Reto semanal 3. Manchester Robotics

Carlos Adrián Delgado Vázquez A01735818

Alfredo Díaz López A01737250

Juan Paulo Salgado Arvizu A01737223

Bruno Manuel Zamora García A01798275

24 de abril del 2025

Resumen

Se desarrolló un sistema de control en ROS para que el Puzzlebot alcanzara una posición deseada usando odometría y controladores PID. La actividad integró localización, navegación punto a punto y control en lazo cerrado, permitiendo al alumno aplicar conceptos clave del entorno ROS y evaluar el desempeño del robot en tiempo real.

Objetivos

Objetivo principal

Desarrollar en el alumno la capacidad de implementar un sistema de control en lazo cerrado para navegación punto a punto del robot móvil Puzzlebot, utilizando el framework ROS, con el fin de alcanzar una posición deseada mediante el uso de odometría y controladores automáticos.

Objetivos particulares

- Implementar nodos en ROS que permitan leer la odometría del Puzzlebot y estimar su posición en tiempo real.
- Diseñar un algoritmo que calcule el error de posición entre la ubicación actual y la deseada.
- Aplicar y comparar diferentes estrategias de control PID para reducir el error de posición.
- Evaluar el desempeño del sistema mediante herramientas de visualización en ROS.
- Fomentar el análisis crítico y la mejora continua del comportamiento del robot en función de los resultados obtenidos.

Introducción

La robótica móvil ha experimentado un crecimiento significativo en las últimas décadas, impulsada por la necesidad de automatizar tareas en entornos dinámicos y no estructurados como en Ilustración 1. Entre los diversos tipos de robots móviles, los de tracción diferencial destacan por su simplicidad mecánica y versatilidad en aplicaciones como exploración, vigilancia y asistencia en entornos industriales y domésticos.

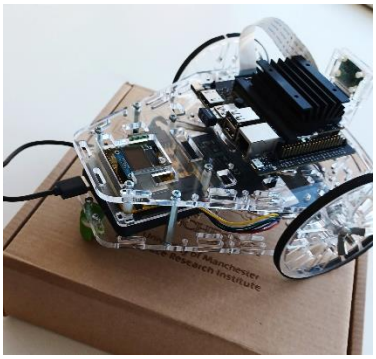

Ilustración 1 Robot móvil Puzzlebot

Para garantizar un desplazamiento preciso y confiable, es fundamental implementar estrategias de control que permitan al robot seguir trayectorias planificadas y adaptarse a perturbaciones externas. En este contexto, el control en lazo cerrado, como en la Ilustración 2, se presenta como una solución eficaz, ya que utiliza la retroalimentación de las variables del sistema para minimizar el error entre la posición deseada y la real del robot. Cuando queremos que la señal de salida

alcance un valor determinado el sistema tiene que medir continuamente dicha señal. En este caso el sistema es realimentado, y hablamos de un sistema automático de lazo cerrado. Por tanto, son aquellos en los que la salida influye sobre la señal de entrada (de trabajo., 1. Evolución. n.d). Las propiedades del sistema en lazo cerrado dependerán del desempeño del controlador y la configuración inicial del robot, así la convergencia local y estabilidad pueden ser obtenidas de las formas (Aviles, C. O. C., Díaz, A. M., & Angeles, A. R. n.d).

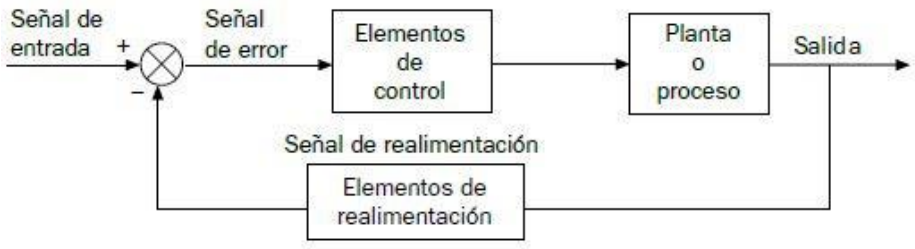

Ilustración 2 Lazo cerrado

Uno de los métodos de control más ampliamente utilizados en sistemas de robótica móvil es el controlador PID (Proporcional-Integral-Derivativo). Este controlador ajusta la señal de control en función del error actual, la acumulación histórica del error y la tasa de cambio de este, lo que permite una respuesta rápida y estable del sistema, en la Ilustración 3, se representa la estructura de un controlador PID. En este, se intenta emular el comportamiento de un controlador PID para controlar los perfiles de velocidad de los motores de tracción un robot móvil tipo diferencial (Millán et al., 2016).

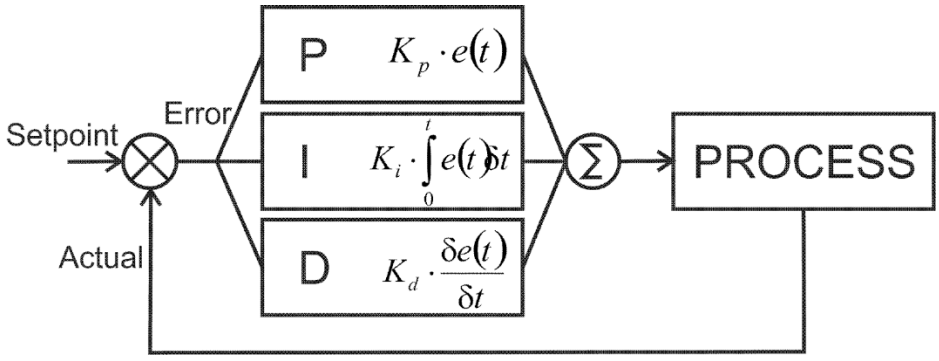

Ilustración 3 Controlador PID

Sin embargo, la implementación efectiva de un controlador PID requiere una adecuada sintonización de sus parámetros. Además, es esencial considerar aspectos como el cálculo y análisis de errores, tanto sistemáticos como aleatorios, que pueden afectar la precisión del sistema. El estudio y comprensión de estos errores permiten mejorar la robustez del controlador y, por ende, la fiabilidad del robot en diversas condiciones de operación.

Uno de los aspectos fundamentales en la implementación de un sistema de control en lazo cerrado es el cálculo del error, entendido como la diferencia entre la referencia deseada y la medida real obtenida por los sensores del robot. En el contexto del control de un robot móvil diferencial, este

error puede tener componentes tanto lineales como angulares, que deben ser calculados y procesados en cada instante de tiempo para actualizar la acción de control de manera precisa, nos podemos basar en el cálculo del error relativo, Ilustración 4, para calcular el error entre una distancia real y la indicada por el puzzlebot para hacerlo exacto.

$$\textit{Error relativo} = \frac{\textit{Error absoluto}}{\textit{Error real}}$$

Ilustración 4 Calculo de error relativo

En entornos reales, el cálculo del error también debe considerar factores como el retardo en la comunicación con los sensores, la cuantización de datos y la precisión de los actuadores. Estos aspectos hacen necesario un enfoque cuidadoso y sistemático en el diseño del algoritmo de control, para garantizar la robustez del sistema ante la inevitable presencia de incertidumbre (Márquez-Rubio, J., Del-Muro-Cuéllar, B., Velasco-Villa, M., & Álvarez-Ramírez, J. n.d.).

La robustez de un sistema de control en lazo cerrado se refiere a su capacidad para mantener un desempeño aceptable frente a incertidumbres en el modelo, perturbaciones externas o variaciones en los parámetros del sistema. Sin embargo, en el caso de sistemas inestables con retardo, esta robustez se ve severamente comprometida. Tal como se ha analizado, una mínima desviación en la modelación puede ser suficiente para inestabilizar el sistema en lazo cerrado. Esta sensibilidad crítica está estrechamente relacionada con la razón entre el retardo de tiempo (τ) y la constante de tiempo inestable (τ_{an}), donde un aumento en τ o una disminución en τ_{an} reduce la robustez del sistema. Por ello, al diseñar un controlador para robots móviles o cualquier sistema con dinámica inestable, es esencial considerar esta relación y aplicar técnicas que mejoren la tolerancia a errores estructurales y de modelado (Márquez-Rubio et al., n.d.). Un controlador robusto, tiene una estructura similar a la Ilustración 5.

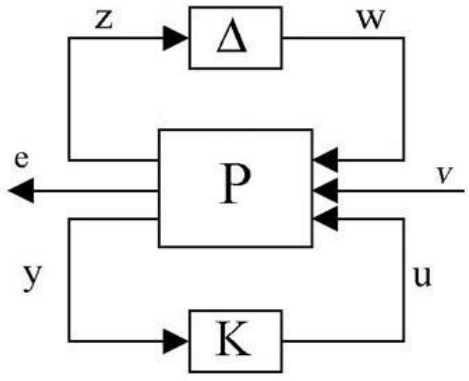

Ilustración 5 Controlador robusto

Solución del problema

Medición de errores en distancia lineal y angular (giro a la derecha y giro a la izquierda):

-Distancia lineal:

Distancia real	Distancia medida (encoders)	Error
1 m	.945m	5.5
1 m	.943m	5.7
3 m	2.823 m	5.9
3 m	2.817 m	6.1
5 m	4.685 m	6.3
5 m	4.675 m	6.5

El error promedio para la distancia lineal es de -5.5%, por lo que para el código final será necesario agregar esa diferencia para compensar el error.

-Distancia angular (izquierda y derecha):

Giro a la izquierda			**Giro a la derecha**		
Angulo real (grados)	**Angulo medido(encoder)**	**Error**	**Angulo real (grados)**	**Angulo medido(encoder)**	**Error**
90	84.11	6.54%	90	81.76	9.15%
180	166	7.3%	180	164.5	7.8%
360	332	7.8%	360	332.5	7.69%

El error promedio para estas pruebas fue de -7.71%, por ello, será necesario compensar este errar, agregando esta cifra al ángulo medido para que se acerque lo más posible al ángulo real

El cálculo de los anteriores errores es necesario para que el PID funcione adecuadamente, puesto que si el PID solo recibiera los datos medidos de los encoders tendría mediciones incorrectas porque como se observa tanto a la velocidad lineal como angular miden menos de los que realmente nosotros medidos visualmente.

Cálculo de valores para el controlador (PID,P, PI):
Se considero un PID tanto para la velocidad lineal, como para la velocidad angular. Los cuales fueron calculados empiricamente, calculando como primera instancia el KP, hasta que alcance el valor, y posteriormente, se tunearon tanto la KI como la KD.

Los valores para el PID para la velocidad lineal son:
P =1.6
I = 0.001
D = 0.05

Los valores para el PID para la velocidad lineal son:
P =1.0
I = 0.01
D = 0.05

Archivo de parámetros .yaml:

```
waypoints:
  - [1.2, 0.0]
  - [1.2, 1.2]
  - [0.0, 1.2]
  - [0.0, 0.0]
```

Ilustración 6 Archivo de coordenadas

Para poder establecer los puntos de coordenadas que debe alcanzar el robot se hizo un archivo .yaml como el de la imagen. Esto con el motivo, de no modificar estas coordenadas de manera manual en el código, se hizo con la finalidad de hacer esta modificación más sencilla y práctica.

Contenido de los 3 nodos:
Odometria:

El código implementa un nodo de ROS2 llamado puzzlebot_odometry, el cual calcula y publica la pose (posición y orientación) de un robot diferencial en tiempo real usando las velocidades angulares medidas de sus ruedas. Se explica cada parte de su funcionamiento:

1. **Inicialización del Nodo:**
    a. El nodo se llama puzzlebot_odometry.
    b. Se inicializan las variables de estado del robot: posición (X, Y) y orientación (Th) en 2D.
    c. Se configuran los parámetros físicos del robot: distancia entre ruedas (_l) y radio de rueda (_r), así como el tiempo de muestreo (_sample_time).
2. **Suscripción a las velocidades de ruedas:**
    a. El nodo se suscribe a dos tópicos: VelocityEncR y VelocityEncL, que proporcionan las velocidades angulares (wr, wl) de las ruedas derecha e izquierda, respectivamente.
3. **Cálculo de velocidad y orientación:**
    a. A partir de las velocidades angulares recibidas, el código calcula:
        i. La velocidad lineal del robot: V = 0.5 * (v_r + v_l)
        ii. La velocidad angular del robot: Omega = (1 / _l) * (v_r - v_l)
    b. Estas velocidades se usan para integrar la pose del robot utilizando un modelo de odometría diferencial.
4. **Integración para estimar la pose:**
    a. Se calcula el desplazamiento en x y y usando la orientación actual (Th) y la velocidad lineal V mediante:
        i. delta_x = V * dt * cos(Th)

ii. delta_y = V * dt * sin(Th)

b. La orientación Th se actualiza con la velocidad angular Omega y se normaliza en el rango [-π, π].

5. **Publicación del mensaje de odometría:**
   a. El nodo publica un mensaje de tipo nav_msgs/Odometry en el tópico odom, el cual contiene:
      i. Posición: coordenadas x, y (en metros)
      ii. Orientación: convertida a un cuaternión usando transforms3d
      iii. Velocidad lineal (linear.x) y angular (angular.z)
   b. Esto permite a otros nodos del sistema (por ejemplo, un visualizador o un controlador) acceder a la estimación de estado actual del robot.
6. **Temporizador de ejecución periódica:**
   a. Un temporizador ejecuta el método run() cada 1/200 segundos (frecuencia de 200 Hz), asegurando un seguimiento continuo de la odometría.

Path_generator:

El código implementa un nodo de ROS2 llamado path_generator, cuya función principal es gestionar la navegación hacia una serie de puntos objetivo (waypoints), corrigiendo los errores sistemáticos de la odometría y publicando la pose corregida del robot. Se describe detalladamente lo que hace el nodo:

1. Carga y corrección de odometría

- El código se suscribe al tópico odom, el cual proporciona la odometría calculada sin correcciones.
- A partir de la orientación en cuaterniones, el código extrae el ángulo de orientación (raw_th) usando trigonometría inversa.
- El código aplica correcciones porcentuales:
  - A las posiciones x e y (un 6% de incremento).
  - A la orientación θ (un 5% de corrección).
- Esta corrección busca compensar errores sistemáticos en la estimación del movimiento del robot.

2. Publicación de odometría corregida

- El código reescribe el mensaje original de odometría (msg) con los valores corregidos.
- Publica el mensaje corregido en el tópico corrected_odom, permitiendo a otros nodos usar una mejor estimación del estado real del robot.

3. Carga de waypoints desde archivo YAML

- El código carga una lista de coordenadas (waypoints) desde el archivo waypoints.yaml, ubicado en la carpeta config del paquete puzzlebot_localisation.
- Cada waypoint representa un objetivo a alcanzar en el plano.

4. Publicación de metas (goal_pose)

- El código publica la posición del siguiente waypoint en el tópico goal_pose, como un mensaje tipo PoseStamped.
- La publicación se realiza usando la referencia odom como sistema de coordenadas.
- El código inicia automáticamente publicando el primer punto objetivo.

5. Verificación de llegada a un waypoint

- Un temporizador ejecuta la función _check_goal() cada 0.05 segundos.
- Esta función verifica si la pose corregida del robot está dentro de un umbral (3 cm) respecto al objetivo actual.
- Si el robot alcanza el waypoint, se actualiza el índice de waypoint y se publica el siguiente.
- Una vez alcanzados todos los puntos, el código detiene la navegación y muestra un mensaje informativo.

Función general del nodo

En conjunto, el nodo PathGenerator actúa como el módulo de navegación en lazo abierto, encargado de:

- Proveer metas secuenciales al robot.
- Corregir errores sistemáticos de la odometría.
- Informar el progreso hacia cada punto objetivo.

Este nodo es ideal para entornos conocidos, donde los waypoints pueden predefinirse. En un sistema más avanzado, podría complementarse con un módulo de planeación global y detección de obstáculos.

Controller:

El código define un nodo de ROS2 llamado controller, cuya función principal es controlar el movimiento de un robot diferencial para que alcance una secuencia de puntos objetivos (waypoints) utilizando controladores PID tanto para la distancia como para el ángulo. Se describe su funcionamiento por partes:

1. Inicialización del nodo

- El nodo se llama controller.

- Se definen dos controladores PID:
  - Uno para la distancia lineal al objetivo (Kp_d, Ki_d, Kd_d).
  - Otro para la orientación angular (Kp_th, Ki_th, Kd_th).
- Se inicializan variables para errores acumulados e históricos, así como límites de saturación para velocidades lineales y angulares:
  - Zona muerta: evita comandos muy pequeños.
  - Saturación: limita el máximo de las velocidades para prevenir comportamientos no deseados.

2. Subscripciones

- El nodo se suscribe a:
  - corrected_odom: posición y orientación actual corregidas del robot.
  - goal_pose: posición del objetivo actual a alcanzar.

3. Manejo de odometría

- El callback odom_cb() actualiza la pose actual del robot (x, y, theta) a partir de los datos de odometría corregida (corrected_odom).
- La orientación se convierte de cuaterniones a ángulo en radianes.

4. Recepción de metas

- El callback goal_cb() recibe la siguiente meta (gx, gy) y reinicia los acumuladores del PID para asegurar una respuesta limpia al nuevo objetivo.

5. Control PID

- La función control_loop() se ejecuta cada 0.05 s y realiza lo siguiente:
  - Calcula errores de distancia (e_d) y ángulo (e_th) hacia el objetivo.
  - Aplica un PID angular para calcular el comando w_cmd.
  - Si el error angular es mayor a 15° (fase de giro), el robot no avanza linealmente (v_cmd = 0).
  - Si el ángulo está alineado, se activa el PID de distancia para calcular v_cmd.

6. Zona muerta y saturación

- Si el valor absoluto de v_cmd o w_cmd es menor que sus mínimos definidos, se anulan (zona muerta).
- Luego, ambos comandos se saturan para mantenerlos dentro de rangos seguros definidos por max_v y max_w.

7. Publicación de comandos de velocidad

- Se construye un mensaje Twist y se publica en el tópico cmd_vel, el cual controla directamente al robot.

Función general del nodo

El nodo controller es el encargado de ejecutar la navegación reactiva del robot, asegurando que:

- Se mantenga alineado con el objetivo antes de avanzar.
- Se ajusten constantemente las velocidades para mantener una trayectoria suave y eficiente.
- Se respeten los límites físicos del robot evitando sobresaturación.

Este enfoque permite al robot seguir una ruta definida por waypoints con precisión usando únicamente información de odometría corregida, sin sensores externos.

Resultados

Despues de las pruebas de funcionamiento se hicieron 3 figuras que trazan un zigzag, un cuadrado y un triángulo. En la siguiente liga se presentan los videos de cómo funciona el puzzlebot realizando dichas figuras.
https://drive.google.com/drive/folders/11eihgept-HmVKvE88mRNvufFgomtzHYT

Conclusiones

Los objetivos de la actividad se cumplieron al implementar un sistema de control en lazo cerrado en ROS, que permitió al Puzzlebot alcanzar posiciones deseadas utilizando odometría y controladores PID. La estructura modular de ROS facilitó la integración de los componentes necesarios para la localización, el cálculo del error y el control del movimiento. Si bien el robot logró navegar de forma efectiva, se presentaron algunas imprecisiones, principalmente por errores acumulados en la odometría y la sensibilidad del sistema a la sintonización del controlador. Como mejora, pensamos complementar la odometría con sensores adicionales como visión por computadora, así como aplicar técnicas más precisas para el ajuste de los parámetros del controlador. En general, la actividad permitió aplicar y consolidar conceptos fundamentales de localización, control y navegación dentro de un entorno real, fortaleciendo las competencias prácticas en robótica móvil. Además en las pruebas finales, notamos un error mecanico en el robot, que hacía que los motores se vencieran, evitando la presición de las figuras y los resultados reales, por lo que, corrigiendo estos, podríamos tener un controlador más eficiente.

Bibliografía o referencias

Aviles, C. O. C., Díaz, A. M., & Angeles, A. R. (n.d.). *Control de Sincronización de un Robot Móvil Diferencial*. Amca.mx. Recuperado el 24 de abril de 2025, de https://amca.mx/memorias/amca2010/Art%C3%ADculos/Rob%C3%B3tica/amca2010_submission_63.pdf

de las formas de trabajo., 1. Evolución. (n.d.). *CONTROL Y ROBÓTICA*. Xunta.es. Recuperado el 24 de abril de 2025, de https://centros.edu.xunta.es/ieseduardopondal/tecnoweb/temas_informatica/robotica1.pdf

De una magnitud f, P. D. el V. R., De ella, se R. M., De un n, N. M. la C., De sucesos o por comparaci, U., De la magnitud en cuesti, on C. U. U. de M. P. el P. P. es I. D. el V. V., Sufrir, on T. L. V., La limitada precisi, an E. D. a., Del observador, on de L. A. de M. y. L. S., Intr, C. a. O., & De la materia, N. de la E. (n.d.). *Calculo de errores y presentacion de resultados experimentales*. Www.uv.es. Recuperado el 24 de abril de 2025, de https://www.uv.es/jbosch/PDF/CalculoDeErrores.pdf

Márquez-Rubio, J., Del-Muro-Cuéllar, B., Velasco-Villa, M., & Álvarez-Ramírez, J. (n.d.). *Control basado en un esquema observador para sistemas de primer orden con retardo*. https://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1665-27382010000100006

Millán, G. H., Gonzales, L. H. R., & López, M. B. (2016). *Implementación de un controlador de posición y movimiento de un robot móvil diferencial*. https://www.redalyc.org/journal/2570/257046835010/html/