Instituto Tecnológico de Estudios Superiores de Monterrey

Implementación de robótica inteligente & Manchester Robotics

Actividad R_2. Cálculo del error de un robot móvil diferencial

Profesores:

Rigoberto Cerino Jiménez

Dr. Mario Martinez

Integrantes

Daniel Castillo López A01737357

Emmanuel Lechuga Arreola A01736241

Paola Rojas Domínguez A01737136

22 de Abril de 2025

Índice

Resumen

El resumen presenta una síntesis de los aspectos más relevantes del proceso de investigación y desarrollo llevado a cabo, destacando los objetivos, la metodología empleada y los principales resultados obtenidos.

Esta práctica tuvo como propósito reforzar el uso de herramientas básicas de ROS mediante la interacción directa con el Puzzlebot. Para lograrlo, desarrollamos dos nodos principales: uno para estimar la posición del robot usando odometría, y otro para calcular el error de distancia y orientación hacia una posición deseada.

Primero, controlamos el robot utilizando el nodo teleop_twist_keyboard, lo que nos permite moverlo manualmente y verificar la estimación de su posición en tiempo real. Después, establecimos una meta fija y programamos un nodo que constantemente calcula la distancia (*ed*) y el ángulo (*eθ*) desde la posición actual hasta la deseada. Se cuidó que los ángulos estuvieran dentro de un rango correcto (normalizados entre -π y π) para evitar errores de cálculo.

Finalmente, todo el código desarrollado fue organizado en un workspace y subido a una carpeta de GitHub llamada “Tareas_IRI”, donde incluimos los nodos, archivos de configuración y launch, además de un video mostrando cómo el robot publica los errores en tiempo real mientras se mueve. El trabajo fue realizado en equipo y se compartió en la plataforma Canvas para su evaluación.

Objetivos

En esta sección se presentan el objetivo general y los objetivos particulares del reto, los cuales están enfocados en encontrar el error de trayectoria de un Puzzlebot.

Objetivo general: Desarrollar un sistema básico de localización y navegación punto a punto en lazo abierto para el Puzzlebot utilizando ROS2, integrando conceptos de odometría y cálculo de errores en tiempo real.

Objetivos específicos:

- Diseñar un nodo en ROS2 que calcule la posición del robot a partir de los datos proporcionados por los encoders.
- Verificar la estimación de la posición del robot mediante la ejecución del nodo de teleoperación teleop_twist_keyboard.
- Desarrollar un nodo que calcule y publique en tiempo real errores de distancia y orientación respecto a una posición objetivo definida por el usuario.
- Evaluar el comportamiento del sistema de navegación

Introducción

La introducción presenta el tema de investigación, proporcionando el contexto necesario para comprender la relevancia del reto a realizar. Se ofrece una visión general que facilita la construcción de un esquema mental sobre las metodologías utilizadas, además de los conceptos y tecnologías que se abordarán en el desarrollo del reporte.

Odometría de un robot móvil diferencial

En los robots móviles diferenciales, la odometría es un método fundamental para estimar la posición y orientación del robot a lo largo del tiempo, utilizando la información proporcionada por los encoders de las ruedas. Este proceso se basa en un modelo cinemático diferencial que describe cómo las velocidades individuales de las ruedas determinan el desplazamiento del robot en un plano (ManchesterRoboticsLtd, s. f.-h).

$$V = r\frac{\omega_R + \omega_L}{2} \tag{1}$$

$$\omega = r\frac{\omega_R - \omega_L}{l} \tag{2}$$

Donde:

- ω_R es la velocidad angular de la rueda derecha
- ω_L es la velocidad angular de la rueda izquierda
- V corresponde a la velocidad lineal del robot
- ω corresponde a la velocidad angular del robot
- l es la distancia entre las ruedas
- r es el radio de las ruedas

Estas velocidades se integran con el paso del tiempo para estimar la posición del robot (x, y) y su orientación θ. Para ello, se utiliza una forma discretizada del modelo cinemático implementada con el método de Euler (ManchesterRoboticsLtd, s. f.-h).

$$x_{k+1} = x_k + V \cdot cos(\theta_k) \cdot dt \tag{3}$$

$$y_{k+1} = y_k + V \cdot cos(\theta_k) \cdot dt \tag{4}$$

$$\theta_{k+1} = \theta_k + \omega \cdot dt \tag{5}$$

Este procedimiento, conocido como dead reckoning, es útil en entornos donde el robot no tiene acceso a sensores externos como GPS. Sin embargo, hay que considerar que su precisión se ve afectada por errores acumulativos provocados por el deslizamiento de las

ruedas o la discretización del tiempo. Por ello, la odometría suele complementarse con técnicas de localización más robustas.

Cálculo del error en un robot móvil diferencial.

El cálculo del error es esencial para el diseño de sistemas de control destinados a que guíen al robot hacia una posición deseada. Este proceso se conoce como estabilización de punto, y consiste en definir una pose objetivo $X_g = (x_g, y_g, \theta_g)$ que el robot debe alcanzar, comparándola con su pose actual estimada $X_r = (x_r, y_r, \theta_r)$ (ManchesterRoboticsLtd, s. f.-h).

A partir de esta comparación, se obtiene un vector error en coordenadas cartesianas y orientación:

$$e_x = x_g - x_r \tag{6}$$

$$e_y = y_g - y_r \tag{7}$$

$$e_\theta = atan2(e_y, e_x) - \theta_r \tag{8}$$

Este último término descrito en la ecuación (8), representa el ángulo entre la orientación actual del robot y la línea que conecta su posición con el objetivo. Dado que los ángulos pueden crecer indefinidamente, es común utilizar una función de envoltura para limitar los valores de e_θ dentro del rango $[-\pi, \pi]$, lo que garantiza un control estable (ManchesterRoboticsLtd, s. f.-h).

El error de distancia es una magnitud escalar que representa que tan lejos está el robot de su destino:

$$e_d = \sqrt{e_x^2 + e_y^2} \tag{9}$$

Con estos errores definidos, se puede aplicar una ley de control proporcional simple:

$$V = K_d \cdot e_d \tag{10}$$

$$\omega = K_\theta \cdot e_\theta \tag{11}$$

Donde K_d y K_θ son constantes que ajustan la respuesta del sistema. Esta forma de control es suficiente para lograr un comportamiento estable en la mayoría de los casos. El cálculo de errores, es conjunto con la odometría, permite al robot moverse de forma autónoma hacia una meta definida en un entorno conocido.

Solución del problema

En esta sección se desarrolla una solución basada en la interacción con el Puzzlebot, con el objetivo de comprender las coordenadas de la posición del robot y establecer una coordenada deseada, empleando conceptos como la odometría y la navegación punto a punto. Con esto, se plantea una estrategia que consiste en desarrollar dos nodos: uno encargado de calcular la localización del robot y otro que determine el valor del error en la distancia y el error en theta.

Nodo puzzlebot_odometry

La función de este nodo es implementar una estimación de posición por odometría para nuestro Puzzlebot, utilizando las velocidades proporcionadas por los encoders. Para ello, se establece una base con posición inicial cero tanto en el eje x como en el eje y.

- **Configuración de los parametros**

Se define una orientación inicial con el nombre Th, así como valores físicos como la distancia entre ruedas y el radio de las mismas. Con estos datos, se establece un tiempo de muestreo y una frecuencia de actualización para la odometría.

- **Localización**

Con estos parámetros definidos y mediante la suscripción a los tópicos de velocidad de los encoders, podemos calcular la velocidad lineal del robot. Esto se realiza con la ayuda de un temporizador que recopila periódicamente los datos de los encoders. Así, es posible calcular las velocidades tangenciales de las ruedas, siendo la velocidad lineal del robot el promedio de ambas, y también se determina la velocidad angular del mismo. A partir de estas velocidades, se procede a calcular las posiciones del robot integrando dichas velocidades. Esta integración se lleva a cabo utilizando el método de Euler, basándose en el valor de dt (tiempo entre muestras) y en la acumulación de las velocidades sobre los valores anteriores.

Nodo controller

Este nodo tiene como objetivo calcular los errores de navegación que presenta el Puzzlebot con respecto a un punto deseado. Para ello, se suscribe a los datos de odometría publicados por el nodo puzzlebot_odometry y, a partir de esta información, determina el error en la distancia y en la orientación respecto al objetivo. Este proceso es primordial para el funcionamiento del control de navegación punto a punto.

- **Configuración de los parámetros**

Dentro del nodo se establecen como parámetros iniciales las coordenadas del punto objetivo, en este caso funcionan como el setpoint, definido por xg = 1.0 y yg = 0.0. A partir de esta posición deseada, el robot calculará sus errores actuales cada vez que se reciba un mensaje del canal odom

- **Cálculo de errores**

Dentro de este nodo se extraen las coordenadas actuales del robot. La orientación, que se obtiene en forma de cuaterniones, se convierte en un ángulo de orientación en el plano (ángulo YAW).

Tomando como referencia las fórmulas presentadas del número 6 al 9, es posible calcular el error en función de la posición actual con respecto a la posición deseada, esto en el contexto de coordenadas cartesianas. Para el caso del error angular, este se determina como la diferencia entre la dirección deseada hacia el punto objetivo (como se muestra en la fórmula 8) y la orientación actual del robot. Este valor de error en theta (orientación) se normaliza entre $-\pi$ y , con el fin de evitar discontinuidades durante el control de orientación. Esta normalización es esencial para que el robot siempre tome la ruta de menor giro al alinearse con el objetivo.

Resultados

En este apartado se insertan las evidencias del funcionamiento del reto, agregando descripciones de lo que representa cada una de estas, así como un análisis de tales resultados, para saber si se consideran resultados satisfactorios o completos.

Para validar el correcto funcionamiento del sistema de odometría y cálculo de errores, se realizaron diversas pruebas utilizando el robot Puzzlebot. En primer lugar, el robot fue operado manualmente mediante el nodo, teleop_twist_keyboard permitiendo observar en tiempo real la estimación de su posición calculada a partir de los datos de los encoders.

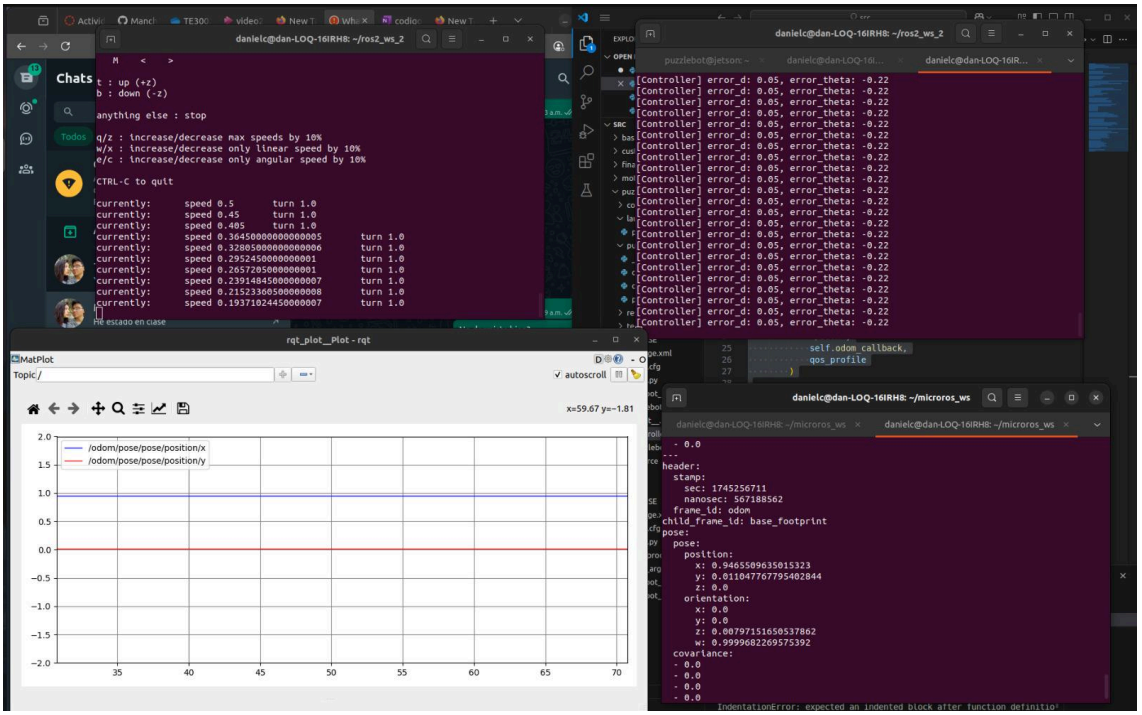

Figura 1. Terminal nodo controller error distancia

Durante el desplazamiento del robot, se ejecutó simultáneamente un nodo encargado de calcular los errores de distancia y orientación, hacia una posición objetivo fija (en este caso siendo un metro). Estos errores fueron impresos en consola como podemos ver en la Figura 1, mostrando de mejor manera los intervalos regulares, y se observaron variaciones coherentes conforme el robot se acercaba o alejaba de la meta. En el video de youtube (https://youtu.be/WDXfdODfxQ4) se puede apreciar el funcionamiento.

En particular, se evidenció que el error de distancia disminuye consistentemente cuando el robot se dirige en línea recta hacia el objetivo, presentando un promedio de 0.3 a 0.5 siendo el margen de error de nuestro Puzzlebot.

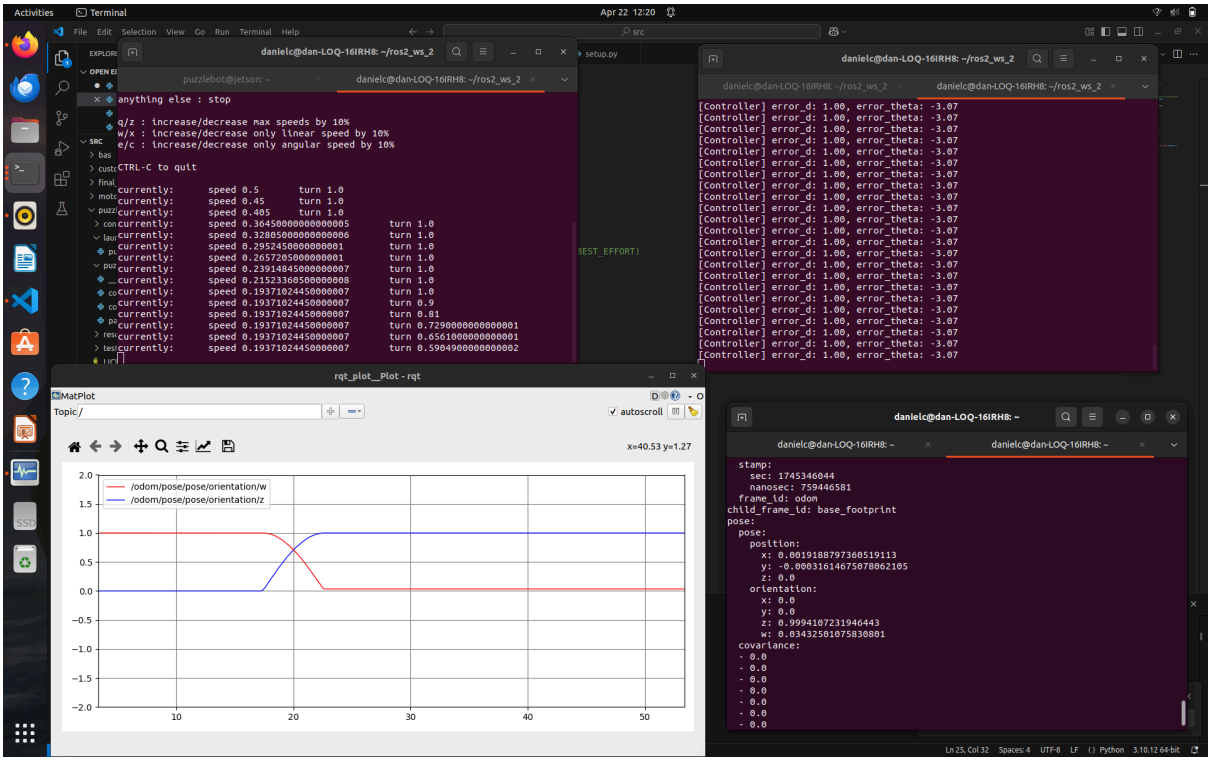

Figura 2: Terminal nodo controller error theta

Las rotaciones en el espacio tridimensional están representadas con cuaterniones, un cuaternión tiene cuatro componentes, donde x, y, z representan el eje de rotación y ω está relacionado con el coseno del ángulo de giro. Entonces, cuando nuestro puzzlebot gira 180° alrededor de su eje vertical (eje Z), se puede apreciar un cambio en los valores de z y ω, con ω acercándose a 0, mientras que z se acerca a 1 tal y como se muestra en la Figura 2.

El error en este caso acercándose a 3.14 debido a que los ángulos están cerrados en un círculo que sería de $-\pi\, y\, \pi$, lo que nuestro robot al dar un giro de 180° lo acerca al 3.14, mostrando que su error está en promedio de 0.2 y 0.4.

El video de youtube (https://youtu.be/yS1Wie72iOc) muestra el funcionamiento del giro de 180°

Conclusiones

Por último, se presentan los logros alcanzados a lo largo del desarrollo del reto, analizando el grado de cumplimiento de los objetivos planteados. Se evalúa si estos fueron alcanzados en su totalidad, identificando las razones detrás de su éxito o, en caso contrario, los factores que pudieron haber limitado su cumplimiento. Asimismo, se proponen posibles mejoras a la metodología utilizada, con el propósito de optimizar los resultados obtenidos.

La presente entrega permitió desarrollar un sistema funcional de localización básica y cálculo de errores para navegación utilizando ROS 2. A través de la implementación de nodos específicos, se logró estimar la posición del robot a partir de datos de encoders, así como calcular y reportar errores de orientación y distancia respecto a una meta definida (dependiendo de la distancia y recorrido).

Los objetivos propuestos fueron alcanzados en su totalidad. Se logró verificar la estimación de la posición mediante control manual, y se desarrolló un sistema que permite monitorear los errores de navegación en tiempo real, lo cual es un paso esencial para implementar un controlador de movimiento más complejo en el que se aplique más robustez.

Entre los desafíos encontrados, destaca la sensibilidad de la odometría ante deslizamientos o pequeñas imprecisiones en los datos de los encoders y por parte del robot, lo cual podría afectar la precisión a largo plazo. Sin embargo, para entornos controlados o simulados, el sistema se comporta de manera estable y confiable.

Referencias

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

ManchesterRoboticsLtd. (s. f.-g). *TE3002B_Intelligent_Robotics_Implementation_2025/Week3/Presentations/PDF/MCR2_Closed_Loop_Control_v3.pdf at main · ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025*. GitHub. https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/blob/main/Week3/Presentations/PDF/MCR2_Closed_Loop_Control_v3.pdf

Freddy. (s. f.). *Implementacion-de-robotica-inteligente-2025/Implementación de Robótica Inteligente (sesion 5).pptx at main · freddy-7/Implementacion-de-robotica-inteligente-2025*. GitHub. https://github.com/freddy-7/Implementacion-de-robotica-inteligente-2025/blob/main/Implementaci%C3%B3n%20de%20Rob%C3%B3tica%20Inteligente%20(sesion%205).pptx

Freddy. (s. f.-b). *Implementacion-de-robotica-inteligente-2025/Implementación de Robótica Inteligente (sesion 8).pptx at main · freddy-7/Implementacion-de-robotica-inteligente-2025*. GitHub. https://github.com/freddy-7/Implementacion-de-robotica-inteligente-2025/blob/main/Implementaci%C3%B3n%20de%20Rob%C3%B3tica%20Inteligente%20(sesion%208).pptx