Tecnológico de Monterrey

Instituto Tecnológico de Estudios Superiores Monterrey

CAMPUS PUEBLA

Integración de Robótica y Sistemas Inteligentes TE3003B

Mini challenge II

Integrantes

José Jezarel Sánchez Mijares A01735226

Antonio Silva Martínez A01173663

Dana Marian Rivera Oropeza A00830027

Profesor

Rigoberto Cerino Jiménez

Fecha: 24 de abril de 2024

Código

Declaramos las librerías y generamos el nodo al igual que el tópico en el que vamos a publicar los mensajes con las velocidades correspondientes.

```python
import rospy
from geometry_msgs.msg import Twist
import numpy as np

class SquareTrajectory:
    def __init__(self):
        rospy.init_node('square_trajectory_node')
        self.cmd_vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        self.rate = rospy.Rate(10)  # 10 Hz
```

Generamos la función move donde tomando como entrada las velocidades lineales, angulares y la distancia genera el cálculo del tiempo de desplazamiento considerando las condiciones dadas para cada punto de la trayectoria, una vez terminado el tiempo para el desplazamiento se envía un mensaje para detener el desplazamiento del robot, seguido a esto se envía un mensaje para rotar la dirección del robot y que inicie su recorrido hacia la siguiente posición.

Una vez que se detiene el movimiento, se llama a esta función para calcular el tiempo de rotación y dejar que el robot cambie de posición para así iniciar su movimiento hacia el siguiente punto declarado.

```python
def move_ang(self, angular_speed):
    twist = Twist()
    angular_distance = np.pi / 4

    time_to_rotate = angular_distance / angular_speed

    # Crear mensaje Twist para rotar
    twist.angular.z = angular_speed

    # Publicar el mensaje para rotar
    start_time = rospy.Time.now()
    while (rospy.Time.now() - start_time).to_sec() < time_to_rotate:
        self.cmd_vel_pub.publish(twist)
        self.rate.sleep()

    twist.angular.z = 0.0
    self.cmd_vel_pub.publish(twist)
    self.rate.sleep()
```

Finalmente en este segmento de código realizamos la declaración de las velocidades lineales, angulares y de la distancia para cada uno de los puntos dentro de las rutinas, es decir, los puntos necesarios para generar la forma del cuadrado y de zigzag que alimentan a las funciones anteriores.

```python
def run(self):
    # Mover al punto (1,0)
    self.move(linear_speed=0.2, angular_speed=0.2, distance=1)

    # Mover al punto (1,1) ahaha
    self.move(linear_speed=0.2, angular_speed=0.2, distance=1)

    ## Mover al punto (0,1)
    self.move(linear_speed=0.2, angular_speed=0.2, distance=1)

    ## Mover al punto (0,0)
    self.move(linear_speed=0.2, angular_speed=0.2, distance=1)
    #
    ###ZIGZAG o diamante
    self.move_ang(angular_speed=0.2)
    self.move(linear_speed=0.2, angular_speed=0.2, distance=1)

    self.move_ang(angular_speed=0.2)
    self.move(linear_speed=0.2, angular_speed=0.2, distance=1)

    self.move_ang(angular_speed=0.2)
    self.move(linear_speed=0.2, angular_speed=0.2, distance=1)
    self.move(linear_speed=0.2, angular_speed=0.0, distance=1)
```

Resultados

En la siguiente liga se podrán visualizar los resultados de la simulación, mostrando al turtlebot realizando ambos recorridos:

https://youtu.be/AWN3HJxUdrY?feature=shared