Reto semanal 3. 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:
El siguiente proyecto involucra la interacción con el Puzzlebot, un robot móvil diferencial, con el objetivo de desarrollar habilidades de control. Este reto nos dio la oportunidad de explorar aspectos clave como la localización mediante odometría, el control en lazo cerrado y la navegación punto a punto, todo ello dentro del entorno de desarrollo ROS.

El Puzzlebot será utilizado como plataforma para aplicar conceptos teóricos en un contexto práctico y dinámico. Controlaremos en todo momento las coordenadas de la posición del robot, así como daremos una posición deseada. Un aspecto crucial de este desafío es la capacidad de calcular en tiempo real el error de posición y, posteriormente, diseñar y aplicar un controlador proporcional (P), proporcional-integral (PI) o proporcional-integral-derivativo (PID) para minimizar dicho error.

La integración de estos conceptos previos junto con la implementación de un controlador permitirá comprender los fundamentos teóricos del control de robots y también adquirir habilidades prácticas para optimizar el desempeño del Puzzlebot en la navegación hacia una posición deseada.

Objetivos:
Comprender los conceptos fundamentales de la localización mediante odometría y su aplicación en la determinación de la posición del Puzzlebot en tiempo real. Aplicar técnicas de control en lazo cerrado para minimizar el error de posición del Puzzlebot y lograr una navegación precisa hacia una posición deseada. Diseñar y desarrollar controladores proporcionales (P), proporcionales-integrales (PI) o proporcionales-integrales-derivativos (PID) para mejorar la precisión y estabilidad en la navegación del Puzzlebot. Implementar la navegación punto a punto, permitiendo al Puzzlebot desplazarse de manera autónoma hacia una posición deseada especificada por el usuario.

Introducción:
El control en lazo cerrado para un robot móvil es un método utilizado para mantener la posición o el comportamiento del robot dentro de un rango deseado mediante la retroalimentación continua del estado del sistema. Funciona comparando la posición o el estado actual del robot con el valor deseado y utilizando esta diferencia para ajustar el control del robot en tiempo real. En el caso de un robot móvil, esto puede implicar ajustar las velocidades de las ruedas o las acciones de control para corregir cualquier desviación de la trayectoria deseada.

El controlador PID aplicado a un robot móvil diferencial es un tipo específico de control en lazo cerrado que utiliza tres componentes para calcular la señal de control: proporcional, integral y derivativo.

- El término proporcional produce una salida proporcional a la diferencia entre la posición deseada y la posición actual del robot, lo que ayuda a corregir el error de manera proporcional a su magnitud.

- El término integral considera la acumulación del error a lo largo del tiempo y ajusta la salida del controlador en función de esta integral del error.
- El término derivativo anticipa las tendencias futuras del error calculando su tasa de cambio y ajustando la salida del controlador en consecuencia.

El controlador PID se utiliza comúnmente en aplicaciones de control de robots móviles debido a su capacidad para proporcionar un rendimiento robusto y estable en una amplia variedad de condiciones y entornos.

El cálculo del error en el contexto del control en lazo cerrado y el controlador PID se refiere a la diferencia entre la posición deseada del robot y su posición actual. Este error se utiliza como entrada para el controlador, que luego genera una señal de control para corregir el error y llevar al robot a la posición deseada.

La robustez de un controlador se refiere a su capacidad para mantener un rendimiento estable y preciso en presencia de perturbaciones, incertidumbres y variaciones en las condiciones del sistema. Un controlador robusto es capaz de mantener un control efectivo del robot móvil incluso cuando se enfrenta a condiciones adversas como cambios en la carga, fricción variable o errores en los sensores. La robustez se logra mediante un diseño cuidadoso del controlador y la incorporación de técnicas de sintonización y compensación que minimizan los efectos de las perturbaciones y garantizan un rendimiento confiable en una variedad de situaciones.

Solución del problema:

La solución del problema se divide en el funcionamiento de 4 nodos estos son robot_node, odometry_node, controller, point_generator_node. al nodo de robot node se le llama al el funcionamiento del robot en el cual nosotros leemos los encoders desde el nodo para la odometría como se muestra a continuación.

Odometry_node:

```
1 import rclpy
2 from rclpy.node import Node
3 from std_msgs.msg import Float32
4 from geometry_msgs.msg import Pose2D
5 import numpy as np
```

Figura 1: importación de las dependencias necesitadas.

```python
#Definition of node class Velocity
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')

```

Figura 2: inicializacion de la clase Odometry y atributos de la misma.

```python
        #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('Odometry 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)

        #initialize msg data type
        self.msg = Float32()

    #method timer calback for node Velocity R
    def listener_callbackR(self, msg):

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

```

Figura 3: Inicialización de suscriptores y publicadores usados.

```
55      #method timer calback for node Velocity R
56      def listener_callbackR(self, msg):
57
58          #Converts encoder input to linear and angular velocity
59          self.velR = msg.data
60
61      #method timer calback for node Velocity L
62      def listener_callbackL(self, msg):
63
64          #Converts encoder input to linear and angular velocity
65          self.velL = msg.data
66
```

Figura 4: Definición de suscriptores a EncL, EncR.

```
67      #method timer callback
68      def timer_callback(self):
69
70          #Fórmulas para obtener las velocidades
71          self.thp = self.r * ((self.velR - self.velL)/self.l)
72          self.th = self.th + (self.thp * self.tp)
73
74          self.xp = self.r * ((self.velR + self.velL)/2) * np.cos(self.th)
75          self.x = self.x + (self.xp * self.tp)
76
77          self.yp = self.r * ((self.velR + self.velL)/2) * np.sin(self.th)
78          self.y = self.y + (self.yp * self.tp)
79
80          pose_msg = Pose2D()
81          pose_msg.x = self.x * 1.1
82          pose_msg.y = self.y * 1.1
83          pose_msg.theta = self.th
84          self.publisher.publish(pose_msg)
85
```

Figura 5: definición del publicador de posición del robot a tópico /odom.

```
86  #Main Fnc
87  def main(args=None):
88      #Inicialiation for rclpy
89      rclpy.init(args=args)
90      #create node
91      m_p = Odometry()
92      #Spin method for publisher calback
93      rclpy.spin(m_p)
94      #Destoy node
95      m_p.destroy_node()
96      #rclpy shutdown
97      rclpy.shutdown()
98
99  #main call method
100 if __name__ == 'main':
101     main()
```

Figura 6: Método main para la ejecución del nodo.

Los tramos importantes del nodo anterior son las suscripciones a los tópicos para la lectura de los encoders del robot, así como el cálculo de la posición del robot por medio de la integración; por métodos numéricos, de las velocidades obtenidas con los encoders. El punto anterior es visible en la línea 70 a 84.

Poinnt_generator_node:

```
1 import rclpy
2 from rclpy.node import Node
3 from std_msgs.msg import Float32
4 from std_msgs.msg import Int64
5 from geometry_msgs.msg import Pose2D
6 import numpy as np
7
```

Figura 7: importacion de dependencia para el nodo point generator.

```
8 #Definition of node class Velocity
9 class Point_generator(Node):
10
11     #Constructur method for node class
12     def __init__(self):
13
14         #definition of topic
15         super().__init__('Point_Generator')
16
17         #Parametro del numero de puntos
18         self.declare_parameter('Points',3)
19
20         #variables a utilizar
21         self.num_points = (self.get_parameter('Points').get_parameter_value().integer_value)
22         self.x = []
23         self.y = []
24         self.i = 0
25         self.s = 0
26         self.sa = 0
27
```

Figura 8: inicialización de la clase point generator y de atributos de la misma.

```
27
28         #Timer para operaciones
29         timer_period = 0.5
30         self.timer = self.create_timer(timer_period, self.timer_callback)
31
32         #Print info to confirm node was made.
33         self.get_logger().info('Point_generator node successfully initialized!!!')
34
35         #Creación del tópico subcriptor: contador
36         self.subscriber = self.create_subscription(Int64, 'contador', self.contador_callback,
37
38         #create publisher for topic Point
39         self.publisher = self.create_publisher(Pose2D, 'Point', 10)
40
41         #initialize msg data type
42         self.msg = Pose2D()
43
```

Figura 9: inicialización de subscriptor y publicador para el nodo point_generator_node.

```
    def contador_callback(self, msg):
        #Dato recibido
        self.s = msg.data

    def timer_callback(self):
        pose_msg = Pose2D()
        #calculamos los aungulos para cada uno de los puntos utilizando linspace
        angles = np.linspace(0,2*np.pi,self.num_points, endpoint=False)
        #guardamos la coordenadas de "x" y "y" en arreglos
        self.x = list(0.5 * np.cos(angles))
        self.y = list(0.5 * np.sin(angles))

        if(self.s != self.sa):
            self.i = self.i + 1

        if(self.i > (self.num_points-1)):
            #sumamos los arreglos y los guardamos en msg
            pose_msg.x = self.x[self.num_points-1]
            pose_msg.y = self.y[self.num_points-1]
            pose_msg.theta = 0.0
        else:
            #sumamos los arreglos y los guardamos en msg
            pose_msg.x = self.x[self.i]
            pose_msg.y = self.y[self.i]
            pose_msg.theta = 0.0
        #mandamos msg a Point
        self.publisher.publish(pose_msg)
        self.get_logger().info('Punto mandado')

```

Figura 10: descripción de suscriptor y publicador de los puntos generados.

```
#Main Fnc
def main(args=None):
    #Inicialiation for rclpy
    rclpy.init(args=args)
    #create node
    m_p = Point_generator()
    #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 11: método main para la ejecución del nodo.

Dentro del nodo para la generación de puntos lo que se buscar hacer el que se manden múltiples puntos a un arreglo los cuales se mandan uno a uno dentro de un tópico /Point, esto se va haciendo mientras se escucha al tópico /contador con el fin de recorrer el arreglo de puntos y cambiar el punto que se está publicando.

Controller_node:

```python
#Definición de la clase
class Controller(Node):
    #Constructor
    def __init__(self):

        #Definición del tópico
        super().__init__('Azul_Controller_node')

        #Variables para lectura de topico /odom
        self.x = 0.0
        self.y = 0.0
        self.w = 0.0

        #Punto objetivo
        self.xp = 0.0
        self.yp = 0.0
        self.wp = 0.0

        #Variables para lectura de topico
        self.i = 0

        #Creación del tópico publicador: cmd_vel
        self.publisher = self.create_publisher(Twist, 'cmd_vel', 10)
        self.publisher2 = self.create_publisher(Int64, 'contador', 10)

        #Creación del tópico subcriptor: odom
        self.subscriber = self.create_subscription(Pose2D, 'odom', self.pose_callback, 10)

        #Creación del tópico subcriptor: point
        self.subscriberP = self.create_subscription(Pose2D, 'Point', self.point_callback, 10)

        #Definición del periodo de tiempo
        timer_period = 0.01

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

        #Confirmación de la creación del nodo
        self.get_logger().info('Controller position error node successfully initialized!!!')

        #Tipo de mensaje
        self.msg = Float32()

        #Parametros de los controladores PI
        self.kp_ang = 0.5
        self.ki_ang = 0.0
        self.kp_lin = 0.5
        self.ki_lin = 0.0

        #Variables de error integral
        self.error_integral_ang = 0.0
        self.error_integral_lin = 0.0
```

Figura 12. Definición de la clase controller

En esta parte del código se inicializan las variables locales con las que se leen la posición del robot y el punto objetivo. También se crean los suscriptores a los tópicos donde se publican dichas posiciones y se crea el publicador a “cmd_vel” para controlar por medio de un PI las velocidades lineales y angulares.

```python
    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}')

    def point_callback(self, msg):
        #Datos de objetivo recibidos
        self.xp = msg.x
        self.yp = msg.y

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

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

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

        #angle calculation
        self.wp = np.arctan2((self.yp - self.y), (self.xp - self.x))

        #Errores de posicion y orientación
        error_w = self.wp - self.w

        #Error for v
        error_v = np.sqrt(((self.yp - self.y)**2) + ((self.xp - self.x)**2))

        #Controlaador PI para velocidad angular
        self.error_integral_ang += error_w
        ang_control_output = self.kp_ang * error_w + self.ki_ang * self.error_integral_ang

        #Controlador PI para velocidad lineal
        self.error_integral_lin += error_v
        lin_control_output = self.kp_lin * error_v + self.ki_lin * self.error_integral_lin

        #Limitar vel
        ang_control_output = np.clip(ang_control_output, 0.1, 0.3)
        lin_control_output = np.clip(lin_control_output, 0.1, 0.3)

        if (ang_control_output > 0.3):
                ang_control_output = 0.3

        if (lin_control_output > 0.3):
                lin_control_output = 0.3

        #Mandar vel
        self.get_logger().info(f'V: {lin_control_output}')
        self.get_logger().info(f'W: {ang_control_output}')
        cmd_vel.angular.z = ang_control_output
        cmd_vel.linear.x = lin_control_output

        self.publisher.publish(cmd_vel)
```

Figura 13. Métodos de la clase

En estas secciones del código se leen de los tópicos las posiciones reales y objetivo del robot; también se calcula el error angular y diferencial del robot y se aplica el control proporcional e integral para la velocidad del robot al llegar al punto objetivo.

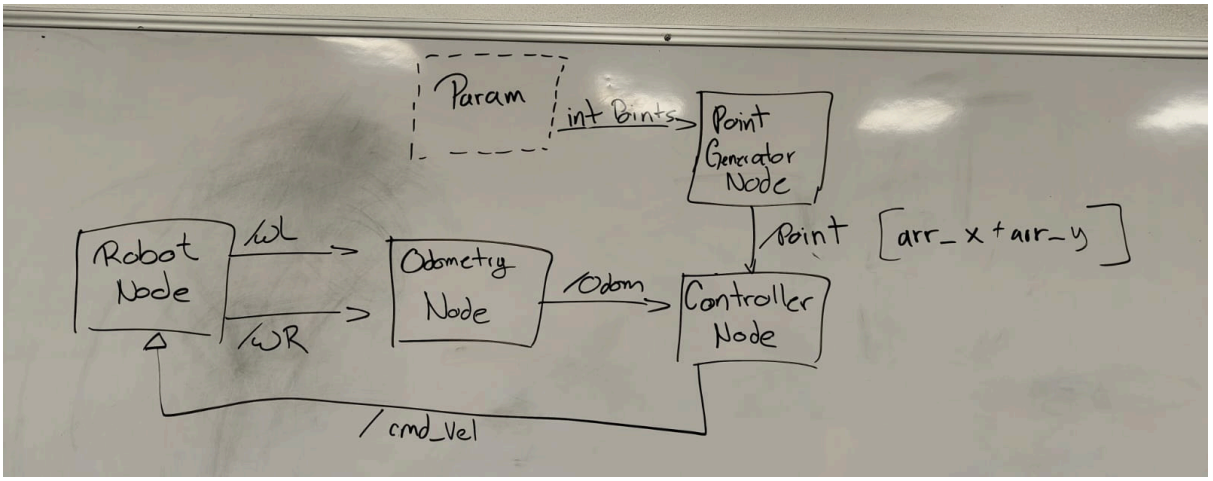

Figura 14. Diagrama de la implementación

Este es el diagrama que representa cómo se emplean todos los nodos, los nombres de los tópicos con los que se conectan y el lazo cerrado que se implementó.

Conclusiones:

El desafío de interactuar con el Puzzlebot nos permitió explorar y aplicar conceptos clave en el control de robots móviles diferenciales, incluyendo la localización mediante odometría, el control en lazo cerrado y la navegación punto a punto. La experiencia práctica con el Puzzlebot en el entorno de desarrollo ROS proporcionó una oportunidad para vincular los conceptos teóricos con aplicaciones reales y dinámicas en el campo de la robótica. La capacidad de controlar en todo momento las coordenadas de la posición del robot y establecer una posición deseada demostró la efectividad del enfoque de control en lazo cerrado para mantener la precisión en la navegación del robot.

La implementación y ajuste de controladores proporcionales (P), proporcionales-integrales (PI) o proporcionales-integrales-derivativos (PID) nos permitió comprender y aplicar distintas estrategias de control para minimizar el error de posición del Puzzlebot. La integración exitosa de los conceptos teóricos con la práctica en la implementación de controladores destacó la importancia de adquirir habilidades prácticas para optimizar el desempeño de los robots móviles en entornos del mundo real.

El reto no sólo facilitó la comprensión de los fundamentos teóricos del control de robots, sino que también nos permitió desarrollar habilidades prácticas para mejorar la navegación y el desempeño del Puzzlebot utilizando técnicas de control avanzadas.

Bibliografía o referencias:

1. *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/

2. *"Descubre los principios fundamentales de control en robótica.". roborevolucion.com. (s.f).* https://roborevolucion.com/descubre-los-principios-fundamentales-de-control-en-robotica/
3. *CONTROL ROBUSTO DE UN SISTEMA MECÁNICO SIMPLE MEDIANTE UNA HERRAMIENTA GRÁFICA. revistas.unal.edu.co. (s.f).* https://revistas.unal.edu.co/index.php/dyna/article/view/15852/36191