Reto semanal 2. Manchester Robotics

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:
Para lograr el reto que se nos propuso esta semana, seguimos la siguiente metodología:

Primero, enriquecemos la familiarización que teníamos con el Puzzlebot conociendo más a profundidad sus componentes como motores, sensores, ruedas y unidad de control, y cómo se integran con ROS a través de nodos, tópicos y mensajes. Después, configuramos el entorno de desarrollo para trabajar con ROS y Puzzlebot, asegurándonos de establecer una buena conexión entre la computadora y el robot.

Luego, comenzamos con el control de lazo abierto básico, aprendiendo a enviar comandos de velocidad lineal y angular al Puzzlebot utilizando mensajes `Twist` (`geometry_msgs/Twist`). Después, trabajamos con la navegación multipunto con control de lazo abierto definiendo una serie de velocidades de navegación que el Puzzlebot debe seguir. Desarrollamos una secuencia de comandos de control para dirigir el robot a través de estas velocidades.

Finalmente, evaluamos el desempeño del Puzzlebot en la navegación multipunto, ajustando los parámetros de control para mejorar la trayectoria y el comportamiento del robot.

Objetivos:
Uno de los objetivos es incrementar la interacción y la implementación de funciones avanzadas disponibles en ROS, como la planificación de trayectorias y la navegación autónoma, además de explorar la integración de bibliotecas y herramientas de terceros para mejorar las capacidades del robot y su interacción con ROS. También se busca que el alumno practique la personalización de funciones de ROS para adaptarse a necesidades específicas de los casos de uso del robot.

Conocer la interconexión de los elementos que componen el Puzzlebot e interactuar físicamente con ellos es otro objetivo fundamental. Esto implica examinar cómo se comunican los diferentes componentes, como los sensores, actuadores y sistemas de control, a través de los mensajes y tópicos de ROS, y experimentar con distintas configuraciones de hardware y software para lograr una integración óptima y su interacción con ROS.

Otro de los objetivos es familiarizarse con los principios fundamentales del control de lazo abierto, incluidos los comandos de velocidad lineal y angular, y cómo afectan el movimiento del robot. Además, implementaremos algoritmos de control de lazo abierto en el Puzzlebot para ejecutar trayectorias simples, como movimientos en línea recta o giros específicos, y analizar posibles errores y limitaciones del control de lazo abierto, como la inestabilidad y la falta de corrección de la trayectoria.

Por último, el objetivo de interactuar con la respuesta generada por el robot al definir una velocidad o tiempo deseado para cumplir con una trayectoria definida a través de ROS es fundamental.

Introducción:
Para entender la teoría detrás de este reto hay que entender primero algunos conceptos que se utilizaran en la resolución del mismo.

- Mensajes Twist

- En ROS, los mensajes Twist (geometry_msgs/Twist) se utilizan para definir comandos de velocidad lineal y angular de un robot móvil. Un mensaje Twist contiene dos vectores de tres dimensiones: uno para la velocidad lineal y otro para la velocidad angular. Cada vector tiene componentes x, y, z, aunque, en general, para un robot móvil en un plano, solo se utilizan las componentes x de la velocidad lineal y z de la velocidad angular.
- Los mensajes Twist se envían a un nodo ROS que controla un robot móvil para definir su velocidad de traslación y rotación. Al publicar un mensaje Twist, se ordena al robot moverse con una determinada velocidad lineal hacia adelante o hacia atrás y una velocidad angular para girar a la izquierda o derecha.

- Cinemática de un Robot Diferencial

- Un robot diferencial es un tipo de robot móvil con dos ruedas motrices en paralelo y una rueda de apoyo. La cinemática de un robot diferencial describe cómo los movimientos de sus ruedas afectan su desplazamiento en el espacio.
- En un robot diferencial, la velocidad de las ruedas izquierda y derecha determina su movimiento. Si ambas ruedas se mueven a la misma velocidad, el robot avanza en línea recta. Si una rueda gira más rápido que la otra, el robot girará en la dirección de la rueda más lenta. La diferencia entre las velocidades de las ruedas se puede utilizar para controlar el giro y dirección del robot.

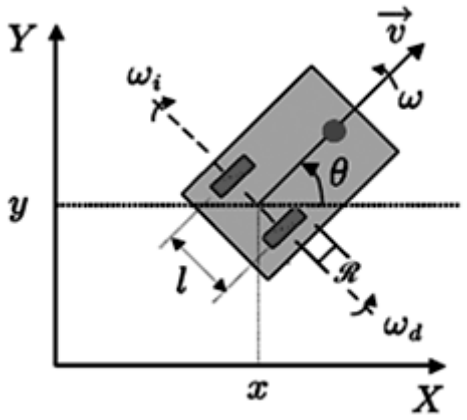


Fig.1. Modelo de un robot diferencial

- Lazo Abierto de Control en un Robot Móvil

- En un lazo abierto de control, se envían comandos de control a un robot sin recibir retroalimentación directa sobre el estado actual o la respuesta del robot.

- En un robot móvil, un sistema de control de lazo abierto significa enviar comandos de velocidad lineal y angular sin medir o corregir la trayectoria o posición actual del robot. Por ejemplo, al enviar un mensaje Twist para avanzar a una velocidad determinada, el robot seguirá esa velocidad hasta que se le dé un nuevo comando, sin realizar ajustes automáticos en respuesta a su entorno.

- Cálculo de la Distancia y Ángulo Recorrido por un Robot Móvil

- Distancia: Para calcular la distancia recorrida por un robot móvil, se puede usar la fórmula distancia = velocidad * tiempo. Al conocer la velocidad lineal del robot y el tiempo que ha estado en movimiento, se puede calcular la distancia total que ha recorrido.
- Ángulo: Para calcular el ángulo recorrido, se usa la fórmula ángulo = velocidad angular * tiempo. Conociendo la velocidad angular del robot y el tiempo durante el cual ha estado girando, se puede calcular el ángulo total que ha girado.

Estos cálculos pueden realizarse utilizando los datos de los sensores del robot, o controlando los comandos de velocidad enviados al robot en función del tiempo transcurrido. En sistemas más avanzados, como los de lazo cerrado, se utiliza retroalimentación de sensores para mejorar la precisión de estos cálculos y el control del robot.

Solución del problema:
Para solución del reto se generaron dos códigos de python uno llamado “path generator” y otro llamado “controller”. Path generator contiene las velocidades lineales y angulares para generar los paths que escogimos, este codigo tambien recibe un input con el tiempo en el que el robot debe de completar el path también seleccionando por el usuario con un input de 1 a 3, una vez con estos inputs, path generator hace los cálculos para la velocidad del robot y envía estas velocidades a controller por medio de un tópico llamado pose.

Controller, es el encargado de leer las velocidades del tópico pose y mandarlas a cvd_vel para que nuestro robot pueda ejecutar el path seleccionado.

```
#Declaración de arreglos de trayectorias
#Cuadrado
self.c = 0
self.pro = 0
self.x = [0.21, 0.0, 0.21, 0.0, 0.21, 0.0, 0.21, 0.0]
self.z = [0.0, .1379, 0.0, .1379, 0.0, .1379, 0.0, .1379]
#Circulo
self.x2 = [0.21, 0.21, 0.21, 0.21, 0.21, 0.21, 0.21, 0.21]
self.z2 = [0.21, 0.21, 0.21, 0.21, 0.21, 0.21, 0.21, 0.21]
#Zigzag
self.x3 = [0.21, 0.0, 0.21, 0.0, 0.21, 0.0, 0.21, 0.0]
self.z3 = [0.0, .034, 0.0, -.034, 0.0, .034, 0.0, -.034]
self.i = 0
```

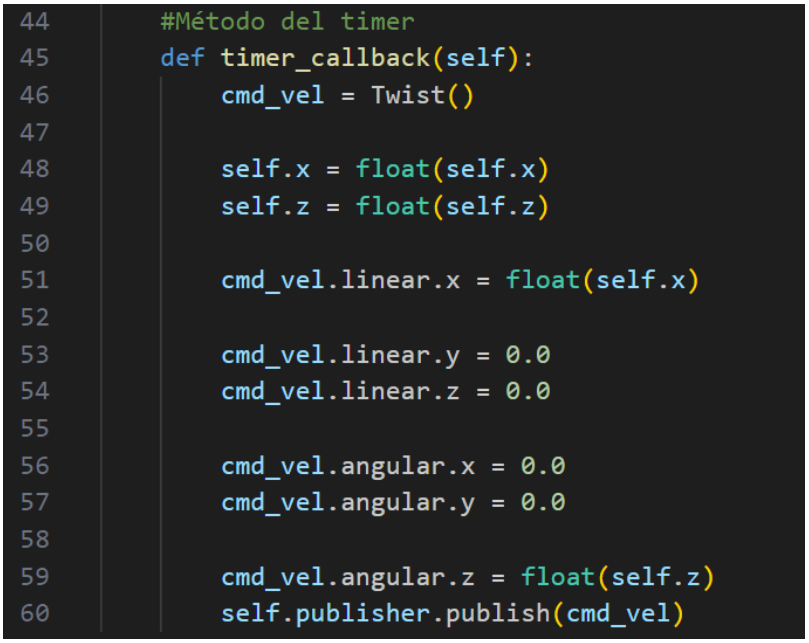
Fig.2. Arreglos que guardan las velocidades lineales y angulares en Path_generator.

```
    #Método del timer
    def timer_callback(self):
        self.c = self.get_parameter('type').get_parameter_value().integer_value
        pose_msg = Pose2D()
        if self.c == 1:
            if self.i < 8:
                pose_msg.x = self.x[self.i] * self.pro

                pose_msg.y = self.z[self.i] * self.pro

            else:
                pose_msg.x = 0.0

                pose_msg.y = 0.0
        elif self.c == 2:
            if self.i < 8:
                pose_msg.x = self.x2[self.i] * self.pro

                pose_msg.y = self.z2[self.i] * self.pro

            else:
                pose_msg.x = 0.0

                pose_msg.y = 0.0

        elif self.c == 3:
            if self.i < 8:
                pose_msg.x = self.x3[self.i] * self.pro

                pose_msg.y = self.z3[self.i] * self.pro

            else:
                pose_msg.x = 0.0

                pose_msg.y = 0.0

        self.i = self.i + 1
        if pose_msg.x >= 0.8 or pose_msg.y >= 0.8:
            self.get_logger().info('Las velocidades superan las capacidades del robot. Poco tiempo de ejecucion')
        else:
            self.publisher.publish(pose_msg)
```

Fig.3. Función que envía las velocidades a ejecutar al tópico pose dependiendo el caso en Path_generator.

```
    def pose_callback(self, msg):
        #Datos de pose recibidos
        self.x = float(msg.x.data)
        self.z = float(msg.y.data)
        self.get_logger().info(f'Received pose: x={msg.x}, y={msg.y}, theta={msg.theta}')
```

Fig.4. Función que recibe los datos en el tópico pose de controller.



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

        self.x = float(self.x)
        self.z = float(self.z)

        cmd_vel.linear.x = float(self.x)

        cmd_vel.linear.y = 0.0
        cmd_vel.linear.z = 0.0

        cmd_vel.angular.x = 0.0
        cmd_vel.angular.y = 0.0

        cmd_vel.angular.z = float(self.z)
        self.publisher.publish(cmd_vel)
```

Fig.5. Función que envía las velocidades a cmd_vel para que el robot las ejecute.

Resultados:
Dentro de los resultados obtenidos en el desarrollo del reto semanal encontramos tanto buenas como malas respuestas, en cuanto a los errores se tendrá que trabajar más adelante. para volver nuestro código más robusto y eficaz.

1)Video de funcionamiento y resultados:

https://drive.google.com/drive/folders/1599xu67r-MDdgGYgXESITAbsb735xr0n

Conclusiones:
Con la realización de este reto tenemos las siguientes conclusiones, tomando en cuenta que a pesar de los inconvenientes que se los presentaron, pudimos llegar a los resultados:

Logramos adquirir un conocimiento más sólido de los componentes del Puzzlebot y su integración con ROS. Esto incluye la comprensión de la comunicación entre los elementos del robot a través de nodos, tópicos y mensajes. También desarrollamos habilidades prácticas para enviar comandos de velocidad lineal y angular al Puzzlebot mediante mensajes Twist. La capacidad de controlar el robot de forma efectiva en la práctica demuestra la comprensión de los principios del control de lazo abierto. Logramos que el Puzzlebot siga trayectorias definidas a través de velocidades de navegación preestablecidas, aplicando control de lazo abierto. El robot ha demostrado ser capaz de cumplir con las trayectorias deseadas, siguiendo los comandos de velocidad y tiempo definidos. Mediante las pruebas evaluamos el desempeño del Puzzlebot en la navegación multipunto, ajustando los parámetros de control según sea necesario para mejorar la precisión y consistencia de las trayectorias. Esta capacidad de ajuste es un indicador de la comprensión de los principios del control de lazo abierto.

Bibliografía o referencias:

1. *PuzzleBot Jetson Edition – Manchester Robotics*. (s. f.).

   https://manchester-robotics.com/puzzlebot/puzzlebot-jetson-edition/

2. *"geometry_msgs/Twist" - ROS Wiki*. (s. f.).

   https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html

3. *Modelo cinemático y simulación con Python: Robot móvil diferencial.*

   Roboticoss.com. (s.f.).

   https://roboticoss.com/modelo-cinematico-y-simulacion-con-python-robot-movil-diferencial/

4. *"Descubre los principios fundamentales de control en robótica.". roborevolucion.com. (s.f).* https://roborevolucion.com/descubre-los-principios-fundamentales-de-control-en-robotica/