

Informe Taller 3

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

3rd Laura Milena Gamba Sánchez

Departamento de Ingeniería Eléctrica y Electrónica
Universidad de los Andes
Bogotá, Colombia
lm.gamba@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

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 tercer taller se diseñó el modelo mecánico y electrónico de un robot diferencial con un brazo en su parte superior y una cámara en su cara frontal. Para tal cometido, se emplearon nodos y tópicos empleados para el primer y segundo taller. De forma tal que, desde el teclado, se permite el control del brazo robótico y el control sobre el movimiento del carro, dando la posibilidad de seguir la trayectoria visualmente. Así mismo, la cámara implementada es capaz de detectar elementos del entorno de tonalidades específicas, controlando el brazo para poder agarrar el objeto.

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

- Crear 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.
- Crear un nodo en ROS, llamado que permita a un usuario controlar por teclado el robot. Es decir, que las velocidades de cada una de las junturas que tiene su robot se puedan controlar de forma directa con las teclas del computador.
- Crear un nodo de ROS, que permita visualizar la posición en tiempo real del endeffector de su robot, a través de una interfaz gráfica.
- Cree un nodo en ROS que permita a un usuario llevar el end-effector del robot a una posición destino deseada.
-

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 proporcionaron mayor estabilidad, hasta obtener la lista completa que se observa en la tabla I.

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 19x19cm de 4 niveles, entre los cuales se encontrarán los motores, y con espacios a los costados para las ruedas de 3cm de ancho. Entre el primer y segundo nivel, partiendo de abajo para arriba, se

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-voltaje-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-recargable-litio-lipo-74v-1500-mah-arduino-robot-JM
Brazo Robótico	4 grados de libertad	https://articulo.mercadolibre.com.co/MCO-854877874-brazo-robotico-arduino-JM#position=18&search_layout=stack&type=item&tracking_id=c0115ff9-aace-4fdc-8792-edd3f35432dd
Servo-motores	SG90	https://electronilab.co/tienda/micro-servo-9g-towerpro/
Camara Web	Raspberry Pi 5mpx	https://articulo.mercadolibre.com.co/MCO-464854164-camara-raspberry-pi-5n-JM#position=2&search_layout=stack&type=item&tracking_id=06dcae5b-429c-4d6e-b698-c40b5f9094f8

Tabla I

LISTA DE MATERIALES CON REFERENCIA

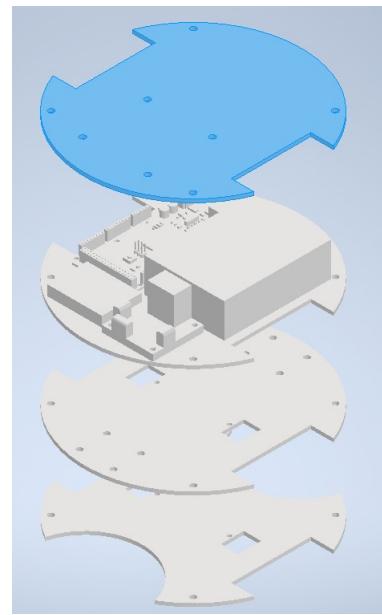


Figura 1. Diseño mecánico del robot

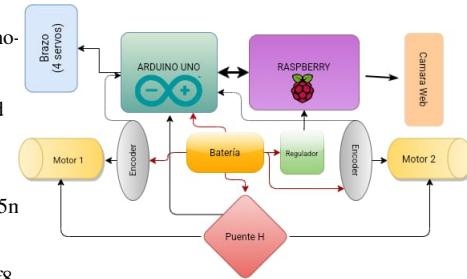


Figura 2. Diagrama general del diseño electrónico

encuentran los motores, las ruedas de apoyo y los encoder. En el tercer nivel se encuentran los componentes y el circuito del robot. Finalmente, en el cuarto nivel se encuentra el brazo robótico. Así mismo, la cámara está ubicada entre la raspberry y este último nivel.

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 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 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.

A continuación se presenta e diagrama circuital vinculado al brazo robótico. Es válido mencionar que la cámara web está conectado al puerto usb de la raspberry pi 4.

IV-C. Movimiento traslacional y rotacional del carro:

Como se venía explicando en la sección de diseño electrónico, se implementó mediante rosserial entre ROS y el arduino. En donde ROS envía la información de las velocidades y los comandos de las teclas mediante un nodo a arduino, mientras este último se encarga de activar los motores dependiendo de las órdenes del usuario.

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 nodo *robot_manipulator_teleop*. Para

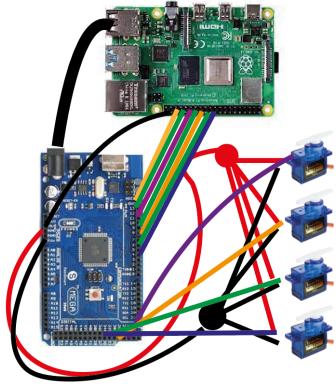


Figura 3. Diagrama general del diseño electrónico

el movimiento hacia adelante y hacia atrás se modificarán los datos msg.x, mientras que para los movimientos rotacionales hacia la derecha y la izquierda se modifican los datos angular.z.

Luego, la comunicación se establece publicando en el *robot_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 7 teclas para el movimiento de los 4 servos. 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_press* y luego el valor dado por *on_release*.

El funcionamiento y relación entre el nodo y el tópico se puede apreciar en la figura 4.



Figura 4. rqt graph del punto 1

IV-D. Visualización gráfica de la posición del robot:

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 lineal 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

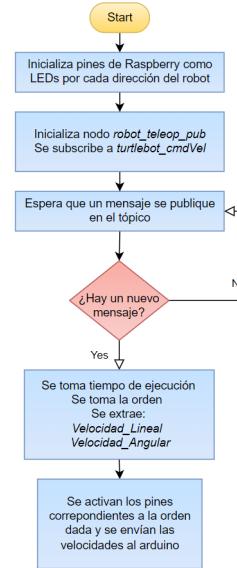


Figura 5. Diagrama de flujo del Subscriber del punto 1

cual se calcula a partir de las ecuaciones que se presentan a continuación:

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

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

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

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 cassette abajo de la imagen, seleccionando el destino de este.

Finalmente, se presenta el diagrama de flujo correspondiente a este punto en la figura 6. 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. Almacenamiento de recorrido:

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

interfaz para la visualización gráfica de la posición de robot.

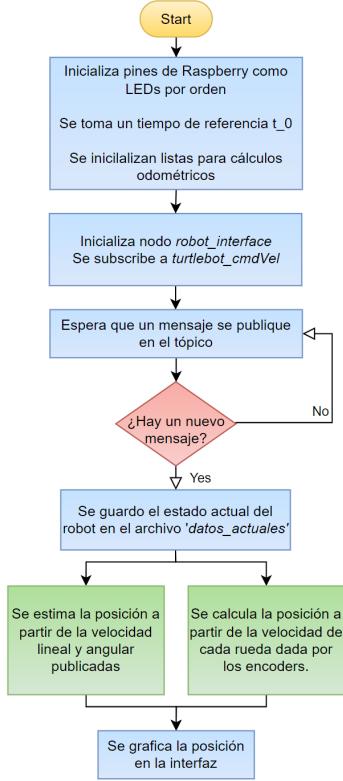


Figura 6. Diagrama de flujo del la interfaz

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*.

Así mismo, se modificó el nodo *turtle_bot_teleop.py* haciendo que este se suscriba 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 suscribe 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 nodo mencionado en este párrafo, sigue publicando el valor *twist* asociado a las velocidades a la interfaz.

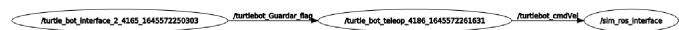


Figura 7. Grafo de ROS de CoppeliaSim y los nodos interface 2 y teleop

Por otro lado, a continuación se muestra el diagrama de ROS en las circunstancias del anterior, sumado a correr la

Figura 8. Grafo de ROS de CoppeliaSim y los nodos *interface.py*, *interface_2.py* y *teleop.py*

Finalmente, en la Figura 9 se muestra el diagrama de flujo asociado al funcionamiento y planteamiento de este punto. Así mismo, en la figura 5 se puede observar el Subscriber del nodo *turtle_bot_teleop.py* que también se utiliza en este punto.

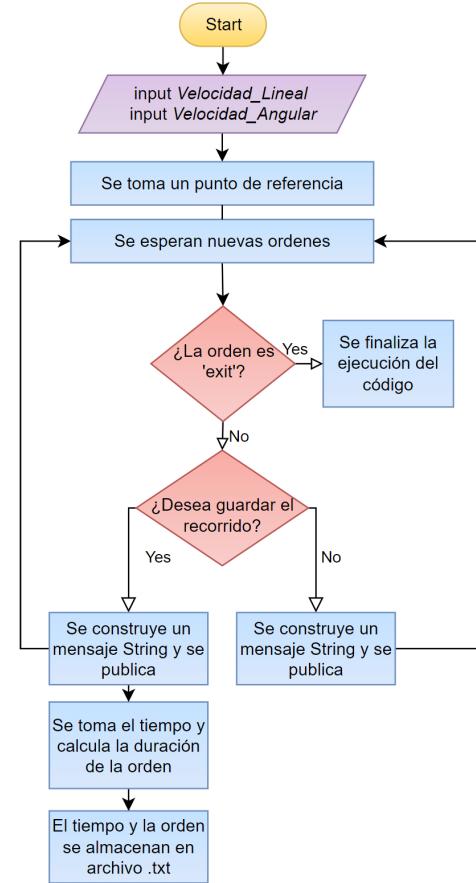


Figura 9. Diagrama de flujo del punto 3

IV-F. Reproducción de recorrido almacenado:

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 utilizará el comando *rosservice call* para llamar dicho servicio y poder ejecutarlo. Este proceso puede ser descrito gráficamente como se puede ver en la figura 17.

Así, cuando un usuario llamará el servicio se podrá realizar una implementación muy similar a la realizada en el punto

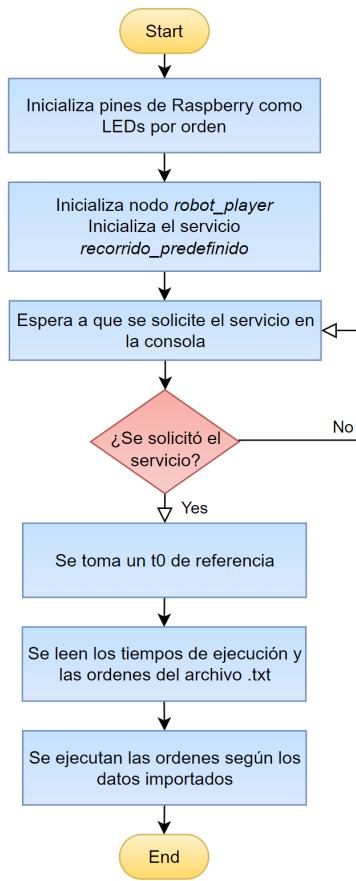


Figura 10. Diagrama de flujo del servicio

1, solo que el origen de las órdenes no sería una entrada del teclado detectada sino una lista con dichas órdenes. Por ello, lo primero que se realiza al ser llamado el servicio es precisamente leer el .txt y almacenar los datos en una lista cuya estructura iterativa permite mantener la secuencialidad de las órdenes almacenadas.

Finalmente, se traducen dichas órdenes 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 11.



Figura 11. 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 órdenes secuencialmente a través de un archivo de texto permite un mejor manejo del robot, o a lo sumo dismi-

nuye el laggeo característico asociado a su propia simulación

IV-G. Punto 1:

Se implementó la comunicación mediante rosserial entre ROS y el arduino. En donde la Raspberry publica la información de la variación de ángulos deseados para cada servomotor con respecto a la posición actual. El Arduino, por su parte, se subscribe a este tópico, y mediante la librería Servo.h, se encarga de enviar un tren de pulsos tal que el desplazamiento rotacional es el esperado. Adicionalmente, se suaviza este movimiento, determinando una cantidad de pasos para cumplir las variaciones angulares deseadas en cada servomotor. Para usar la librería, es necesario calcular la variación angular requerida para cada servomotor e ingresarla a un método que discrimina que señal enviar a cada motor, por lo que si no se quiere que se muevan, esta variación debe ser cero.

A continuación se presentan los planos de las partes que componen el brazo robótico implementado:

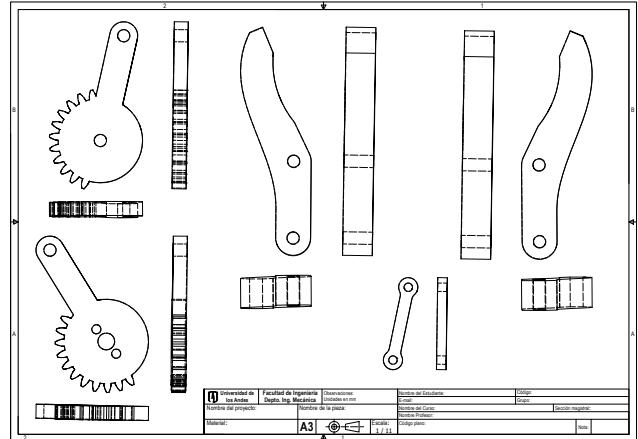


Figura 12. Partes ensamblaje de brazo robótico.

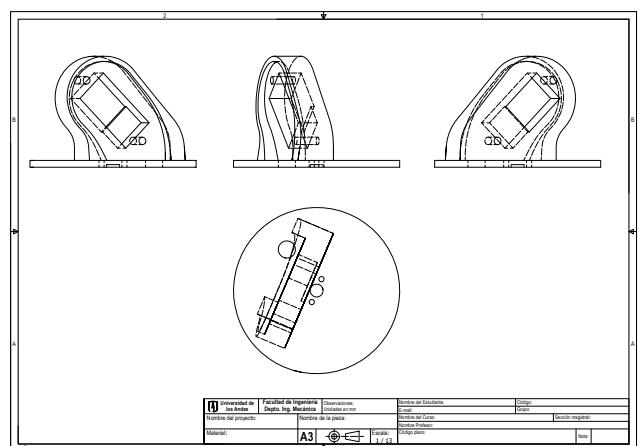


Figura 13. Partes ensamblaje de brazo robótico.

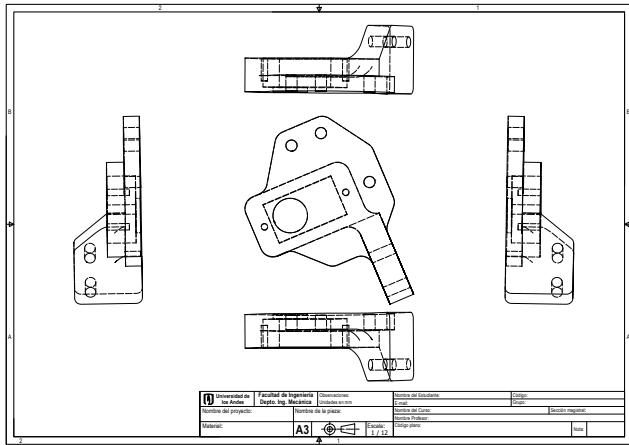


Figura 14. Partes ensamblaje de brazo robótico.

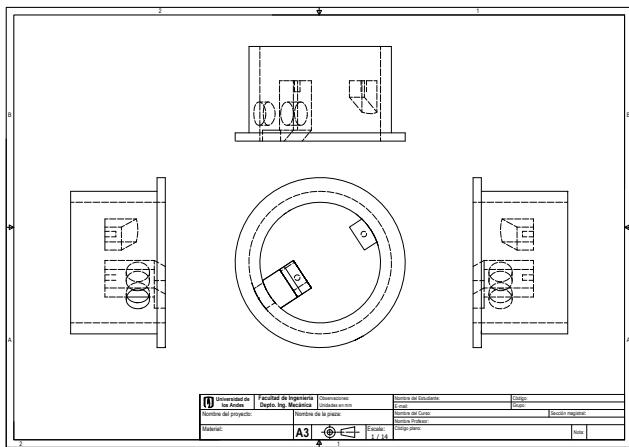


Figura 15. Partes ensamblaje de brazo robótico.

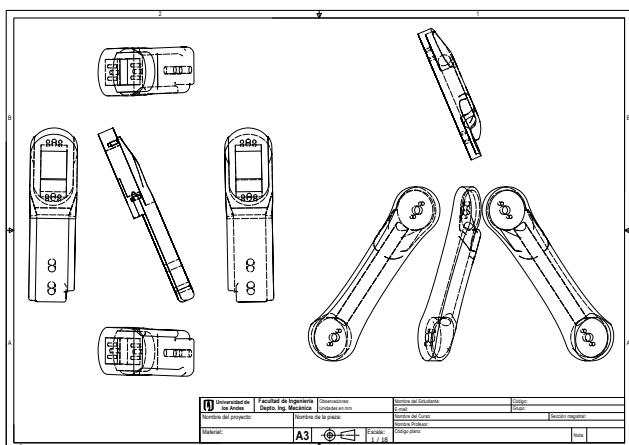


Figura 16. Partes ensamblaje de brazo robótico.

mediante la aplicación de Matplotlib. A partir de la cinemática directa, es posible, determinar la posición de la punta de la pinza con los valores actuales de los ángulos comprendidos entre las junturas, las fórmulas empleadas se muestran a continuación:

$$x = a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) \quad (1)$$

$$y = a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) \quad (2)$$

A continuación, se presenta el diagrama de flujo planteado. En este se integra el funcionamiento del punto anterior, identificación de comando y publicación de datos, junto a la visualización de la posición del brazo robótica. En este se plantean tres escenarios que desencadenan movimiento del actuador: Oprimir teclas de movimiento del brazo, ingreso de posición final deseada, y detección de objetivo por la cámara mediante análisis visual. Para el segundo y tercer caso, se requiere el cálculo por cinemática inversa para hallar los ángulos. Posteriormente, estos se publicarán y graficarán.

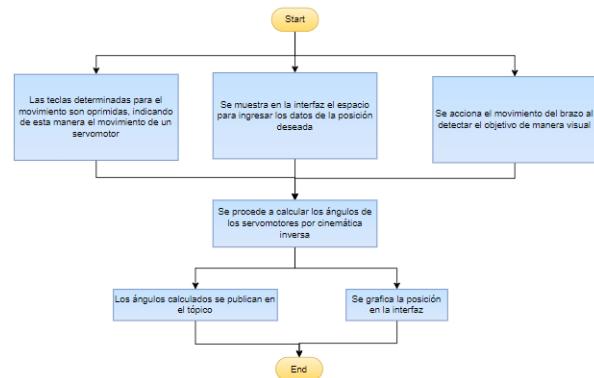


Figura 17. Diagrama de flujo de visualización y publicación de datos.

IV-I. Punto 3:

Adecuando la cinemática inversa a las dimensiones y naturaleza del brazo robótico, se establece la longitud de cada juntura, los marcos locales y los ángulos entre estos, implementando, finalmente, las siguientes ecuaciones a partir de la posición deseada ingresada por la interfaz. Las fórmulas empleadas, para determinar los ángulos en 2D, se muestran a continuación, suponiendo que el robot está en dirección al punto de destino.

$$\cos(\theta_2) = \frac{1}{2a_1a_2}((x^2 + y^2) - (a_1^2 + a_2^2)) \quad (3)$$

$$\sin(\theta_2) = \pm \sqrt{1 - \cos(\theta_2)^2} \quad (4)$$

$$\text{Cos}(\theta_1) = \frac{1}{x^2 + y^2} (x(a_1 + a_2 \text{Cos}(\theta_2)) \pm y a_2 \sqrt{1 - \text{Cos}(\theta_2)^2}) \quad (5)$$

$$\sin(\theta_1) = \frac{1}{x^2 + y^2} (y(a_1 + a_2 \cos(\theta_2)) \pm x a_2 \sqrt{1 - \cos(\theta_2)^2}) \quad (6)$$

Es claro que la posición deseada debe estar dentro del **volumen de trabajo** del manipulador, el cual se resume como una semiesfera de 16.5cm de radio a partir del primer link, y nos más abajo de la altura de la plataforma donde se encuentra el brazo debido a la fuerza que requeriría para volver a levantarse. El volumen de trabajo se puede apreciar en las Figuras 18 y 19

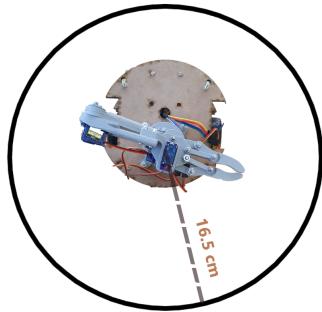


Figura 18. Vista superior del Volumen de trabajo

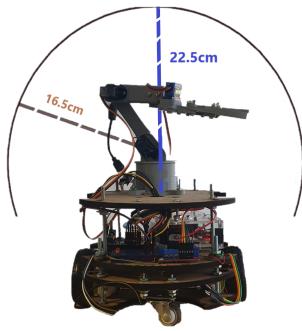


Figura 19. Vista frontal del Volumen de trabajo

IV-J. Punto 4:

Para realizar la identificación de colores y siluetas, se realizó una conversión de color RGB a HSV, con lo cual se facilita diferenciar los colores gracias al parámetro Hue y así es posible establecer un rango de valores para que la cámara los agrupe dentro de un color. La calibración de la cámara se realizó con las imágenes de las pelotas tomadas en el laboratorio.

V. ANÁLISIS DE RESULTADOS

V-A. Diseño Mecánico:

El montaje del robot resultante se puede observar en la figura 21, en la cual se aprecia que el dimensionamiento



Figura 20. Imagen utilizada para la calibración colores con la cámara Web

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 21. 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. Movimiento translacional y rotacional del carro:

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 *turtlebot_cmdVel*.

V-D. Visualización gráfica de la posición del robot:

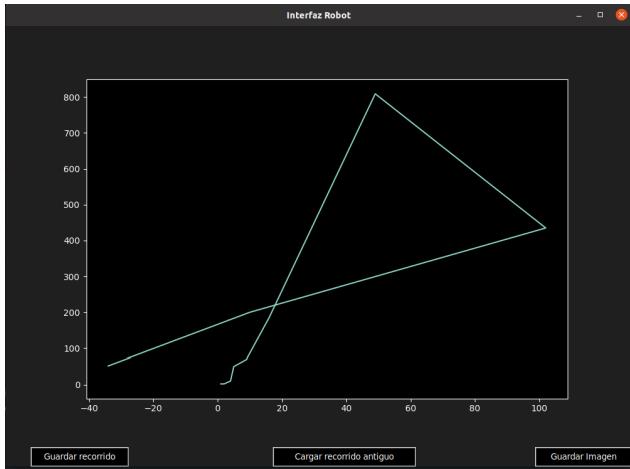


Figura 22. Gráfica de la trayectoria recorrida

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, visualización 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 graficación es la apropiada para el seguimiento de la trayectoria del robot, tal como se observa en la figura 22. 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. Almacenamiento de recorrido:

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 opera correctamente, sería más amigable para el usuario que estas funciones se encontraran en la misma interfaz donde se gráfica la posición del robot.

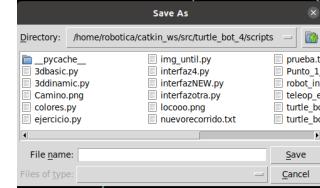


Figura 23. Cuadro de diálogo que pregunta el nombre del recorrido a guardar

V-F. Reproducción de recorrido almacenado:

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

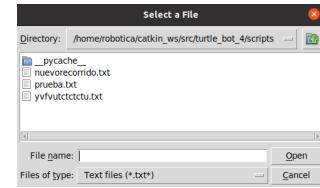


Figura 24. Cuadro de diálogo que pregunta el nombre del recorrido que se va abrir

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.

V-G. Punto 1: Movimiento del manipulador

V-H. Punto 2: Visualización gráfica de la posición del manipulador

V-I. Punto 3:

La posición deseada se puede ingresar desde la interfaz para el posterior cálculo de trayectorias. En la figura 26 se aprecia la ventana donde se entra al posición deseada.

V-J. Punto 4:

Se logró el efectivo reconocimiento de las pelotas, identificando el color de las tres, así como su contorno y centro de este. De acuerdo con este último aspecto, se controló de forma reactiva que el centro de la pinza coincidiese con el centro del contorno de la pelota del color deseado. Con respecto a la profundidad, es decir, el eje y, se toma un valor fijo de acuerdo con las dimensiones de la guía. De igual manera, la altura se ajustó con el mismo planteamiento.

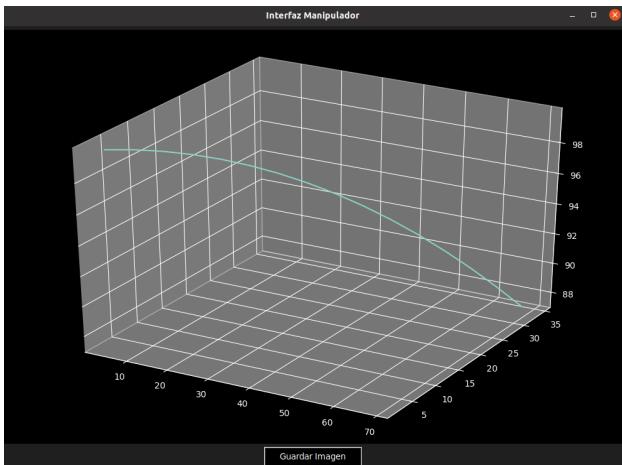


Figura 25. Gráfica de la trayectoria del manipulador

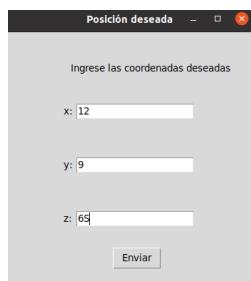


Figura 26. Imagen de la ventana donde se ingresa la posición deseada

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 acomode 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 mediante 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 suscribirse 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