Error de un robot móvil diferencial

Implementación de Robótica Inteligente

Integrantes:

A017314050 | Eleazar Olivas Gaspar

A01735696 | Azul Nahomi Machorro Arreola

A01732584 | Angel Estrada Centeno

A01735692 | Arick Morelos del Campo

Profesores:

Rigoberto Cerino Jimenez

Alfredo Garcia Suarez

Juan Manuel Ahuactzin Larios

Cesar Torres Huitzil

Abril 2024

Resumen:
Este documento describe el proceso de desarrollo de un nodo en ROS específicamente diseñado para publicar los cálculos de los errores de posición y orientación. El objetivo de este nodo es permitir un control más preciso del robot al proporcionar información constante y actualizada sobre su posición relativa a un punto objetivo. Lo que buscamos es establecer un objetivo para el robot y luego hacer que este se mueva por el entorno mientras se monitorean y actualizan dos parámetros a corregir: la distancia hasta el objetivo y el ángulo hacia el mismo. Es fundamental recordar que todos los cálculos angulares deben mantenerse dentro de los límites de un círculo completo, es decir, 360 grados o 2π radianes, para evitar discontinuidades y errores en la interpretación de los ángulos.

Este enfoque no solo facilita una navegación más precisa, sino que también es esencial para el desarrollo de estrategias de corrección y ajuste en tiempo real, permitiendo al robot ajustar su trayectoria de acuerdo con las desviaciones detectadas respecto a la ruta planeada. Implementar este nodo en ROS fortalece las capacidades autónomas del robot, proporcionando una herramienta fundamental para tareas de navegación autónoma en entornos complejos. A continuación se explicará en este documento la teoría, metodología y proceso para resolver esta actividad y se expondrán los resultados.

Objetivos:
Desarrollar cada vez más las habilidades en el manejo del framework ROS, también buscamos incrementar el grado de interacción con ROS facilitando la implementación de funciones permitiéndonos interactuar con el Puzzlebot. Otro de los objetivos de este trabajo es obtener conocimientos sobre la determinación de las coordenadas de la posición del robot en todo momento para introducir una posición deseada en el Puzzlebot, buscamos calcular el error de posición en tiempo real, integrando conceptos previos de localización, como la odometría. todo esto para tener un buen manejo de la navegación punto a punto en lazo abierto utilizando ROS.

Introducción:
La odometría en un robot móvil diferencial es un procedimiento fundamental que permite estimar la posición y orientación del robot en función de las mediciones de las rotaciones de sus ruedas. En un sistema de este tipo, se emplean sensores, como encoders, para registrar con precisión la cantidad de rotación de cada rueda a medida que el robot se desplaza. Con esta información, se calcula la distancia recorrida por cada rueda, lo que proporciona una estimación de la trayectoria del robot.

Además de medir la distancia recorrida, la odometría también considera el ángulo de giro del robot. Esto se logra mediante el seguimiento de la diferencia entre las velocidades de las ruedas izquierda y derecha, lo que permite determinar el cambio en la orientación del robot. Al combinar la información sobre la distancia recorrida y el ángulo de giro, la odometría proporciona una estimación de la posición y orientación del robot en un marco de referencia global.

Sin embargo, la precisión de la odometría puede verse comprometida por diversos factores. Por ejemplo, el deslizamiento de las ruedas, las irregularidades en la superficie del terreno y los errores en los sensores pueden generar discrepancias entre la estimación de la odometría y la posición y orientación reales del robot. Estas discrepancias se conocen como errores de odometría.

Para abordar estos errores y mejorar la precisión de la estimación de la posición y orientación del robot, se recurre a técnicas de cálculo del error. Estas técnicas implican la comparación de la posición y orientación estimadas por la odometría con mediciones precisas de la posición y orientación reales del robot. Con base en estas comparaciones, se pueden aplicar correcciones para ajustar la estimación de la odometría y reducir los errores.

Una técnica comúnmente utilizada para corregir errores de odometría es la localización simultánea y el mapeo (SLAM). Este enfoque permite al robot construir un mapa de su entorno mientras estima simultáneamente su posición dentro de ese mapa. Al fusionar la información de la odometría con datos de sensores adicionales, como sistemas de posicionamiento global (GPS) o sensores de odometría visual, el SLAM puede mejorar significativamente la precisión de la estimación de la posición y orientación del robot.

En resumen, la odometría es un componente crucial en la navegación de robots móviles diferenciales, ya que proporciona una estimación inicial de la posición y orientación del robot. Sin embargo, para garantizar una navegación precisa, es necesario abordar y corregir los errores de odometría mediante técnicas como el SLAM y la integración de datos de sensores adicionales. Esto permite al robot realizar tareas de manera más efectiva en una variedad de entornos y condiciones.

$$\begin{pmatrix} \Delta x \\ \Delta y \\ \varphi \end{pmatrix} = \begin{pmatrix} \cos(\theta_0) & -\sin(\theta_0) & 0 \\ \sin(\theta_0) & \cos(\theta_0) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\sin(\varphi)}{\varphi} & \frac{\cos(\varphi)-1}{\varphi} & 0 \\ \frac{1-\cos(\varphi)}{\varphi} & \frac{\sin(\varphi)}{\varphi} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Delta x_c \\ \Delta x_\perp \\ \varphi \end{pmatrix}$$

Figura 1. Matrices de posición

El cálculo del error en un robot móvil diferencial se refiere a la evaluación de las discrepancias entre la posición y orientación estimadas por la odometría y la posición y orientación reales del robot. Estas discrepancias pueden surgir debido a diversos factores, como el deslizamiento de las ruedas, irregularidades en la superficie del terreno o errores en los sensores utilizados para medir las rotaciones de las ruedas.

El proceso de cálculo del error implica comparar la posición y orientación estimadas por la odometría con mediciones precisas de la posición y orientación reales del robot. Se pueden utilizar técnicas como la localización simultánea y el mapeo para corregir los errores de odometría y mejorar la precisión de la estimación de la posición y orientación del robot.

Solución del problema:

- Código para la odometría implementado

```python
class Odometry(Node):
    #Constructur method for node class
    def __init__(self):

        #Parametros velocidad
        self.thp = 0.0
        self.xp = 0.0
        self.yp = 0.0
        self.tp = 0.0

        #Parametros posicion
        self.th = 0.0
        self.x = 0.0
        self.y = 0.0

        #Parámetros físicos
        self.r = 0.05
        self.l = 0.19

        #Velocidades angulares de encoders
        self.velR = 0
        self.velL = 0

        #definition of topic
        super().__init__('Azul_Odometry')

        #create subscriber for topic Azul_Velocity
        self.subR = self.create_subscription(Float32, 'VelocityEncR', self.listener_callbackR, rclpy.qos.qos_profile_sensor_data)
        self.subL = self.create_subscription(Float32, 'VelocityEncL', self.listener_callbackL, rclpy.qos.qos_profile_sensor_data)

        #Print info to confirm node was made.
        self.get_logger().info('Velocity node successfully initialized!!!')

        #Definition for timer period
        timer_period = 0.01
        self.tp = timer_period

        #Timer para operaciones
        self.timer = self.create_timer(timer_period, self.timer_callback)

        #create publisher for topic Azul_Velocity
        self.publisher = self.create_publisher(Pose2D, 'odom', 10)
```

Figura 2. Nodo para la odometría

En este código se define el nodo para la medición de odometría implementado. En primer lugar se declaran las variables a utilizar. Posteriormente se definen los suscriptores a las velocidades de los encoders de los motores.

```python
class Odometry(Node):
    #method timer calback for node Velocity R
    def listener_callbackR(self, msg):

        #Converts encoder input to linear and angular velocity
        self.velR = msg.data

    #method timer calback for node Velocity L
    def listener_callbackL(self, msg):

        #Converts encoder input to linear and angular velocity
        self.velL = msg.data

    #method timer callback
    def timer_callback(self):

        #Fórmulas para obtener las velocidades
        self.thp = self.r * ((self.velR - self.velL)/self.l)
        self.th = self.th + (self.thp * self.tp)

        self.xp = self.r * ((self.velR + self.velL)/2) * np.cos(self.th)
        self.x = self.x + (self.xp * self.tp)

        self.yp = self.r * ((self.velR + self.velL)/2) * np.sin(self.th)
        self.y = self.y + (self.yp * self.tp)

        pose_msg = Pose2D()
        pose_msg.x = self.x * 1.1
        pose_msg.y = self.y * 1.1
        pose_msg.theta = self.th
        self.publisher.publish(pose_msg)

#Main Fnc
def main(args=None):
    #Inicialiation for rclpy
    rclpy.init(args=args)
    #create node
    m_p = Odometry()
    #Spin method for publisher calback
    rclpy.spin(m_p)
    #Destoy node
    m_p.destroy_node()
```

Figura 3. Timer_callback

En esta parte del código se define el método timer_callback para leer las velocidades de los encoders, integrarlas y calcular la posición del robot en tiempo real; después las componentes de la posición en X, Y, θ se publica en el tópico “odom”.

- Código para el cálculo del error implementado

```
9   #Definición de la clase
10  class Controller(Node):
11      #Constructor
12      def __init__(self):
13
14          #Definición del tópico
15          super().__init__('Azul_Controller_node')
16
17          #Variables para lectura de topico /odom
18          self.x = 0.0
19          self.y = 0.0
20          self.w = 0.0
21
22          #Punto objetivo
23
24          self.xp = 1.0
25          self.yp = 1.0
26          self.wp = 0.785398
27          #self.wp = np.arctan2(self.yp,self.xp) #pi radianes
28
29          #Definición del periodo de tiempo
30          timer_period = 0.01
31
32          #Creación del tópico publicador: Azul Error
33          self.publisher = self.create_publisher(Pose2D, 'error_position', 10)
34
35          #Creación del tópico subcriptor: pose
36          self.subscriber = self.create_subscription(Pose2D, 'odom', self.pose_callback, 10)
37
38          #Timer para operaciones
39          self.timer = self.create_timer(timer_period, self.timer_callback)
40
41          #Confirmación de la creación del nodo
42          self.get_logger().info('Controller position error node successfully initialized!!!')
43
44          #Tipo de mensaje
45          self.msg = Float32()
46
47      def pose_callback(self, msg):
48          #Datos de pose recibidos
49          self.x = msg.x
50          self.y = msg.y
```

Figura 4. Definición del nodo para calcular el error

Este código se creó para medir constantemente el error de la posición real del robot con respecto a un punto específico en el espacio, esto se hace mediante una suscripción al tópico del nodo que publica la posición obtenida del robot, y calcula la diferencia entre las componentes X, Y, θ del punto fijo.

Primero se declaran las variables a utilizar que componen la posición del robot de las líneas 18 a 25. Después se definen el suscriptor a la posición del robot y el publicador del error del nodo. Por último, la función “pose_callback” lee la información obtenida del tópico de posición y guarda las componentes en sus respectivas variables.

```python
class Controller(Node):
    def pose_callback(self, msg):
        #Datos de pose recibidos
        self.x = msg.x
        self.y = msg.y
        self.w = msg.theta
        self.get_logger().info(f'Received pose: x={msg.x}, y={msg.y}, theta={msg.theta}')

    #Método del timer
    def timer_callback(self):
        cmd_vel = Twist()
        pose_msg = Pose2D()

        self.x = float(self.x)
        self.y = float(self.y)
        self.w = float(self.w)

        #Calculo de errores
        pose_msg.x = self.xp - self.x
        pose_msg.y = self.yp - self.y
        pose_msg.theta = self.wp - self.w

        #Publicar errores
        self.publisher.publish(pose_msg)

#Main Fnc
def main(args=None):
    #Inicialiation for rclpy
    rclpy.init(args=args)
    #create node
    m_p = Controller()
    #Spin method for publisher calback
    rclpy.spin(m_p)
    #Destoy node
    m_p.destroy_node()
    #rclpy shutdown
    rclpy.shutdown()

#main call method
if __name__ == 'main':
    main()
```

Figura 5. Función para calcular el error

La función “timer_callback” resta la posición del punto fijo con la del robot en tiempo real y publica esta diferencia o error en la posición en tiempo real, a modo que varía conforme el robot se mueve.

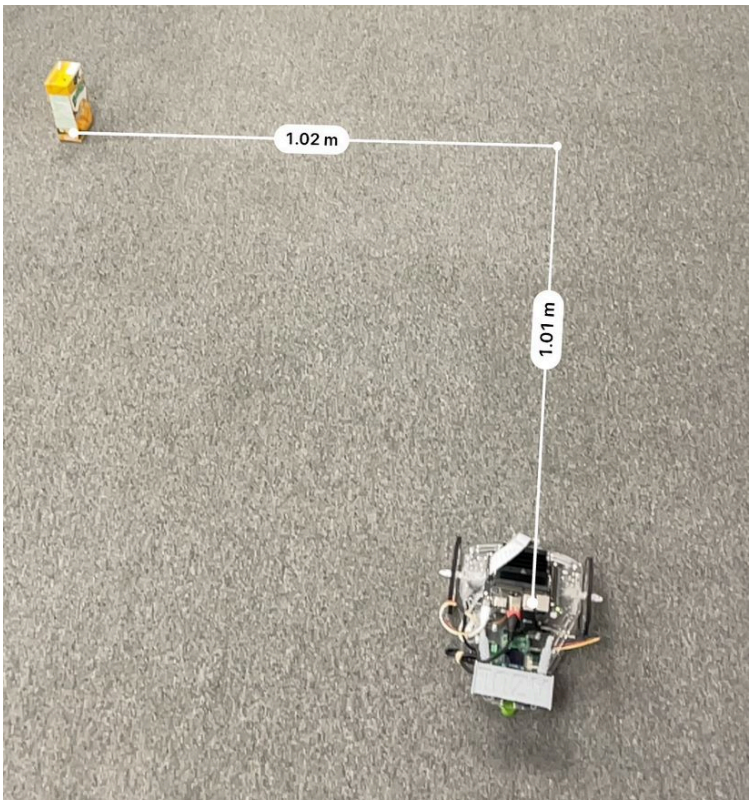

Figura 6. Marcas de referencia para distancias.

Resultados:

Los resultados del trabajo indican que el código fue exitoso en medir y publicar constantemente el error de la posición real del robot con respecto a un punto fijo en el espacio. La implementación efectiva se logró mediante la suscripción al tópico que publica la posición del robot y el cálculo constante de la diferencia entre las coordenadas X, Y y θ de la posición del robot y las del punto fijo.

1)Video de funcionamiento y resultados:

https://drive.google.com/file/d/1O4_HHYDM7zqVJr5sBMzX_yfvHBU3-G5n/view?usp=sharing

Este es el video demostrativo del funcionamiento en tiempo real del robot. En pantalla se puede observar, en primer lugar, dos terminales de la JetsonNano; la del lado izquierdo muestra la posición actual del robot actualizada en tiempo real, la del lado derecho muestra el error calculado del robot con respecto al punto de referencia fijado. Conforme avanza el video se puede observar cómo a medida que el robot se acerca al punto de referencia la posición mostrada en la terminal incrementa, y del mismo modo la diferencia del lado derecho disminuye.

Conclusiones:

Nuestro trabajo presenta resultados que están relacionados con los conceptos de odometría y cálculo de error en la navegación de robots. La capacidad del sistema para rastrear con precisión y en tiempo real los errores en la ubicación del robot se alinea con la función de la odometría, que consiste en estimar la posición y orientación del robot mediante la medición de las rotaciones de las ruedas y el ángulo de giro. El ajuste automático del curso del robot en función de estas discrepancias también refleja la capacidad del sistema para calcular y corregir errores en la navegación.

Además, el registro y análisis en tiempo real de estos errores representan una aplicación práctica del cálculo de error en la navegación de robots. Este análisis continuo permite identificar áreas de mejora en el algoritmo de navegación, lo que es fundamental para el desarrollo de sistemas robóticos más inteligentes y autónomos.

En conclusión, el sistema diseñado no solo demuestra la aplicación efectiva de los conceptos de odometría y cálculo de error en la navegación de robots, sino que también destaca la importancia de estas técnicas para mantener la precisión y la efectividad operativa en una variedad de aplicaciones robóticas.

Bibliografía o referencias:

1. *Pérez, J. & Martínez, L. (2010). Análisis del comportamiento robótico en entornos controlados. Revista Latinoamericana de Ingeniería, 16(3), 234-248.* https://www.redalyc.org/pdf/849/84916680034.pdf

2. *Mejoras de la localización odometrica de un robot diferencial mediante la corrección de errores sistemáticos.(2024). ResearchGate.* https://www.researchgate.net/publication/26611518_Mejoras_de_la_localizacion_odometrica_de_un_robot_diferencial_mediante_la_correccion_de_errores_sistematicos