Taller 2, Diseño de un robot diferencial - IELE3338

Santiago Rodríguez Ávila - *s.rodrigueza@uniandes.edu.co*
Sergio Andrés Guerra Sánchez - *sa.guerra20@uniandes.edu.co*
Santiago Muñoz Correa- *s.munozc20@uniandes.edu.co*
Isabela Buitrón Burbano - *i.buitron@uniandes.edu.co*

Universidad de los Andes
Departamento de Ingeniería Eléctrica y Electrónica

02 de abril de 2022

Introducción

En este taller se presenta el proceso de diseño de un robot diferencial u omnidireccional que permita moverse y comunicarse a través de ROS. En este documento se presenta el diseño realizado con planos mecánicos y electrónicos acompañados de las medidas generales de su robot. El robot debe ser de 20x25 cm.

1. Diseño del Robot

1.1. Lista de Materiales:

#	Componente	Referencia	Link de referencia	Precio und. (COP$)
2	Motorreductor 200RPM	Dagu MOT_0458	https://tienda.tdrobotica.co/chasis-ruedas/458-motorreductor-200rpm.html	5.000 (ambos)
2	Rueda con neumático de goma	Dagu MEC_0457	https://tienda.tdrobotica.co/chasis-ruedas/457-par-de-ruedas-con-neumatico-de-goma-65mm-.html	8.000 (ambas)
3	MDF 2mm	N/A	N/A	3.000 + 6.000 x3
1	Módulo Driver L298	Pololu MOT_0544	https://tienda.tdrobotica.co/categoria/544-modulo-driver-l298.html	10.350
1	Ball Caster	Pololu MEC_0155	https://tienda.tdrobotica.co/chasis-ruedas/263-ball-caster-34-plastico-pololu.html	9.700
2	Sensor Encoder Óptico Hc-020k	B83609	N/A	15.000 x2
1	Arduino UNO		N/A	N/A
1	Raspberry Pi 4		N/A	N/A
1	Protoboard	ACC_0429	https://tienda.tdrobotica.co/protoboards-cajas-soportes/429-protoboard-pequena.html	6.000
1	Módulo regulador LM2596		N/A	8.700
2	batería lipo 5000MAH 7.4V		N/A	N/A
				104.750

Para la alimentación del robot se usa una batería lipo 5000MAH 7.4V con regulador como alimentación de la tarjeta Raspberry y para los motores se usa una batería lipo de 800mAh 7.4V. Los elementos que se incluyeron en la lista de materiales sin precio Se incluyeron las referencias de ciertos componentes dentro de la bibliografía al final del presente documento. En la Figura 2 de la sección 1.2 Plano Mecánico se puede visualizar la estimación de espacio que ocupa la lista de materiales dentro del chasis diseñado para el robot.

1.2. Plano Mecánico:

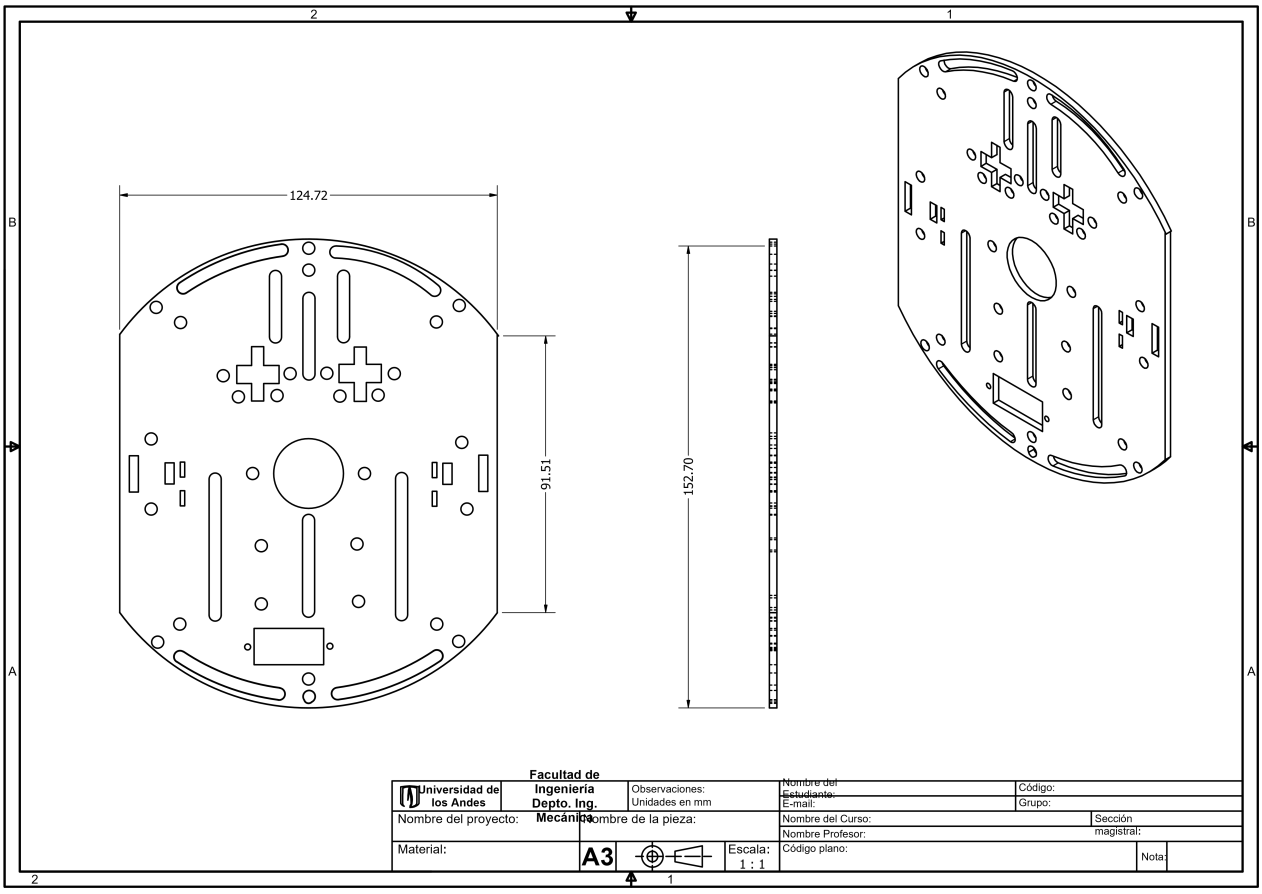

Figura 1: Plano mecánico.

Se propone un modelo compacto casi circular de tal manera que el centro de masa del robot sea más fácil de determinar.

1.3. Plano Eléctrico/Electrónico:

A continuación se encuentra el plano eléctrico/electrónico de las conexiones entre los componentes. Se destaca la conexión de los encoders y los motores que deben ir en los pines 2 y 3 tipo PWM para poder realizar la odometría.

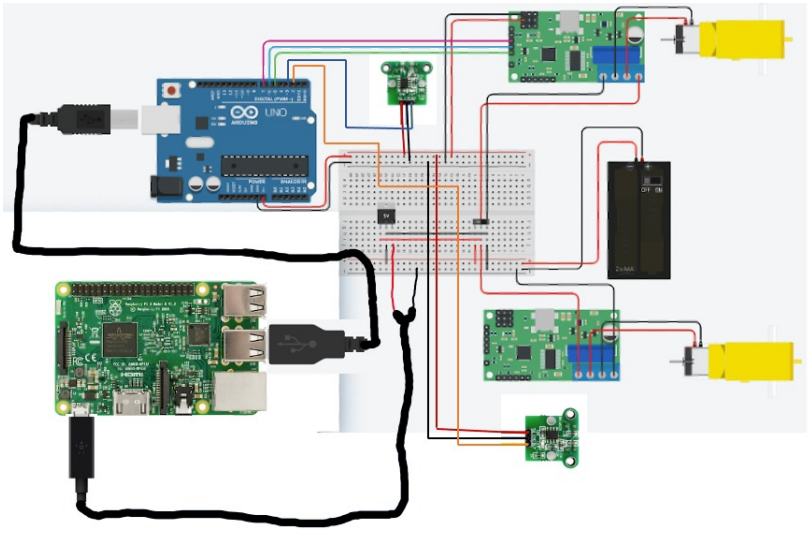

Figura 2: Plano Eléctrico

1.4. Integración Detallada de la Parte Eléctrica y Mecánica:

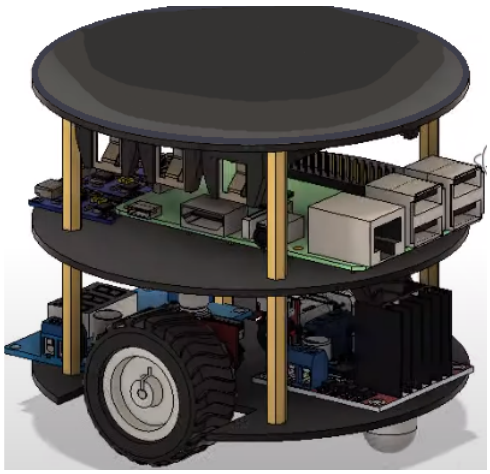

Figura 3: Vista simulada del robot físico.

Se realizó una vista simulada de los componentes que se incluyeron en el robot como llantas, motores, puente H, Raspberry y Arduino para determinar que los espacios fueran los suficientes para soportar todos los componentes que requiere el robot para funcionar. La parte mecánica, el chasis, se integra con la parte eléctrica en los orificios que se tuvieron en cuenta para pasar cables entre pisos. Adicionalmente, los motores fueron fijados con el primer y segundo módulo del chasis para evitar que se desplazen cuando se mueva el robot. La conexión entre pisos se realizó con espaciadores modulares para ajustar los pisos del chasis al espacio de altura necesario.

1.5. Descripción del Funcionamiento del Robot:

El robot realizado para el desarrollo de este taller consiste de dos motores DC con sus llantas. Se utilizó Arduino para realizar las tareas de movimiento del robot y, por lo tanto, se hizo la integración Arduino - ROS a través del nodo /serial_node. Como se detallará en la siguiente sección de descripción de los nodos, el robot se mueve hacia adelante, atrás, derecha o izquierda por medio de las teclas W, S, A o D que le ingresan por parámetro y se ajusta a la velocidad lineal y angular ingresada por parámetro al iniciar. La llanta ball caster brindan un tercer apoyo para el robot diferencial facilitando el movimiento hacia los lados. El robot puede recibir también las instrucciones de movimiento a través de un documento txt cargado por el usuario y se realiza una gráfica en tiempo real de su posición conforme se mueve. Para su funcionamiento, se usa una batería lipo 5000MAH 7.4V con regulador para la alimentación de la tarjeta Raspberry y para los motores se usa una batería lipo de 800mAh 7.4V.

2. Solución

2.1. Desarrollo de los Puntos

2.1.1. Explicación de nodos

El grupo crea los nodos necesarios para desarrollar cada punto del taller. Se puede ver la secuencia de comandos usados para ejecutar la solución.

El grupo describe en el documento de entrega los resultados obtenidos en cada punto usando gráficas y fotos (entre otras) que demuestran el funcionamiento esperado.

- **Punto 1:** para ejecutar la solución del punto 1 se deben abrir tres terminales. En una terminal ejecutar el comando *roscore* y en la otra *rosrun rosserial_python serial_node.py /dev/ttyACM0.* Se ejecuta el comando: *cd catkin_ws/*, para ingresar al workspace de robótica y en la misma terminal se escribe el comando *. devel/setup.bash* para otorgar permisos de ejecución, seguido de *rosrun mi_robot_3 prueba_arduinoROS.py* (este último es el archivo utilizado para iniciar el funcionamiento de los motores desde Arduino y comunicarlo con ROS). Lo anterior abre el paquete del robot de grupo 3 e inicializa el nodo de teleop solicitado. Una vez realizados estos pasos, es posible manejar la dirección del robot desde las teclas W, S, A y D en la terminal y se podrá ver el movimiento físico en el robot.

  La velocidad lineal y angular ingresan por parámetro en la terminal. Con la librería pynput se lee el teclado y cada vez que se oprima una de las teclas de movimiento, el nodo teleop publica en el tópico turtlebot_cmdVel el valor que ingresa por parámetro. Ejecutando el comando *rostopic info /turtlebot_cmdVel* podemos ver que el tipo de mensaje es *std_msgs/Int8*. La razón de lo anterior es debido a que el funcionamiento del movimiento del robot consiste en que Arduino reciba las

letras del teclado y las traduce a movimiento en los motores, dentro del código de Arduino las letras W, S, A y D se traducen a números "1", "2", "3" y "4", que corresponden a los movimientos de ir hacia adelante, atrás, derecha e izquierda y envían a los motores el valor PWM correspondiente. Se realiza la traducción con un mensaje de tipo Int8, se usan mensajes tipo Twist para cmdVel pues tiene un componente lineal para las velocidades (x,y,z) y un componente angular en los ejes (x,y,z). En un terminal con el workspace de robótica abierto se escribe el comando *rqt_graph*. rqt_graph proporciona un complemento GUI (interfaz de usuario gráfica) para visualizar el gráfico de cálculo de ROS. En el gráfico, entre más gruesas las líneas, significa que la conexión tiene un gran rendimiento en Bytes/s.

- **Descripción nodos /turtlebot_teleop y /serial_node:**

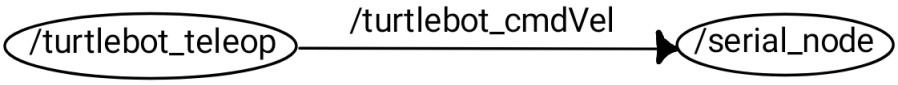

Figura 4: Grafo rqt ROS punto 1.

Se observa que el nodo teleop se comunica con los motores del robot a través del tópico cmdVel por medio del cual comunica las velocidades lineales y angulares para el movimiento el robot a la comunicación Arduino-ROS creada que se expresa en el nodo */serial_node*, como se dijo anteriormente, se usan mensajes tipo Twist. Para visualizar el sistema se usó el gráfico rqt. Lo anterior permite tomar mejores decisiones para la integración de los demás puntos del taller pues de presentarse un error de comunicación entre los nodos, se podrá detectar fácilmente el problema.

- **Punto 2:**
- **Resultados:** en una terminal ejecutar el comando *rosrun rosserial_python*source devel/setup.bash. Luego, en la misma terminal, correr rosrun turtle_bot_3 tutlebot_interface.py. Una vez ejecutados estos comandos se abrirá la interfaz que permite visualizar la posición en tiempo real del robot.

Figura 5: Visualización página 1 interfaz.

Al seleccionar página de gráfica, se visualizará el movimiento del robot en tiempo real (los comandos de movimiento los ejecuta el usuario a través de la terminal con las mismas teclas del punto anterior (W, S, A y D).

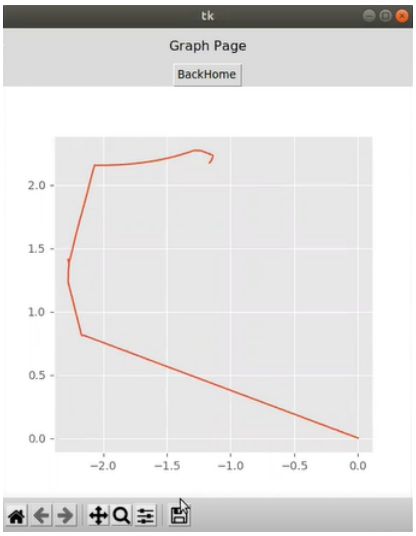

Figura 6: Visualización en tiempo real del movimiento de turtlebot desde la interfaz.

Una vez se haya realizado el recorrido deseado, es posible guardarlo como imagen .PNG desde el menú inferior de la interfaz. Este permite cambiar el nombre de la imagen del recorrido y asignar una carpeta dentro de nuestro computador.

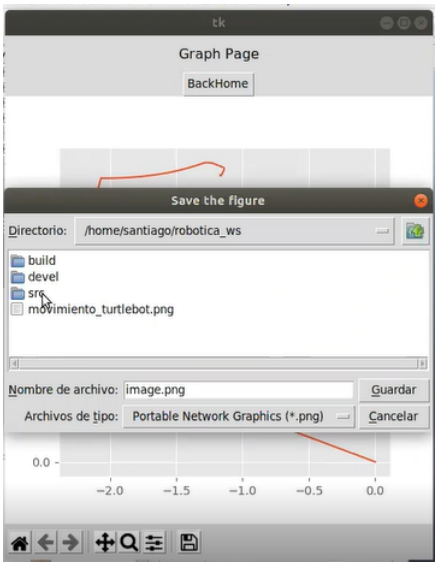

Figura 7: Guardar imagen del recorrido en el directorio deseado al finalizar la simulación.

- **Descripción de nodo Odometría:** se realizó un nodo para la interfaz (turtlebot_interface) que se suscribe al tópico turtlebot_position. Se guarda la posición x e y del robot en un vector y con la función *animation* de python se anima el movimiento. En la gráfica a continuación se evidencia la comunicación creada en ROS en donde el nodo teleop (del punto 1, movimiento) se enlaza con el tópico

cmdVel para llegar al nodo de interfaz quien a su vez comunica la posición de turtlebot para publicarla en la interfaz en tiempo real.

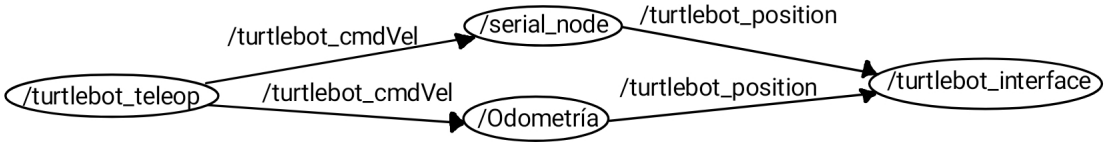

Figura 8: Grafo rqt ROS punto 2.

El robot tiene motores DC y se adjuntaron encoders para hacer el cálculo de odometría que es el estudio de la estimación de la situación de automóviles (como nuestro robot) con ruedas a lo largo de la navegación. Para efectuar esta estimación se emplea información sobre la rotación de las ruedas para apreciar cambios en la situación a lo largo del tiempo. También se emplea para referirse a la distancia que ha recorrido uno de estos automóviles (se usan sensores para su cálculo, como la odometría visual). Los robots móviles emplean la odometría para querer (y no determinar) su situación relativa a su ubicación inicial. Se obtiene buena precisión en un corto plazo y deja tasas de muestreo altas. No obstante la idea esencial de la odometría es la integración de información incremental del movimiento a lo largo del tiempo, lo que acarrea una ineludible acumulación de fallos que van incrementando proporcionalmente con la distancia recorrida por el robot.

El nodo de odometría se fundamenta en ecuaciones simples que se pueden incorporar sencillamente y que usan datos de encoders ubicados en las ruedas del robot. Se ingresaron datos como el diámetro de las llantas y la distancia entre ellas para el cálculo. Cuando recibe la información del tópico cmdVel, ejecuta los procedimientos de cálculo que se mencionaron previamente con la información que llega a través de la comunicación con Arduino el cual entrega las posiciones x, y e theta. El tópico /turtlebot_position expresa posición y orientación en un manifold 2D (float 64) a través de un mensaje tipo Pose2D para realizar el plot en la interfaz. Se realizaron dos tipos de plot, el de la Figura 9 muestra el gráfico de la posición del robot según su movimiento empezando desde la coordenada (0,0) visto desde arriba. El gráfico de la Figura 10 muestra los movimientos individuales (ejemplo; W hacia adelante) graficados desde (0,0), se puede apreciar en esta figura el movimiento hacia adelante y los giros de cada movimiento por separado.

Figura 9: Gráfico posición del robot según su movimiento empezando desde la coordenada (0,0) visto desde arriba

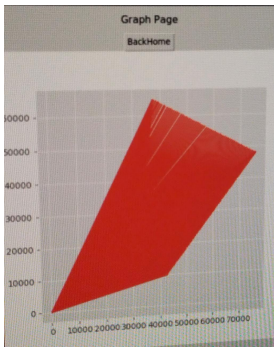

Figura 10: Movimientos individuales (ejemplo; W hacia adelante) graficados desde (0,0)

- **Punto 3:**
  Inicializa roscore y el movimiento del robot como en el punto 1. En otra terminal se escribe el comando *rosrun turtle_bot_3 turtlebot_teleop.py* para inicializar el nodo teleop. Se recrea el procedimiento del punto 1 para crear una partida con un recorrido para que el usuario elija guardarlo. Es decir, se ingresa por parámetro la velocidad lineal y angular y se escribe en la terminal los movimientos deseados para realizar el recorrido (letras W, S, A y D) finalmente, se teclea la letra *g* con el fin de guardar la partida o finalizar el recorrido. En una nueva terminal, dentro del workspace de robótica, se ejecutan los comandos *.devel/setup.bash* seguido de *rosrun turtle_bot_3 turtlebot_interface.py* para inicializar el nodo de la interfaz. Aparecerá una ventana con la interfaz y 3 botones.

Figura 11: Página 1 de la interfaz. El usuario tiene la opción de guardar el recorrido que acaba de realizar con el robot.

El usuario tiene las opciones de guardar el recorrido o usar el recorrido guardado. En esta ocasión se usará el botón de guardar el recorrido, se guardará el archivo y saldrá la opción de elegir el nombre del archivo que contiene el recorrido.

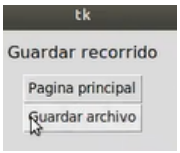

Figura 12: Página 2 de la interfaz. El usuario selecciona la opción de guardar archivo con los movimientos que componen el último recorrido realizado.

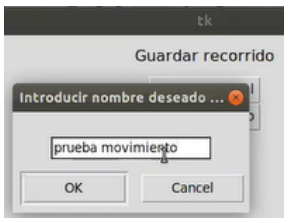

Figura 13: El usuario introduce el nombre del archivo que contiene el recorrido de su partida.

Este archivo con extensión .txt se guardará en la carpeta *Results* al mismo nivel que *Scripts* dentro del paquete de Robótica. El archivo con los movimientos del robot ejecutados al estilo del punto 1 se escribe de manera que cada movimiento especificado por una letra del teclado se presente en cada línea del archivo. Además, las primeras dos líneas del archivo contienen los valores ingresados para las velocidades lineal y angular respectivamente. A continuación se presentarán las imágenes del proceso descrito.

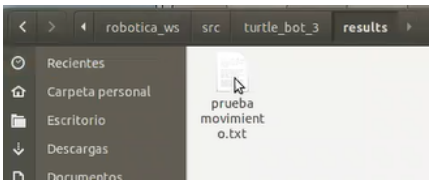

Figura 14: Archivo tipo .txt que contiene los movimientos del recorrido en la carpeta Results (en la imagen se puede apreciar la ubicación de la carpeta dentro del paquete).

Figura 15: Estructura en la que se guardan los movimientos del robot en el archivo de recorrido. Las dos primeras líneas corresponden a las velocidades lineal y angular.

- **Conclusiones:** con *pynput* de Python fue posible capturar los movimientos del robot desde el teclado. Al igual que el punto anterior, el grafo de ROS para el punto 3 comprende la comunicación entre el nodo teleop con el tópico de cmdVel para comunicar pautas de movimiento y a su vez se comunica con la interfaz y serial_node de Arduino para el tópico de posición. El archivo de texto creado tiene una dirección dentro de las carpetas del ordenados específico y por lo tanto es importante que al correr el ejercicio se haga todo desde el workspace de tal manera que permanezcan las rutas de cada carpeta empleada. Desde el nodo teleop se solicita el servicio.

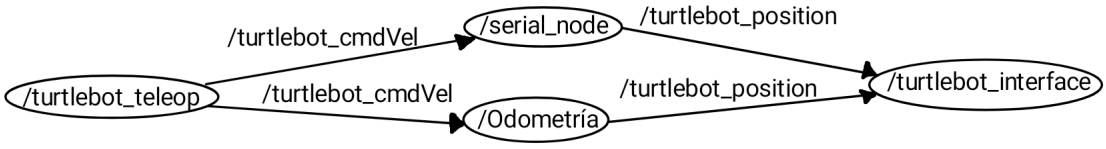

Figura 16: Grafo rqt ROS punto 3.

- **<u>Punto 4:</u>** Tras inicializar roscore y el movimiento del robot con la integración Arduino-ROS, en una nueva terminal se ejecuta dentro del workspace de robótica *. devel/setup.bash*. Seguidamente se inicializa el nodo player *rosrun turtle_bot_3 turtlebot_player.py*. En la terminal aparecerá el anuncio *Listo para recibir nombre*. En otra terminal ejecutar *. devel/ setup.bash* para llamar el servicio del nodo player: *rosservice call /cambiar_nombre ñombre archivo:¨*. En esa misma terminal, introducir el nombre del archivo que contiene la partida que se desea reproducir. Cabe recordar que todos los archivos que se quieran leer deben estar en la carpeta

Results. Una vez realizado este procedimiento se puede visualizar que la primera terminal muestra las coordenadas que están siendo publicadas a la interfaz por el nodo player.

- **Descripción nodo:** un nodo de ROS puede ser *Publisher* o *Subscriber*. Un Publisher coloca mensajes a un *Tópico* en particular. Un Subscriber se suscribe al Tópico de tal manera que reciba los mensajes cada vez que se publiquen al Tópico. Los Publishers o Subscribers no están al tanto de la existencia del otro. La idea es desacoplar la producción de información y todas las direcciones IP de varios nodos son rastreadas por el ROS Master. En el grafo a continuación se evidencia la comunicación entre el nodo player con el tópico de cmdVel para publicar y cargar en la interfaz los movimientos del robot.

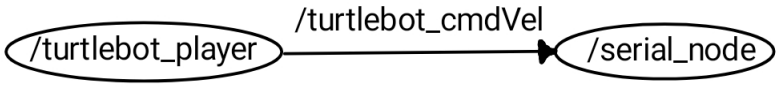

Figura 17: Grafo rqt ROS punto 4.

Referencias

[1] Motor DC. https://tienda.tdrobotica.co/chasis-ruedas/458-motorreductor-200rpm.html

[2] Rueda con neumático de goma. https://tienda.tdrobotica.co/chasis-ruedas/457-par-de-ruedas-con-neumatico-de-goma-65mm-.html

[3] Módulo Driver L298. https://tienda.tdrobotica.co/categoria/544-modulo-driver-l298.html

[4] Ball caster. https://tienda.tdrobotica.co/chasis-ruedas/263-ball-caster-34-plastico-pololu.html

[5] Protoboard. https://tienda.tdrobotica.co/protoboards-cajas-soportes/429-protoboard-pequena.html