Tecnológico de Monterrey

Reto semanal 2. Manchester Robotics

Carlos Adrián Delgado Vázquez A01735818
Alfredo Díaz López A01737250
Juan Paulo Salgado Arvizu A01737223
Bruno Manuel Zamora García A01798275

10 de abril del 2025

Resumen

En este challenge se desarrolló un sistema de control de lazo abierto en ROS2 para un robot móvil tipo Puzzlebot, con el objetivo de ejecutar trayectorias definidas. Inicialmente, se diseñó un controlador en lazo abierto para que el robot recorriera una trayectoria en forma de cuadrado de lado 2 metros. Posteriormente, el sistema fue generalizado mediante la implementación de un generador de trayectorias que permite al usuario definir diferentes rutas por medio de coordenadas, velocidades o tiempos.

Objetivos

Objetivo principal

Desarrollar un sistema de control abierto y generación de trayectorias para un robot tipo Puzzlebot, capaz de ejecutar desplazamientos definidos por el usuario en simulación y en un entorno real.

Objetivos específicos

- Diseñar e implementar un nodo en ROS2 que controle el movimiento del robot en una trayectoria cuadrada con lado de 2 metros.
- Generalizar el sistema mediante la creación de un nodo generador de trayectorias que permita al usuario definir rutas personalizadas (por puntos, velocidades o tiempos).
- Asegurar la robustez del sistema ante perturbaciones externas, ruidos de sensores y dinámicas no lineales del robot.
- Implementar un mecanismo de autoajuste para adaptar los tiempos o velocidades en función de las especificaciones ingresadas por el usuario.

Introducción

Para poder resolver el reto de esta semana es necesario conocer que son los mensajes tipo *Pose* así como los mensajes tipo twist. Ros2 utiliza los mensajes tipos pose para describir completamente la ubicación y la orientación de un objeto en el espacio tridimensional, es decir es una estructura

de datos la cual no solo indica dónde está un objeto (su posición), sino también cómo está orientado (su rotación respecto a un sistema de referencia Ilustración 1). Y es gracias a esto que este tipo de mensaje es fundamental para manejar nuestro puzzlebot (y en general en el mundo de la robótica) y forzosamente necesario para lograr que nuestro puzzlebot siga trayectorias deseadas ya que necesitamos conocer su posición y orientación en todo momento.

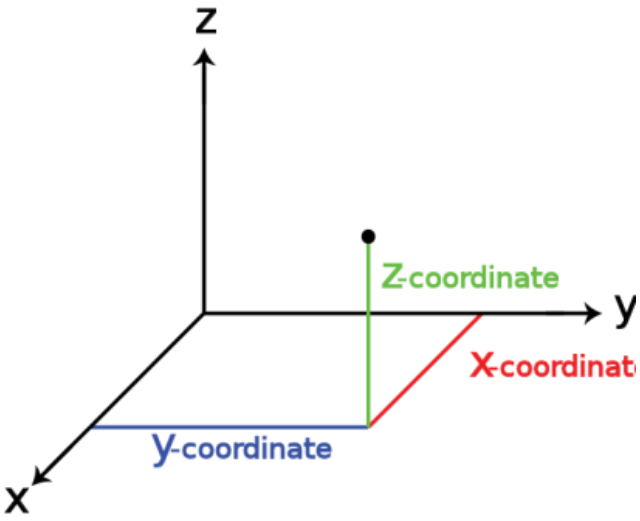

Ilustración 1 Plano tridimensional

El mensaje Pose se compone de dos partes principales: position y orientation. La position es un objeto de tipo Point, el cual almacena tres valores de tipo flotante: x, y y z. Estos representan las coordenadas espaciales de un punto en el espacio tridimensional. Por otro lado, la orientation es un objeto del tipo Quaternion, que contiene cuatro valores: x, y, z y w. Estos valores conforman un cuaternión, una forma matemática muy utilizada en robótica y gráficos 3D para representar rotaciones de manera más estable y sin ambigüedades que otros métodos como los ángulos de Euler (Robotics, s.f.).

Los mensajes tipo *Pose* están dentro del paquete geometry_msgs, Ilustración 2, por lo que podemos suscribirnos o publicar mensajes en un sistema distribuido de robots o nodos dentro del entonrno de ROS2.

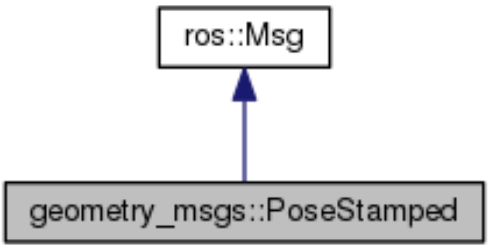

Ilustración 2 Estructura geometry_msgs

Ahora bien, para los mensajes tipo Twist es una herramienta fundamental para describir y controlar el movimiento de un robot. Con este tipo de mensajes podemos modificar la velocidad lineal y angular de nuestros motores, por ejemplo. De manera práctica, el mensaje Twist es comúnmente usado en robots móviles para enviar comandos de movimiento, como avanzar, retroceder o girar, especialmente a través de tópicos como /cmd_vel, en la Ilustración 3 Simulación de robot en plano tridimensionalIlustración 3 se puede ver al robot en el plano.

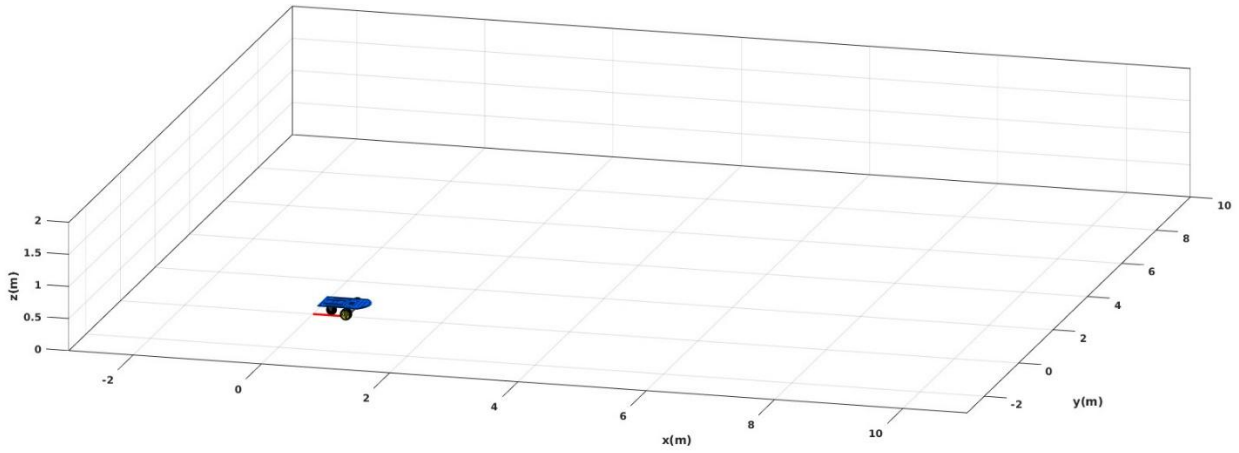

Ilustración 3 Simulación de robot en plano tridimensional

El mensaje Twist está compuesto por dos elementos principales: un vector llamado linear y otro llamado angular. Ambos vectores son de tipo Vector3, lo que significa que cada uno contiene tres componentes: x, y y z. El vector linear representa la velocidad lineal en metros por segundo (m/s) a lo largo de cada eje del espacio tridimensional, mientras que el vector angular representa la velocidad angular en radianes por segundo (rad/s), es decir, qué tan rápido está rotando el objeto alrededor de cada uno de sus ejes (Robotics, s.f.).

Al igual que con los mensajes tipo *Pose,* los mensajes twist permiten ser utilizados para publicar o suscribirse en el sistema de comunicación ROS2. Esta clase contiene dos atributos de tipo Vector3: uno llamado linear y otro llamado angular, que representan las velocidades anteriormente mencionadas. Es ampliamente utilizado en sistemas de navegación, control de movimiento, y simulaciones. Al combinarse con otros mensajes como Pose, permite tener una visión completa del estado y el comportamiento dinámico del robot en un entorno tridimensional (Robotics, s.f.). En la Ilustración 4 podemos ver un ejemplo de Pose en la trayectoria de un robot.

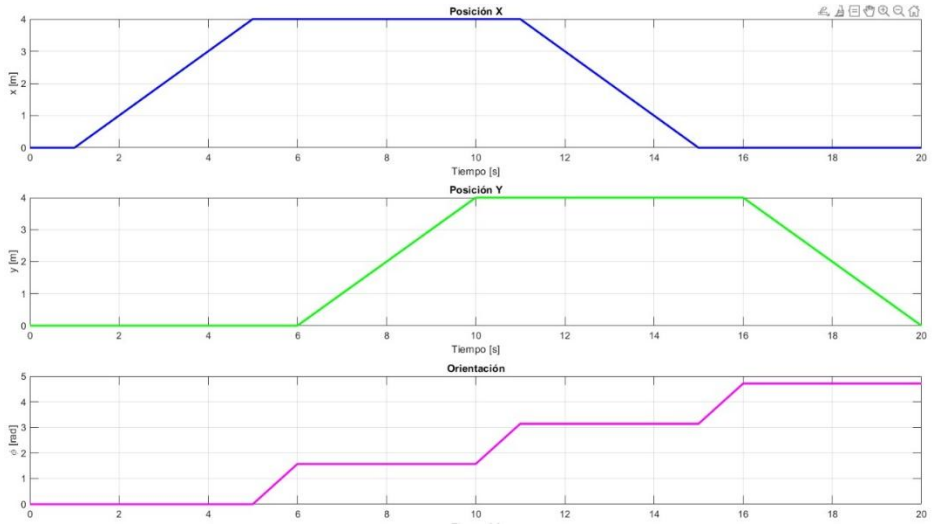

Ilustración 4 Pose en simulación de un robot trazando un cuadrado 2x2

En el ámbito de la robótica móvil, el control de un sistema puede organizarse en dos grandes categorías: lazo abierto y lazo cerrado. El control en lazo abierto, Ilustración 5, es una estrategia en la que el controlador actúa sobre el sistema sin recibir información de retroalimentación. Es decir, una vez que se genera la señal de control, esta se aplica directamente al sistema sin verificar si el resultado deseado fue alcanzado.

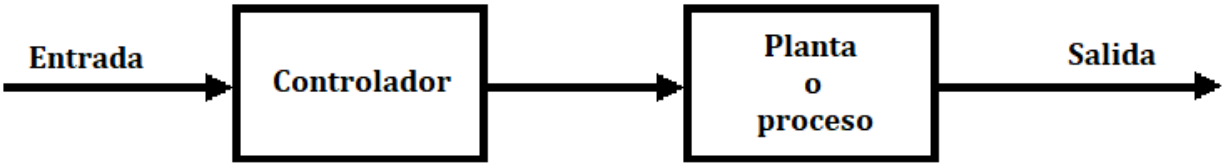

Ilustración 5 Control de lazo abierto

La principal ventaja es su simplicidad: no requiere sensores para medir la salida del sistema ni algoritmos para comparar esa salida con la referencia deseada. Esto lo convierte en una solución más fácil de programar, mantener y comprender, lo cual resulta muy útil en entornos educativos o en prototipos iniciales. También puede ofrecer respuestas rápidas, ya que el controlador no necesita esperar a recibir información del sistema para actuar; simplemente ejecuta las órdenes. Sin embargo, la ventaja más evidente es su incapacidad para corregir errores. Esta limitación lo hace muy sensible a perturbaciones externas, como variaciones en el terreno, desgaste de los motores, condiciones ambientales, o cualquier otro factor que altere la respuesta del sistema. Otra desventaja

significativa es que el control en lazo abierto no garantiza precisión ni exactitud en aplicaciones dinámicas o de larga duración, ya que los errores tienden a acumularse con el tiempo.

Para mejorar la robustez de un sistema de control en lazo abierto, es necesario aplicar algunas estrategias que compensen la falta de retroalimentación. Una de las más importantes es contar con un modelo dinámico preciso del robot. Esto implica conocer y ajustar parámetros como el peso del robot, el radio de las ruedas, la constante de fricción, y las relaciones entre la señal del motor y la velocidad real. Un modelo bien ajustado permite que el controlador prediga con mayor exactitud cómo se comportará el sistema ante ciertos comandos. Otra estrategia consiste en realizar una caracterización empírica del comportamiento del robot: medir cuánto realmente avanza ante diferentes señales de velocidad y corregir los comandos de forma manual para compensar errores observados.

También puede utilizarse el concepto de control anticipado o feedforward, que consiste en prever cómo se comportará el sistema ante una entrada determinada y ajustar la señal de control en función de ese modelo. Aunque el sistema sigue sin recibir retroalimentación en tiempo real, esta técnica mejora la precisión al considerar las propiedades físicas del robot, como el Puzzlebot Ilustración 6, antes de ejecutar el movimiento. Finalmente, una buena práctica es limitar el uso del lazo abierto a entornos controlados donde se minimicen perturbaciones externas, como superficies planas, sin obstáculos ni variaciones ambientales que afecten el comportamiento del sistema.

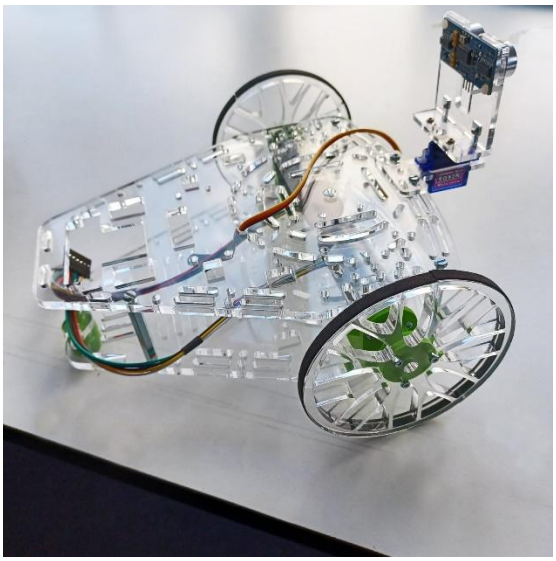

Ilustración 6 Puzzlebot

El cálculo de la distancia y el ángulo recorridos por un robot Puzzlebot se realiza utilizando el modelo cinemático diferencial, que es adecuado para robots móviles con dos ruedas motrices independientes. En este tipo de plataforma, cada rueda puede avanzar o retroceder a distintas velocidades, lo que permite al robot tanto desplazarse en línea recta como girar sobre su propio eje. El sistema de odometría del Puzzlebot se basa en la lectura de encoders instalados en cada rueda, los cuales permiten estimar la distancia recorrida midiendo el número de pulsos generados en cada giro parcial.

La distancia que recorre cada rueda se calcula a partir del número de pulsos registrados y de las características físicas del robot, como el radio de la rueda y la resolución del encoder. Una vez obtenidas las distancias individuales de cada rueda, se puede estimar la distancia lineal promedio recorrida por el robot, así como el cambio de orientación con respecto a su trayectoria inicial. Este cambio angular depende de la diferencia de distancias entre ambas ruedas y de la distancia entre ruedas, también conocida como base del robot.

A partir de estos datos, se puede actualizar la pose del robot en el plano (sus coordenadas x, y y su ángulo θ) utilizando expresiones trigonométricas que consideran la orientación previa. Esta forma de estimación, llamada odometría diferencial, es efectiva para recorridos cortos y en superficies planas. Sin embargo, es importante tener en cuenta que pequeñas imprecisiones se acumulan con el tiempo, por lo que en trayectorias largas o entornos no controlados no es recomendable no usar un lazo cerrado.

La velocidad lineal del robot es el promedio de las velocidades lineales de ambas ruedas, y que la velocidad angular se determina por la diferencia entre esas velocidades, dividida entre la distancia entre ruedas (L). Como la velocidad lineal de cada rueda se obtiene multiplicando su velocidad angular por el radio de la rueda (r), se pueden derivar las siguientes ecuaciones:

$$u = \frac{r}{2}(\omega_r + \omega_L)$$

$$\omega = \frac{r}{L}(\omega_r - \omega_L)$$

En estas ecuaciones:

- u representa la velocidad lineal del robot, es decir, qué tan rápido se desplaza su centro hacia adelante.

- w representa la velocidad angular, es decir, qué tan rápido está girando sobre su propio eje.
- ω_L y ω_R son las velocidades angulares de la rueda izquierda y derecha, respectivamente.
- r es el radio de las ruedas.
- L es la distancia entre ambas ruedas.

Estas fórmulas son muy utilizadas en robótica móvil para implementar odometría diferencial, que es la técnica por la cual el robot estima su trayectoria acumulando pequeños desplazamientos y giros a lo largo del tiempo. A partir de u y w, el robot puede integrar estos valores en el tiempo y actualizar su posición estimada (x, y) y orientación θ en el plano (Siegwart et al., 2011).

Solución del problema

Para la solución del challenge, ocupamos dos nodos, uno que realiza la trayectoria del cuadrado y que puedes modificar la velocidad lineal o el tiempo máximo para realizar la trayectoria y otro nodo el cual le damos el punto al que queremos que avance el Puzzlebot y la velocidad o el tiempo deseado. Con base en esta elección, el sistema realiza los cálculos necesarios para obtener el parámetro faltante (velocidad o tiempo), garantizando que el perfil de movimiento sea físicamente factible y suave.

Para el primer nodo, realizamos un lazo abierto de una trayectoria fija (cuadrado de 2 m por lado) Se implementó el nodo square_controller.py, que controla el movimiento del robot a través de una máquina de estados (FSM), está compuesta por 7 estados principales, cada uno con una función específica dentro del ciclo de movimiento del robot:

Estado 0: Stop inicial
El robot comienza completamente detenido para asegurar que no haya movimiento residual antes de iniciar la trayectoria.

Estado 1: Giro angular
El robot realiza una rotación sobre su eje, ajustando su orientación en base al ángulo predefinido para el tramo actual. El giro se realiza a una velocidad angular constante (0.5 rad/s), y la duración

del giro se calcula dividiendo el ángulo entre dicha velocidad. La rotación se dirige según el signo del ángulo (positivo o negativo).

Estado 2: Pausa post-giro

Después del giro, se realiza una pausa de 1 segundo para estabilizar al robot y evitar que las inercias afecten el inicio del movimiento lineal.

Estado 3: Rampa de aceleración

Se inicia un incremento progresivo de la velocidad lineal, comenzando desde una velocidad base (0.03 m/s) hasta alcanzar la velocidad deseada V. Esta transición se hace linealmente durante un tiempo definido (por ejemplo, 0.3s), lo que evita aceleraciones bruscas que puedan generar errores acumulativos en la posición.

Estado 4: Movimiento recto a velocidad constante

Durante este estado, el robot avanza a velocidad constante V durante el tiempo necesario para recorrer la distancia definida para el tramo. Este tiempo se calcula como:

$$Tcrucero = Ttotal - Taceleración - Tdesaceleración$$

Estado 5: Rampa de desaceleración

Similar al estado 3, pero en sentido contrario: la velocidad se reduce suavemente desde V hasta 0.03 m/s. Esta desaceleración controlada evita que el robot se detenga bruscamente, lo que podría afectar su orientación o posición.

Estado 6: Pausa final

Se agrega una breve pausa (1 segundo) al final de cada tramo antes de pasar al siguiente. Esta pausa asegura que el robot haya terminado completamente su movimiento antes de iniciar un nuevo giro.

Estado 7: Stop final

Una vez completados todos los tramos del cuadrado, el robot se detiene definitivamente y se cancela el temporizador que regula la FSM.

En este nodo, el usuario puede ingresar el modo de control (“tiempo” o “velocidad”) por tramo.

A partir de esta información, se calculan los parámetros necesarios (velocidad o tiempo) para ejecutar la trayectoria con transiciones suaves (aceleración y desaceleración).

El movimiento para cualquier nodo está compuesto por tres fases:

1. Una rampa de aceleración de 1 segundo, donde el robot incrementa su velocidad desde una velocidad mínima inicial (0.03 m/s) hasta alcanzar la velocidad de crucero.
2. Una fase de crucero a velocidad constante.
3. Una rampa de desaceleración de 1 segundo, donde el robot disminuye su velocidad desde la velocidad de crucero hasta la mínima inicial.

El sistema calcula qué velocidad constante debería mantener el robot para recorrer la distancia total en ese tiempo, considerando además los 2 segundos ocupados por las rampas de aceleración y desaceleración. Se asume que el movimiento durante las rampas es lineal, por lo que la distancia cubierta en ellas puede calcularse como el promedio entre la velocidad inicial y la final, multiplicado por el tiempo (1 segundo).

Sin embargo, si el usuario especifica la velocidad deseada, se calcula cuánto tiempo tomaría recorrer el tramo completo considerando nuevamente las rampas y la fase de crucero. Se estima la distancia que se recorrerá durante las rampas (usando el promedio de velocidades) y se resta del total. La distancia restante se recorrerá a velocidad constante, y a partir de eso se determina la duración total necesaria para completar el tramo.

También existe la automatización del controlador con lectura de archivo YAML Se desarrolló el nodo controller.py, el cual carga automáticamente el plan.yaml generado y ejecuta la trayectoria completa sin necesidad de intervención manual. Este nodo utiliza también una FSM para garantizar movimientos suaves y controlados.

También consideramos la dead zone y la zona de saturación de nuestros motores para proteger a nuestro controlador de posibles mal funciones y darle a conocer al usuario que los parámetros que eligió no son posibles.

De acuerdo con varias pruebas validamos que si la velocidad resulta menor a 0.03 m/s estamos en la zona muerta por lo que no podremos salir del estado de la inercia. Y si la velocidad es mayor a 0.9 m/s (los límites dinámicos del robot), se notifica que el valor está fuera de rango.

Resultados

Iniciamos los parámetros para el cuadrado como se ve en la Ilustración 7

```
brunene@brunene-rog:~/ros2_ws$ ros2 run puzzlebot_control path_generator
[INFO] [1744311783.782913754] [path_generator]: Configurando tramo 1: ángulo = 30.00 °, distancia = 1.50 m
Para el tramo 1, ingrese el modo ('tiempo' o 'velocidad'): velocidad
Ingrese la velocidad deseada (en m/s) para el tramo 1 (entre 0.03 y 0.9): 0.4
[INFO] [1744311788.180271711] [path_generator]: Tramo 1: Velocidad = 0.40 m/s, tiempo calculado = 4.67 s
[INFO] [1744311788.180559202] [path_generator]: Configurando tramo 2: ángulo = 120.00 °, distancia = 1.50 m
Para el tramo 2, ingrese el modo ('tiempo' o 'velocidad'): velocidad
Ingrese la velocidad deseada (en m/s) para el tramo 2 (entre 0.03 y 0.9): 0.4
[INFO] [1744311792.501013549] [path_generator]: Tramo 2: Velocidad = 0.40 m/s, tiempo calculado = 4.67 s
[INFO] [1744311792.501377232] [path_generator]: Configurando tramo 3: ángulo = -138.46 °, distancia = 1.50 m
Para el tramo 3, ingrese el modo ('tiempo' o 'velocidad'): velocidad
Ingrese la velocidad deseada (en m/s) para el tramo 3 (entre 0.03 y 0.9): 0.4
[INFO] [1744311796.272051650] [path_generator]: Tramo 3: Velocidad = 0.40 m/s, tiempo calculado = 4.67 s
[INFO] [1744311796.272397009] [path_generator]: Configurando tramo 4: ángulo = -90.00 °, distancia = 1.50 m
Para el tramo 4, ingrese el modo ('tiempo' o 'velocidad'): velocidad 0.4
Modo inválido. Ingrese 'tiempo' o 'velocidad': velocidad
Ingrese la velocidad deseada (en m/s) para el tramo 4 (entre 0.03 y 0.9): 0.4
[INFO] [1744311811.260112381] [path_generator]: Tramo 4: Velocidad = 0.40 m/s, tiempo calculado = 4.67 s
[INFO] [1744311811.262851779] [path_generator]: Plan guardado en 'plan.yaml'.
[INFO] [1744311811.263338434] [path_generator]: Nodo path_generator inicializado y listo.
[INFO] [1744311811.264516638] [path_generator]: Waypoint 1 publicado.
[INFO] [1744311811.270470317] [path_generator]: Waypoint 2 publicado.
[INFO] [1744311811.371236063] [path_generator]: Waypoint 3 publicado.
[INFO] [1744311811.470800321] [path_generator]: Waypoint 4 publicado.
[INFO] [1744311811.570871211] [path_generator]: Waypoint 5 publicado.
[INFO] [1744311811.670625236] [path_generator]: Waypoint 6 publicado.
[INFO] [1744311811.770691006] [path_generator]: Waypoint 7 publicado.
[INFO] [1744311811.870338711] [path_generator]: Waypoint 8 publicado.
[INFO] [1744311811.971237137] [path_generator]: Waypoint 9 publicado.
[INFO] [1744311812.070408246] [path_generator]: Todos los waypoints han sido publicados.
```

Ilustración 7 Parametros cuadrado

Enlace de los videos de evidencia de movimientos del robot:

https://drive.google.com/drive/folders/1qgEeMvpu47YMeIxZNAcoF8ZnR_ANyJBL?usp=sharing

Cuando el usuario decide correr el nodo del cuadrado se despliega lo siguiente Ilustración 8 Nodos del cuadrado:

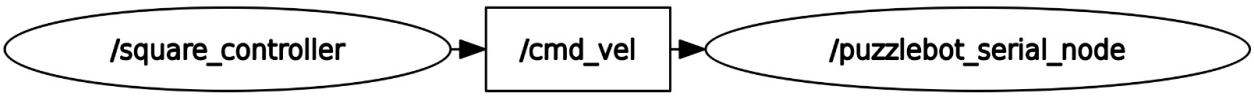

Ilustración 8 Nodos del cuadrado

Y esta es la grafica de las velocidades lineales y angulares cuando esta corriendo el nodo del cuadrado Ilustración 9 Plot de velocidades en el cuadrado

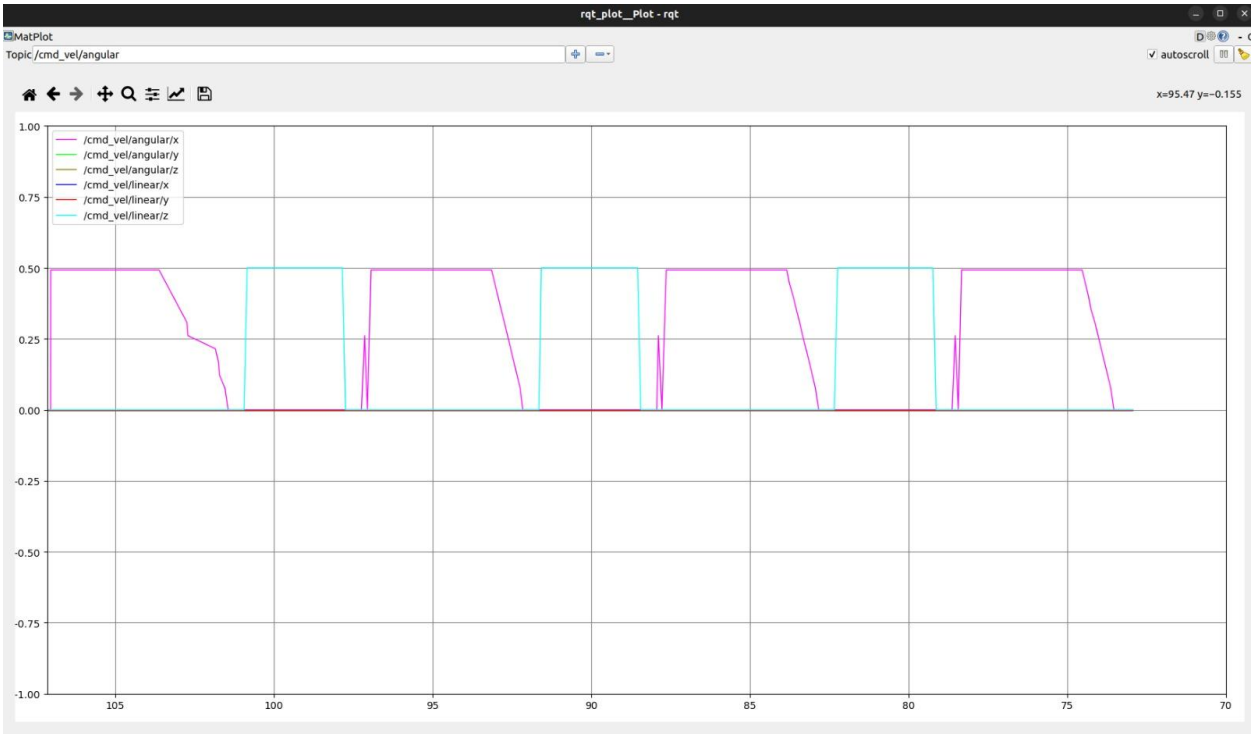

Ilustración 9 Plot de velocidades en el cuadrado

Posteriormente hacemos como en el cuadrado como el zigzag como en Ilustración 10

```
(base) brunene@brunene-rog:~/ros2_ws$ ros2 run puzzlebot_control square_controller
[INFO] [1744310967.401707985] [square_controller]: Configurando tramo 1: ángulo = 0.00°, distancia = 2.00 m
Tramo 1: Ingrese el modo ('tiempo' o 'velocidad'): tiempo
Tramo 1: Ingrese el tiempo total (en segundos, >4) para recorrer el tramo: 5
[INFO] [1744310973.853638253] [square_controller]: Tramo 1 configurado: Modo tiempo, T = 5.00 s, V = 0.49 m/s, tiempo crucero = 3.00 s.
[INFO] [1744310973.853945029] [square_controller]: Configurando tramo 2: ángulo = 90.00°, distancia = 2.00 m
Tramo 2: Ingrese el modo ('tiempo' o 'velocidad'): tiempo
Tramo 2: Ingrese el tiempo total (en segundos, >4) para recorrer el tramo: 5
[INFO] [1744310977.998249416] [square_controller]: Tramo 2 configurado: Modo tiempo, T = 5.00 s, V = 0.49 m/s, tiempo crucero = 3.00 s.
[INFO] [1744310977.998652935] [square_controller]: Configurando tramo 3: ángulo = 90.00°, distancia = 2.00 m
Tramo 3: Ingrese el modo ('tiempo' o 'velocidad'): tiempo
Tramo 3: Ingrese el tiempo total (en segundos, >4) para recorrer el tramo: 5
[INFO] [1744310985.297533204] [square_controller]: Tramo 3 configurado: Modo tiempo, T = 5.00 s, V = 0.49 m/s, tiempo crucero = 3.00 s.
[INFO] [1744310985.297843537] [square_controller]: Configurando tramo 4: ángulo = 90.00°, distancia = 2.00 m
Tramo 4: Ingrese el modo ('tiempo' o 'velocidad'): tiempo
Tramo 4: Ingrese el tiempo total (en segundos, >4) para recorrer el tramo: 5
[INFO] [1744310988.401151319] [square_controller]: Tramo 4 configurado: Modo tiempo, T = 5.00 s, V = 0.49 m/s, tiempo crucero = 3.00 s.
[INFO] [1744310988.401746668] [square_controller]: Nodo square_controller inicializado y listo.
[INFO] [1744310988.402441794] [square_controller]: Estado 0: Stop inicial
[INFO] [1744310988.492294360] [square_controller]: Estado 1: Giro, ángulo = 0.00°
[INFO] [1744310988.592480698] [square_controller]: Estado 2: Pausa post giro
[INFO] [1744310989.092441998] [square_controller]: Estado 3: Rampa de aceleración
[INFO] [1744310990.192111183] [square_controller]: Estado 4: Movimiento lineal a velocidad constante
[INFO] [1744310993.992004043] [square_controller]: Estado 5: Rampa de desaceleración
[INFO] [1744310994.192107561] [square_controller]: Estado 6: Pausa final del tramo
[INFO] [1744310994.691882933] [square_controller]: Estado 1: Giro, ángulo = 90.00°
[INFO] [1744310997.892065877] [square_controller]: Estado 2: Pausa post giro
[INFO] [1744310998.391799290] [square_controller]: Estado 3: Rampa de aceleración
[INFO] [1744310999.492133613] [square_controller]: Estado 4: Movimiento lineal a velocidad constante
[INFO] [1744311003.292034138] [square_controller]: Estado 5: Rampa de desaceleración
[INFO] [1744311003.491236743] [square_controller]: Estado 6: Pausa final del tramo
[INFO] [1744311003.992008718] [square_controller]: Estado 1: Giro, ángulo = 90.00°
[INFO] [1744311007.191670090] [square_controller]: Estado 2: Pausa post giro
[INFO] [1744311007.691738122] [square_controller]: Estado 3: Rampa de aceleración
[INFO] [1744311008.792569199] [square_controller]: Estado 4: Movimiento lineal a velocidad constante
[INFO] [1744311012.591666502] [square_controller]: Estado 5: Rampa de desaceleración
[INFO] [1744311012.791342898] [square_controller]: Estado 6: Pausa final del tramo
[INFO] [1744311013.292130623] [square_controller]: Estado 1: Giro, ángulo = 90.00°
[INFO] [1744311016.492006699] [square_controller]: Estado 2: Pausa post giro
[INFO] [1744311016.991636618] [square_controller]: Estado 3: Rampa de aceleración
[INFO] [1744311018.091493995] [square_controller]: Estado 4: Movimiento lineal a velocidad constante
[INFO] [1744311021.891509837] [square_controller]: Estado 5: Rampa de desaceleración
[INFO] [1744311022.091557079] [square_controller]: Estado 6: Pausa final del tramo
[INFO] [1744311022.591223116] [square_controller]: Estado 7: Stop final
```

Ilustración 10 Parametros en zigzag

Cuando el usuario decide correr el nodo del path generator se despliega lo siguiente Ilustración 11:

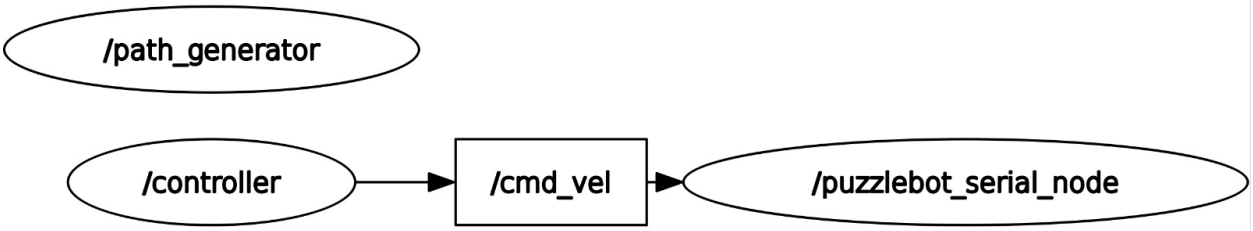

Ilustración 11 Grafo de nodos en zigzag

Esta es la gráfica de las velocidades lineales y angulares cuando está corriendo el nodo del del path Ilustración 12.

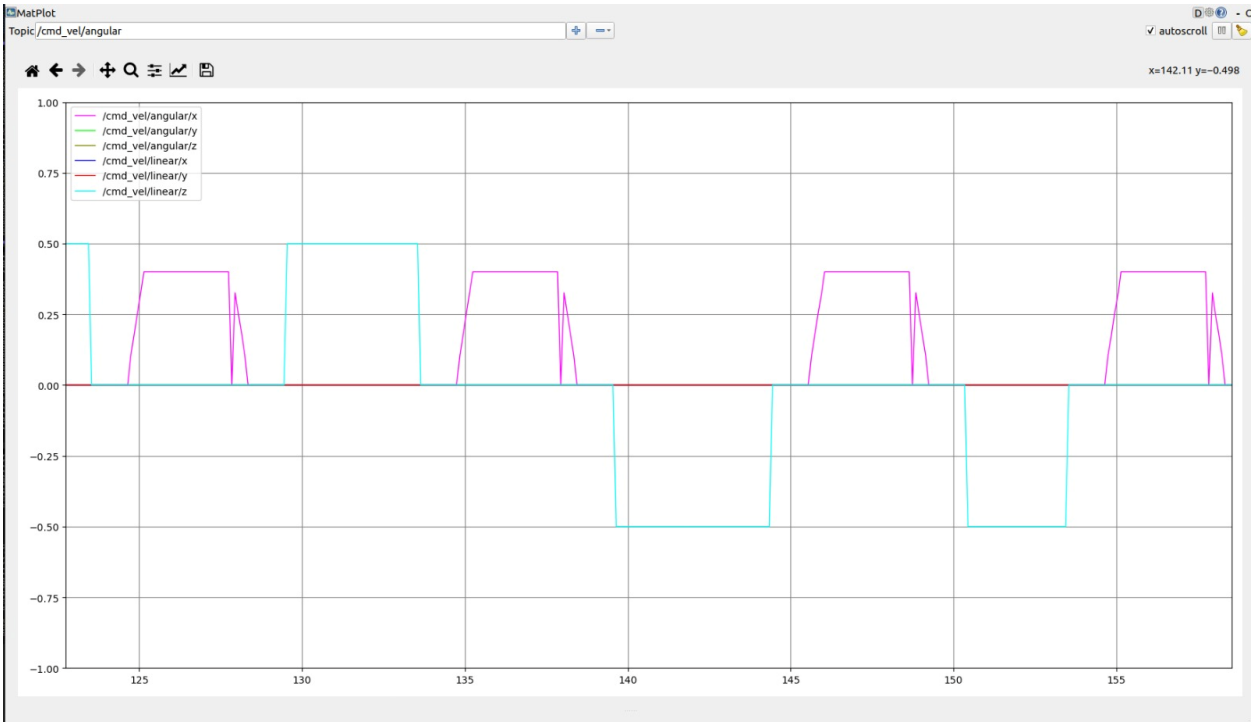

Ilustración 12 Plot de velocidades de zigzag

Conclusiones

El proyecto cumplió en gran medida sus objetivos, logrando que el Puzzlebot ejecutara trayectorias como cuadrados y zigzags, tanto en simulación como en entorno real. Se implementó un sistema

de control en lazo abierto y un generador de trayectorias que permitió definir rutas por puntos, velocidades o tiempos.

Aunque no se logró una robustez completa frente a perturbaciones y errores acumulados, esto se debe a la ausencia de retroalimentación en tiempo real. Como mejora, se propone implementar control en lazo cerrado para mayor precisión y adaptación, así, los errores en la trayectoria pueden ser corregidos, evitando un error acumulativo.

En general, el proyecto permitió comprender los procesos clave del control y planificación de trayectorias, sentando una base sólida para futuras mejoras.

Bibliografía

Open Robotics. (n.d.). *geometry_msgs message documentation (ROS Noetic)*. ROS Wiki. https://docs.ros.org/en/noetic/api/geometry_msgs/html/index-msg.html

Open Robotics. (n.d.). *geometry_msgs/Twist message*. ROS Documentation (ROS Noetic). https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to Autonomous Mobile Robots* (2nd ed.). MIT Press.