Reto semanal 2. Manchester Robotics

Oscar Ortiz Torres A01769292

Yonathan Romero Amador A01737244

Ana Itzel Hernández García A01737526

Fundamentación de robótica

Grupo 101

Jueves 10 de abril de 2025

Resumen

En el siguiente reporte, se describe el desarrollo e implementación de un sistema de control de lazo abierto para un robot móvil utilizando ROS2, como parte del segundo reto semanal. El proyecto se dividió en dos partes: la primera consistió en la creación de un nodo capaz de controlar al robot para seguir una trayectoria cuadrada definida por parámetros como velocidad o tiempo; la segunda parte abordó la ejecución de trayectorias personalizadas definidas por el usuario, mediante mensajes con información adicional como velocidad y duración.

Se desarrollaron dos nodos principales, uno encargado de ejecutar los movimientos del robot con base en parámetros o mensajes recibidos, y otro responsable de generar y enviar la secuencia de puntos que forman la trayectoria deseada. Ambos nodos fueron diseñados considerando la robustez operativa, validación de parámetros, y la modularidad del sistema para garantizar un comportamiento estable y seguro ante perturbaciones.El resultado es un sistema capaz de ejecutar movimientos precisos, adaptarse a configuraciones dinámicas y operar eficientemente en un entorno simulado o real.

Objetivos

El objetivo general de este challenge es el desarrollo de nodos en ROS2 para controlar un robot móvil en entornos reales, permitiendo el seguimiento de trayectorias definidas por el usuario. Aplicando un ajuste automático y robustez ante perturbaciones.

Objetivos del reto del cuadrado:

- Diseñar un nodo de control de lazo abierto capaz de girar el puzzlebot siguiendo el trayecto un cuadrado de 2 metros por lado.

- Garantizar la robustez del controlador ante perturbaciones y ruido considerando los criterios definidos por el usuario.
- Implementación de un mecanismo de autoajuste para estimar los parámetros como velocidad o tiempo a partir de las preferencias del usuario.

Objetivos del reto de la forma libre:

- Implementar un nodo generador de trayectorias que permite al usuario definir caminos personalizados mediante parámetros como puntos, tiempo o velocidad.
- Validar la factibilidad de los puntos considerando el comportamiento dinámico del robot y los parámetros ingresados por el usuario.
- Integrar un mensaje personalizado en el tópico `/pose` que incluye la información adicional sobre las velocidades o el tiempo.

Introducción

En este proyecto se desarrolla un sistema de control de lazo abierto en un robot móvil utilizando ROS2. Para comenzar, se tienen que tener en cuenta los siguientes conceptos, mensajes tipo Pose, el control de lazo abierto, cálculo de la distancia y ángulo recorrido por un robot móvil. Los cuales son la base para operar y monitorear el comportamiento de un robot dentro de un entorno específico.

Los mensajes Pose forman parte del paquete `geometry_msgs` y son utilizados para representar una posición y orientación en el espacio libre. Empleado en aplicaciones robóticas para describir la ubicación de un robot u objeto en un sistema de coordenadas. Está compuesto por dos elementos principales: `Position`, representado por un objeto `point` que contiene las coordenadas x, y, z en el espacio tridimensional y Orientation, representado por un objeto

Quaternion que describe la orientación en términos de rotaciones tridimensionales usando componentes como x, y, z, w. (Geometry_Msgs/Pose Documentation, s. f.)

Por otro lado, el control de lazo abierto, es un sistema de control en el que la acción del control no depende de la salida del sistema, lo que significa que no requiere retroalimentación para ajustar su operación, actuando solamente en función de la entrada proporcionada. (Staff, 2014)

Se caracteriza por la independencia de la salida, es decir, no se considera el estado actual del proceso a la variable controlada. Son sistemas más simples y económicos de implementar debido a que no requieren sensores ni mecanismos de retroalimentación. Además que son más estables que los sistemas de lazo cerrado porque no hay fluctuaciones causadas por ajustes automáticos. (Difference between Open-Loop & Closed-Loop Control System, 2022)

Sus limitaciones se conocen como no poder corregir errores ni adaptarse a cambios en condiciones extremas y no permite la optimización dinámica, lo cual causa que sean menos confiables que un sistema de control de lazo cerrado. El control de lazo abierto es ideal en procesos predecibles y bien definidos, donde las condiciones extremas tienen poca influencia en el resultado esperado, por ejemplo, en semáforos o lavadoras automáticas. (Difference between Open-Loop & Closed-Loop Control System, 2022)

Finalmente, el cálculo de la distancia y el ángulo recorrido permite estimar la trayectoria del robot, útil cuando no se cuenta con sensores de posición avanzados, la distancia de un robot se puede medir mediante los encoders, los cuales generan pulsos al momento en el que giran los motores permitiendo que se realice el cálculo mediante la cantidad de pulsos y la resolución del encoder.(V. et al., s/f)

El cálculo del ángulo rotatorio se puede obtener mediante sensores anulares o por medio del cambio en la orientación del robot a partir de la velocidad angular de las ruedas. Conociendo la distancia y el ángulo se puede obtener la posición actual y su trayectoria. (V. et al., s/f)

Solución del problema

Parte 1 | Controlador de una trayectoria específica

Comenzando con este reto, para asegurar un comportamiento controlado y robusto del robot, se logró la identificación de la región lineal de los motores, es decir, los límites mínimos y máximos en los que el robot responde adecuadamente. Dichos valores se encuentran en un rango entre 0.025 m/s a 0.38 m/s para la velocidad lineal y en un rango de 5.30s a 80.0s para el tiempo. Con respecto a la velocidad angular esta se determinó mantenerse constante en 0.1m/s

En la primera parte del reto se desarrolló el nodo `open_loop_ctrl` en ROS2 encargado de controlar robot para realizar el recorrido de la trayectoria de un cuadrado de 2m por cada lado, mediante comandos de velocidad publicados al tópico `/cmd_vel.`

Los parámetros están definidos como `velocity_goal` para la velocidad deseada al avanzar en línea recta y `time_goal` que es el tiempo recorrido para recorrer cada lado del cuadrado.El nodo se adapta al recibir cualquiera de estos parámetros, si se modifica uno, el nodo calcula automáticamente el otro.

En la máquina de estados, el ciclo de control se ejecuta en una frecuencia de 10 Hz, verificando constantemente el tiempo transcurrido y actualizando las velocidades dependiendo del estado actual, cuenta con tres estados los cuales son: avance recto, donde se envía la velocidad en `X` mientras el tiempo transcurrido no supere el valor de la variable `forward_time`, giro en su mismo eje, donde se envía velocidad ángulo en `Z` para realizar la rotación de 90°

durante el valor de la `rotate_time` y el reposo se ejecuta cuando se completan los 4 lados, deteniendo al robot y reiniciando su estado interno.

Para la publicación de comandos, el nodo se comunica por medio de la publicación periódica de mensajes de tipo `geometry_msgs/msg/Twist` al tópico `/cmd_vel`, a diferencia del mensaje tipo `Pose`, está formado por las velocidades lineales (`linear.x, linear.y, linear.z`) y por las velocidades angulares (`angular.x, angular.y, angular.z`). En este caso, solo ocupamos dos componentes, `linear.x` para mover el robot hacia adelante y `angular.z` para girar el robot sobre su eje a la izquierda.

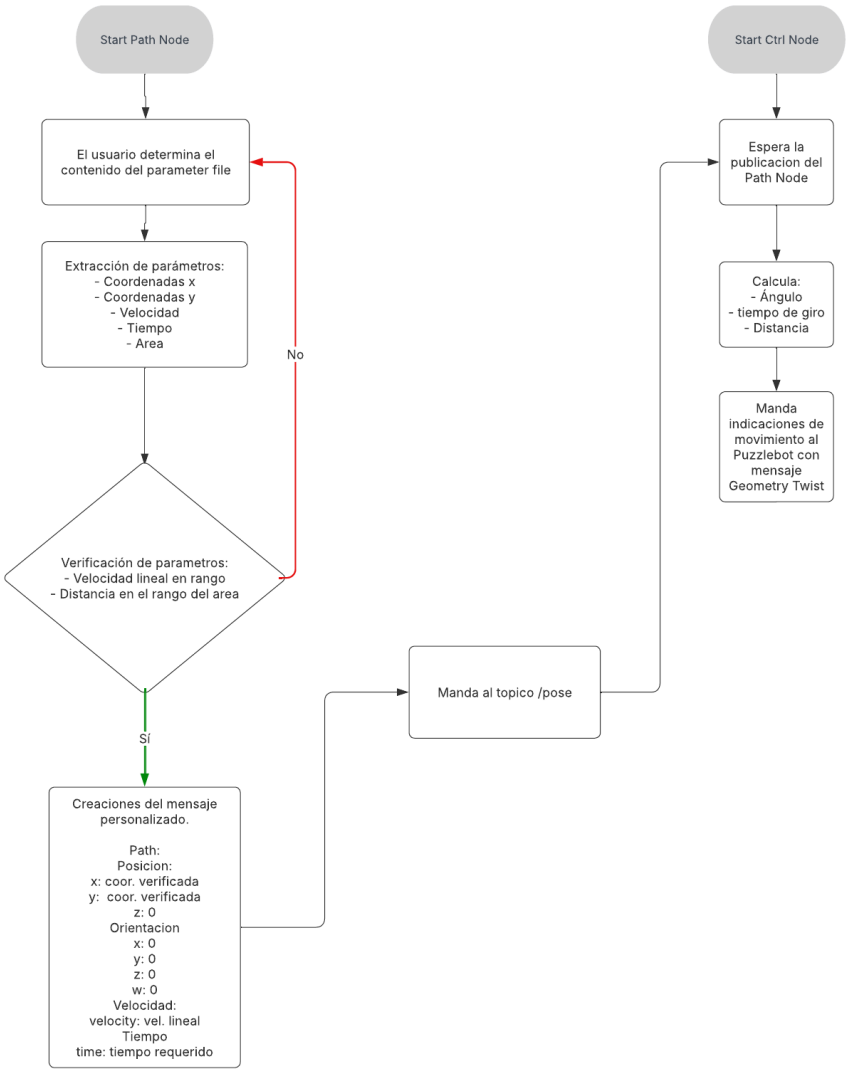
Se permite el cambio de parámetros mediante la interfaz de comunicación `add_on_set_parameters_callback`, pero en el momento en el que el robot se encuentre en movimiento, es decir cuando sea `robot_busy == True`, se bloquean las modificaciones de parámetros para evitar irregularidades.

El nodo `OpenLoopCtrl` presenta robustez en su capacidad para gestionar tanto el control de movimiento como la configuración dinámica de parámetros durante su operación. El diseño modular basado en estados garantiza una transición ordenada entre las fases de movimiento lineal, rotación y reposo, lo que facilita una ejecución controlada y predecible de las trayectorias del robot. Además, la implementación de un callback para la actualización de parámetros permite la modificación dinámica de los valores de tiempo y velocidad del robot sin necesidad de reiniciar el sistema, pero con la protección de que los cambios sólo son aceptados si el robot no está en movimiento, evitando posibles inconsistencias o errores de control. La verificación de los parámetros dentro de un rango válido asegura que los valores ingresados sean correctos antes de ser aplicados, incrementando la fiabilidad del sistema. Por otro lado, el uso de un temporizador de control para el ciclo de ejecución asegura que el robot mantenga un comportamiento

consistente con tiempos de muestreo definidos, permitiendo ajustes finos en el desempeño del robot. Estas características hacen del nodo una solución robusta, capaz de manejar trayectorias predeterminadas y adaptarse a parámetros dinámicos sin comprometer la seguridad o estabilidad operativa.

Parte 2 | Sistema de control para trayectorias personalizadas

Para el reto de la figura libre, se implementa la lógica del diagrama de flujo mostrado en la Imagen 1.



Imgen 1. Diagrama de flujo del sistema

Empezamos iniciando el nodo de control `open_loop_ctrl` y se establece una subscripción al tópico `/pose,` esperando mensajes del nodo `Path_Rosario`. Cuando se envía el mensaje personalizado basado en mensaje pose (la posición y la orientación) y se le agrega la velocidad lineal o el tiempo ejecución, realizando los cálculos para el ángulo al siguiente punto, usando el `np.linialg.norm` , se calcula la distancia a recorrer y el tiempo de recorrido. Con esto obtenido se puede general el mensaje tipo `Twist` que se publica al tópico `/cmd_vel` en el cual se mueve el robot.

Implementación del nodo PathNode

El nodo `PathNode` es responsable de generar y publicar una secuencia de mensajes personalizados que representan una trayectoria definida por el usuario. Su implementación en Python, utilizando ROS 2.

Al iniciar, el nodo declara y obtiene los siguientes parámetros configurables:

`coordenadas_x` y `coordenadas_y`: listas de valores en los ejes `X` y `Y` que definen los puntos por los cuales debe pasar el robot.

- velocidad: velocidades lineales asociadas a cada segmento de la trayectoria.
- tiempo: tiempos estimados de desplazamiento entre puntos.
- área: valor escalar que representa el límite del área de operación permitida para el robot (asumiendo un área cuadrada centrada en el origen).

El nodo primero verifica que todos los puntos se encuentren dentro del área de trabajo válida y que cada punto tenga un par válido de coordenadas. Luego, en función de los valores proporcionados, el nodo realiza uno de los siguientes cálculos:

- Si se proporciona un vector de velocidades, se calcula el tiempo requerido para recorrer cada segmento de la trayectoria.
- Si se proporciona un vector de tiempos, se calcula la velocidad necesaria para cada tramo.
- Si no se proporciona ninguno o ambos vectores están vacíos o incompletos, el nodo muestra un mensaje de error y no continúa con la generación de la trayectoria.

En ambos casos, se verifica que las velocidades estén dentro de un rango permitido (entre 0.025 m/s y 0.38 m/s), asegurando que el robot se mantenga dentro de parámetros seguros de operación.

Posteriormente, el nodo crea un mensaje personalizado del tipo `RosarioPath` para cada punto verificado. Este mensaje incluye:

- La posición (`geometry_msgs/Pose`) del siguiente punto a alcanzar.
- La velocidad calculada o proporcionada.
- El tiempo correspondiente para ese movimiento.

Dichos mensajes son publicados secuencialmente en el tópico `/pose`, con una frecuencia controlada por un temporizador (1 Hz en esta implementación). Este mecanismo permite que otro nodo pueda recibir y ejecutar la trayectoria paso a paso.

Este nodo es la robustez de nuestro sistema puesto que posee mensajes debugs los cuales verifican cada paso realizado, esto se logró implementando varias condiciones que deben cumplir los parámetros ingresados para asegurarse que el robot no sufra daños debido a picos de voltaje asu vez revisando que cada dato haya sido ingresado de manera correcta.

Implementación del nodo open_loop_ctrl

Este nodo fue desarrollado como parte de un sistema de control en lazo abierto para el robot móvil Puzzlebot. Su objetivo es recibir coordenadas sucesivas junto con una velocidad y un tiempo de movimiento, y ejecutar una trayectoria definida mediante una secuencia de rotaciones y desplazamientos lineales. El nodo se suscribe a un tópico `/pose` que publica mensajes del tipo personalizado `RosarioPath` y publica comandos de velocidad en el tópico `/cmd_vel` para controlar al robot.

Estructura y funcionalidad general:

- Inicialización y configuración

  Al iniciar, el nodo configura un temporizador que ejecuta el bucle de control (`control_loop`) a una frecuencia fija de 10 Hz. También establece un publicador para el tópico `/cmd_vel` y un suscriptor al tópico `/pose`.

- Recepción de trayectorias

  Cada mensaje recibido en `/pose` incluye una posición objetivo (x, y), una velocidad lineal, y un tiempo de desplazamiento. Estos datos se almacenan en una cola (`path_queue`) para su procesamiento secuencial.

- Cálculo de movimientos

  Cuando el robot no está en movimiento y hay trayectorias pendientes en la cola, el nodo calcula el ángulo de rotación requerido para orientarse hacia la siguiente posición, así como el tiempo estimado que debe girar con una velocidad angular constante (1 rad/s). El nodo también obtiene el tiempo de desplazamiento rectilíneo desde el mensaje recibido.

- Máquina de estados

El bucle de control opera mediante una máquina de estados:

- Estado 0 – Rotación: El robot gira sobre su eje hasta alinearse con la dirección del siguiente punto.
- Estado 1 – Movimiento lineal: Una vez alineado, el robot se desplaza hacia el punto destino a la velocidad indicada.
- Estado 2 – Reposo: El robot se detiene completamente antes de procesar el siguiente segmento de trayectoria.

- Normalización angular y control de dirección

  Se implementa una función auxiliar (`delta_angle`) para calcular y normalizar el ángulo entre segmentos de trayectoria, asegurando giros mínimos (ángulos entre $-\pi$ y π). Con esto, también se determina la dirección del giro (sentido horario o antihorario).

- Publicación de comandos de velocidad

  En cada iteración del bucle, el nodo construye un mensaje Twist que representa el estado actual (giro, avance o reposo) y lo publica al tópico `/cmd_vel`.

- Gestión de estados y tiempos

  La transición entre estados está controlada por temporizadores que calculan el tiempo transcurrido desde el inicio del estado actual. Esto asegura una duración precisa tanto del giro como del avance, manteniendo la lógica de lazo abierto.

La robustez en su diseño se debe a varias características que aseguran un manejo eficiente y seguro del robot durante la ejecución de trayectorias. En primer lugar, la implementación de una cola de almacenamiento (`path_queue`) para gestionar múltiples trayectorias permite al sistema procesar y ejecutar movimientos secuenciales sin perder información o dejar de recibir

nuevos datos, lo que facilita la adaptabilidad del nodo a cambios dinámicos en la entrada. Además, el uso de una máquina de estados para gestionar los diferentes tipos de movimientos (rotación, movimiento lineal y reposo) asegura que el robot pueda seguir una secuencia lógica y predecible, evitando errores como movimientos simultáneos inapropiados. La implementación de temporizadores de control y la verificación de condiciones de tiempo en cada fase del movimiento (rotación y avance) proporciona un control preciso sobre la duración de cada fase, lo que mejora la estabilidad y evita sobrecargas. Asimismo, el nodo está diseñado para manejar situaciones en las que el robot no está en movimiento, lo que evita la sobrecarga de procesamiento y permite una transición suave entre trayectorias. Todo esto, combinado con la publicación periódica de comandos al tópico `/cmd_vel`, contribuye a que el sistema sea robusto y eficiente en la ejecución de trayectorias en tiempo real.

Resultados

Parte 1 | Trayectoria cuadrada

Para la trayectoria cuadrada de 2 m por lado, se lanzó el nodo del controlador y se modificó el parámetro `velocity_goal`, para qué el robot hiciera la figura con su velocidad máxima. En la Imagen x, se observa el mensaje de recibimiento del valor del parámetro y la ejecución del nodo.

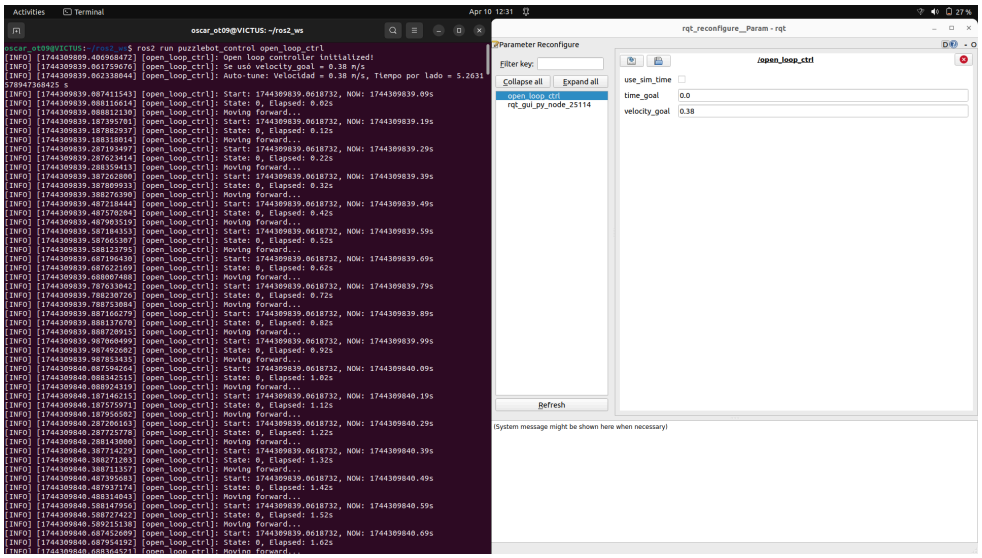


Imagen 2. Modificación de parámetro de velocidad para la trayectoria cuadrada

En la sección de anexos se encuentra el video “Challenge 2 - Parte 1 | ROSario”, en ese se ve el correcto trazado de la figura del cuadrado con la velocidad deseada en el Puzzlebot.

Parte 2 | Trayectoria personalizada

Para la trayectoria de la Imagen 2, obtuvimos los puntos y los almacenamos en el parameter file, como se observa en la Imagen 2.

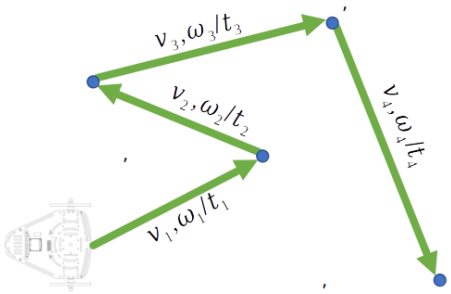


Imagen 3. Trayectoria personalizada

challenge2_ROSario > config > ! params.yaml

```yaml
/path_ROSario:
  ros__parameters:
    coordenadas_x: [1.73205, -0.147334, 2.31468, 3.76468]
    coordenadas_y: [1.0, 1.68404, 2.11816, -0.393313]
    tiempo: [0.0, 0.0, 0.0, 0.0]
    velocidad: [0.38, 0.38, 0.38, 0.38]
    area: 10.0
```

Imagen 4. Los datos ingresados al parameter file

En la Imagen 4, se observa el lanzamiento de los dos nodos del sistema, así como el envío y recepción del mensaje personalizado.

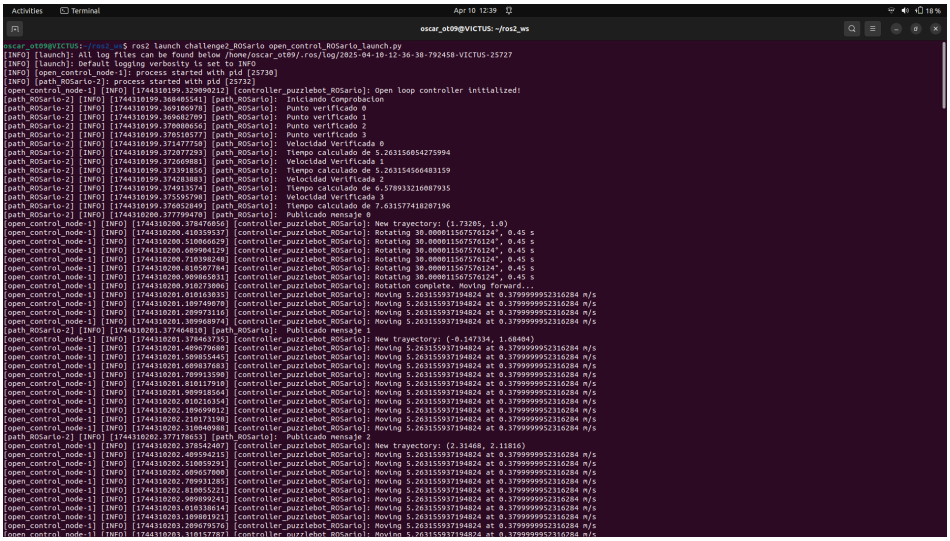


Imagen 5. Retroalimentación de los nodos del sistema

En la sección de Anexo podemos ver un video el cual podemos ver el video “Challenge 2 - Parte 2 | ROSario” podemos ver el correcto trazado de la trayectoria personalizada deseado del Puzzlebot.

Conclusiones

En el desarrollo del reto obserevamos que se cumplio con los objetivos planteados inicialmente tanto en la realización de un cuadrado como en la realización de la figura libre, en el primer reto, con el nodo open_loop_ctrl permitió la ejecución de del cuadrado de manera exitosa debido a la aplicación de una máquina de estados robusta y adaptable. En el segundo caso, el diseño y la implementación de un sistema capaz de recibir trayectorias personalizadas y ejecutadas por medio de parámetros como la velocidad, tiempo y posición íntegra validaciones para evitar errores de entrada y asegurar el comportamiento estable del robot.

Uno de los principales retos enfrentados fue la aparición de errores mecánicos relacionados con los motores del robot, los cuales afectan la precisión de las trayectorias debido a las perturbaciones físicas. Esto evidencia las limitaciones con las que cuenta el lazo de control abierto, debido a que no puede realizar correcciones en las desviaciones por falta de una retroalimentación ni adaptarse dinámicamente a los cambios inesperados, a pesar de esto el sistema logró demostrar ser suficientemente robusto al momento de mantener una ejecución funcional bajo estas condiciones.

Como posibles mejoras, se considera la implementación de un lazo de control cerrado, añadiendo sensores para la retroalimentación que nos permitirá corregir errores en tiempo real y mejorar la precisión del movimiento. También aplicar monitoreos constantes al robot para detectar posibles irregularidades físicas durante recorridos largos o entornos variables.

En resumen, se alcanzaron satisfactoriamente los objetivos iniciales, demostrando la comprensión de los principios del lazo de control abierto y la capacidad de diseñar sistemas modulares, adaptables y seguros para la navegación de robots móviles.

Bibliografía o referencias

En este apartado se anexan los elementos consultadas para el desarrollo del tema de investigación.

geometry_msgs/Pose Documentation. (s. f.). https://docs.ros.org/en/diamondback/api/geometry_msgs/html/msg/Pose.html

Staff, C. E. (2014, 29 agosto). *Open- vs. closed-loop control*. Control Engineering. https://www.controleng.com/open-vs-closed-loop-control/

Difference between Open-Loop & Closed-Loop Control System. (2022, marzo 17). BYJUS; BYJU'S. https://byjus.com/gate/difference-between-open-loop-and-closed-loop-conrol-system/

V., V., Montoya O., J. A., Rios, A., & de La Estimación Odométrica, L. H. M. C. D. E. U. N. R. M. T. D. Y. N. A. P. (s/f). *Scientia Et Technica*. Redalyc.org. Recuperado el 8 de abril de 2025, de https://www.redalyc.org/pdf/849/84916680034.pdf

Anexos

Video Parte 1: https://youtu.be/IUo2u5GVVcM?si=WIo9XnHrFJhbBDLx

Video Parte 2: https://youtu.be/E5nQv5drDL8?si=yl5xq8E2pwyLSIOF