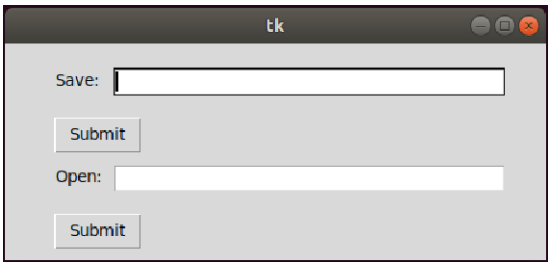

Figura 9. Interfaz gráfica turtle_bot_interface_2.py

nodo mencionado en este párrafo, sigue publicando el valor twist asociado a las velocidades a la interfaz.

Por otro lado, a continuación se muestra el diagrama de ROS en las circunstancias del anterior, sumado a correr la interfaz para la visualización gráfica de la posición de robot.

Finalmente, en la Figura 12 se muestra el diagrama de flujo asociado al funcionamiento y planteamiento de este punto. Así mismo, en la figura 7 se puede observar el Subscriber del nodo

Figura 10. Grafo de ROS de CoppeliaSim y los nodos interface 2 y teleop

Figura 11. Grafo de ROS de CoppeliaSim y los nodos interface.py, interface_2.py y teleop.py

turtle_bot_teleop.py que también se utiliza en este punto.

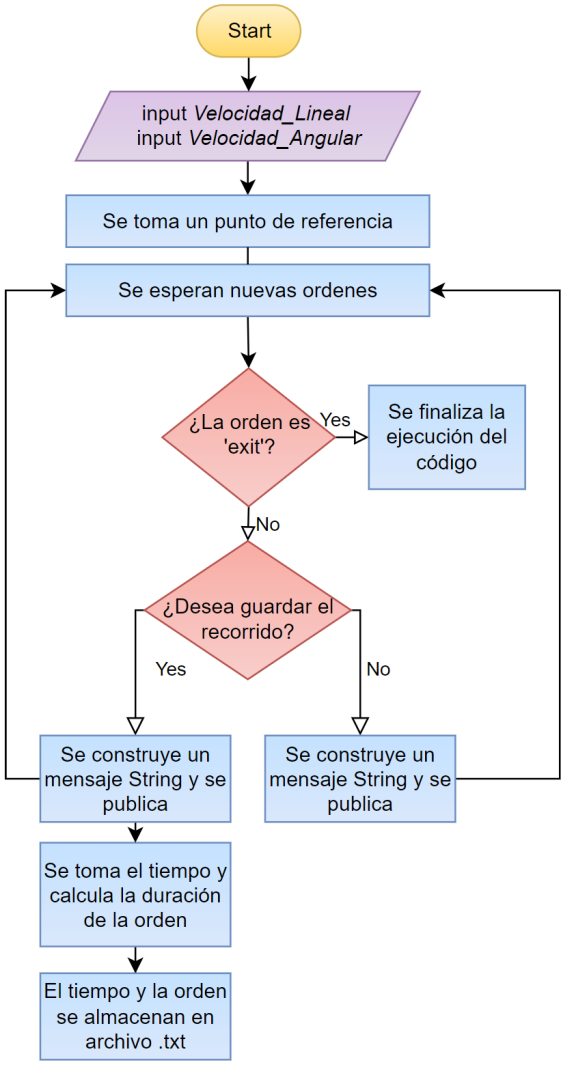

Figura 12. Diagrama de flujo del punto 3

IV-F. Punto 4:

Este punto requiere un esquema de un servicio para el cual se usaron los paquetes rospy o grometry_msgs, se utilizó ros para poder direccionar apropiadamente la información guardada en el un archivo .txt. Así, al principio se inicializó el nodo *turtle_bot_player* para posteriormente iniciar el servicio *sim_datos_user*. Así el sistema podía esperar que alguien utilizara el comando rosservice call para llamar dicho servicio y poder ejecutarlo. Este proceso puede ser descrito gráficamente como se puede ver en la figura 13.

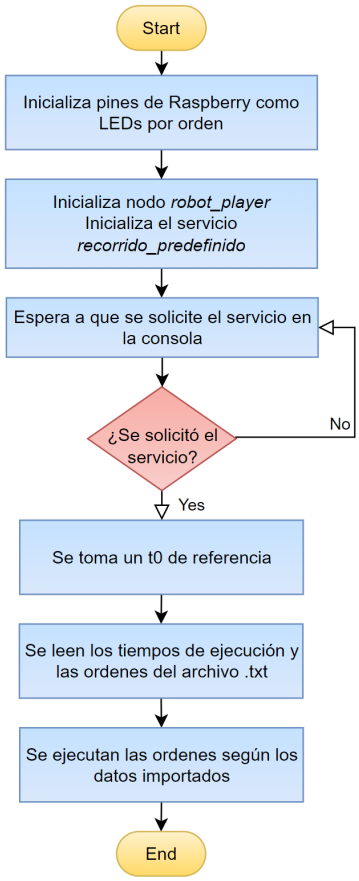

Figura 13. Diagrama de flujo del servicio

Así, cuando un usuario llamara el servicio se podría realizar una implementación muy similar a la realizada en el punto 1, solo que el origen de las ordenes no sería una entrada del teclado detectada sino una lista con dichas ordenes. Por ello, lo primero que se realiza al ser llamado el servicio es preciamente leer el .txt y almacenar los datos en una lista cuya estructura iterativa permite mantener la secuencialidad de las ordenes almacenadas.

Finalmente, se traducen dichas ordenes en diferentes velocidades lineales y angulares que para propósitos de esta implementación también constituyen parámetros del servicio. Así, se establece un publisher en el nodo **turtlebot_cmdVel** y se realiza el loginfo de la información. Esto se puede evidenciar en el rqt graph obtenido durante la simulación que se observa en la figura 14.

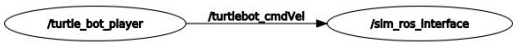

Figura 14. rqt graph del servicio que inicializó un nodo para enviarle información al robot

Los resultados vistos en el video permiten observar que la ejecución de ordenes secuencialmente a través de un archivo de texto permite un mejor manejo del robot, o a lo sumo disminuye el laggeo característico asociado a su propia simulación

V. Análisis de resultados

V-A. Diseño Mecánico:

El montaje del robot resultante se puede observar en la figura 15, en la cual se aprecia que el dimensionamiento realizado de forma virtual fue idóneo, ya que todos los componentes entran en su lugar correspondiente y el chasis es incluso de menor tamaño que el requerido en el enunciado del proyecto final. Adicionalmente, el material utilizado (MDF) es lo suficientemente resistente para soportar el peso de todos los componentes.

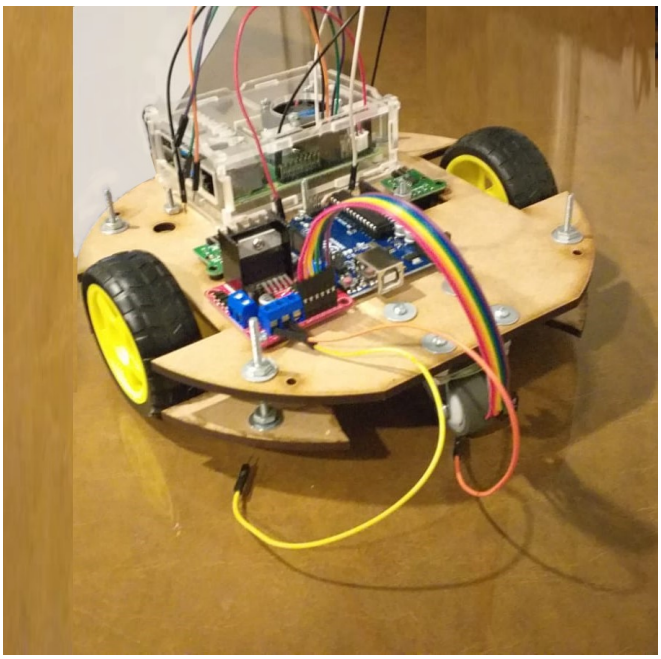

Figura 15. Montaje físico del robot

V-B. Diseño Electrónico:

El diseño electrónico fue exitoso ya que permite una comunicación adecuada entre componentes, lo cual se comprobó utilizando distintos códigos para accionar los motores, cambiar sus direcciones y leer la información de los encoders. De esta forma se garantiza que cualquier falla en el funcionamiento del robot no se deberá a las conexiones ni a la comunicación entre componentes.

V-C. Punto 1:

Al implementar los códigos en la raspberry, se evidencia que al oprimir la tecla 'w' el robot va hacia adelante, con 's' va hacia atrás y con las teclas 'a' y 'd' rota sobre el eje z hacia la izquierda y la derecha respectivamente. Igualmente, se observa que para los movimientos mencionados, se requiere presionar repetidas veces o dejar presionado el botón en cuestión, dando cuenta de cumplimiento del requerimiento de estado estático del robot cuando no se presiona ningún botón. Así mismo, lo anterior comprueba el funcionamiento correcto del nodo *turtle_bot_2* y el tópico respectivo a las velocidades del robot *teturtlebot_cmdVel*.

V-D. Punto 2:

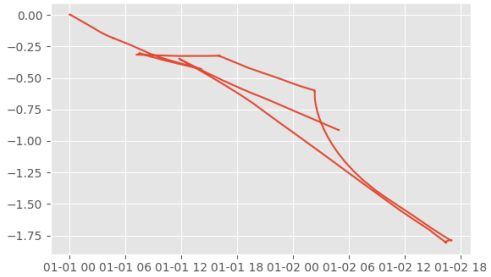

Figura 16. Gráfica de la trayectoria recorrida en la prueba

El código realizado calcula efectivamente de la posición del robot en tiempo real gracias a las ecuaciones de cinemática directa utilizadas con los datos de velocidad de los encoders, lo cual garantiza que la velocidad utilizada en los cálculos sea más cercana a la real y el error de la posición calculada sea despreciable. Sin embargo, viualización de dicha posición es algo más complejo de implementar debido a que la raspberry no posee una pantalla como tal y en algunas ocasiones al conectarla a un monitor externo no se puede apreciar la interfaz realizada en el código. De igual manera, la traficación es la apropiada para el seguimiento de la trayectoria del robot, tal como se observa en la figura 16. Así mismo, se considera valiosa la escala autorregulada por la magnitud de los datos para la mejor visualización de estos. Debido a este último aspecto, se pudo identificar que el robot tiene un ligero movimiento al estar aparentemente estático, que es despreciable al realizar el movimiento normal por los comandos.

V-E. Punto 3:

En esta ocasión se probó la opción de Guardar un recorrido, lo cual generó un nuevo archivo tipo .txt nombrado a partir del input ingresado en la interfaz y que contenía los comandos ingresados posteriormente mediante el nodo *turtle_bot_teleop*. Los resultados de este apartado demuestran principalmente la utilidad de los threads para interconectar nodos y compartir información entre ellos. Sin embargo, aun cuando la interfaz nueva ópera correctamente, sería más amigable para el usuario que estas funciones se encontraran en la misma interfaz donde se gráfica la posición del robot.

V-F. Punto 4:

Se logró la implementación del servicio a través de la Raspberry, dado que esta ya tenía instalado ROS de manera tal que se puede leer el archivo de texto directamente y la Raspberry puede ejecutar las ordenes almacenadas allí. En la medida que se logró la implementación de los servicios, relacionarlos con nodos y ejecutar una función como la previamente planteada, se comprendió que los servicios al igual que los nodos son herramientas muy potentes capaces de ejecutar una gran variedad de funciones, con la diferencia de que los primeros requieren ser llamados desde consola y los segundos solo desde un script.

VI. Conclusión

- Con la realización de este taller fue posible comprender mejor los alcances de una Raspberry y su utilidad en proyectos de todo tipo, pero en especial en robótica gracias a su versatilidad y su tamaño compacto. En este caso fue posible instalarle ROS para así utilizar los códigos previamente realizados en el taller 1.
- Al indagar sobre los protocolos de comunicación posibles entre la Raspberry y el Arduino se encontraron gran variedad de alternativas de las cuales siempre hay una más adecuada para los requerimientos del proyecto. Debido a esto se aprendió que aunque existen varias soluciones de intercomunicación es mejor utilizar la que se adecue mejor a lo que se necesita y a los conocimientos de quién lo va a implementar.
- Se logró una mayor comprensión del framework ROS, el cual al ser de código abierto permite el uso compartido de librerías y módulos para así crear un intercambio de conocimiento en el ámbito de robótica. De esta manera, es posible resolver desafíos cada vez más avanzados en lugar de que cada equipo deba realizar los mismos trabajos desde cero.
- Se comprendió la importancia del manejo de los threads para el desarrollo de procesos simultáneos, los cuales nos permitieron el manejo de información compartida en los diversos nodos, de esta forma optimizando los recursos disponibles de la raspberry. Por otro lado, ante la secuencialidad de arduino, se resalta la importancia del tiempo y sincronía del muestreo de los sensores y el envío de datos, además del posible impacto del código en la ejecución de las tareas.
- Se modificaron nodos para preguntar al usuario en la interfaz si desea guardar el recorrido del robot y posteriormente guardar en un archivo .txt la secuencia de acciones que realizó el usuario durante el recorrido del robot. Es válido resaltar que el nodo fue adecuado de la versión pasada de este informe, dando muestra de la validez de la simulación y la eficacia de la metodología implementada.
- Se creó un nodo de ROS que a partir de un archivo .txt de un recorrido guardado, es capaz de reproducir la secuencia de acciones del robot. Es válido resaltar que el nodo fue adecuado de la versión pasada de este informe, dando muestra de la validez de la simulación y la eficacia de la metodología implementada.
- Fue comprendida y aplicada de manera efectiva la comunicación entre nodos mediantes tópicos, así como el uso servicios para realizar acciones tales como la lectura y publicación de datos. Tal cometido, realizado en primer lugar con la simulación y posteriormente, a mayor escala, con la implementación práctica en el diseño de la comunicación bidireccional tácita en el envío de la información de los sensores y los parámetros del movimiento.
- El diseño y modelamiento de los componentes para la modificación del diseño se considera valiosa, en tanto a la disminución de las medidas contribuye de igual manera a la navegación del robot, y la practicidad de conexión permiten una orden mayor con el cableado con ventajas en el análisis visual del desprendimiento de algún componente o conexión.
- La comunicación implementada por rosserial supone una gran ventaja al permitir realizar publicaciones a tópicos desde el arduino, así como la obtención de información al subscribirse a uno. Tal modelo permitió una mayor facilidad en la adecuación del sistema previamente implementado. Así mismo, al estar conectado mediante el cable serial, el arduino puede obtener información y carga de la raspberry.

Referencias

[1] García,J. Enunciado del Taller 2. Clase de Robótica 2022-1

[2] ROS Wiki. Rosserial. Open Robotics. [Online] recuperado de: http://wiki.ros.org/rosserial

[3] ROS Wiki. Arduino IDE Setup - Rosserial. [Online] recuperado de: http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup