RETO SEMANAL 3. Manchester Robotics

Abraham Ortiz Castro
Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
A01736196@tec.mx

Alan Iván Flores Juárez
Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
a01736001@tec.mx

Jesús Alejandro Gómez Bautista
Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
A01736171@tec.mx

Ulises Hernández Hernández
Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
A01735823@tec.mx

I. Resumen

Este documento es una continuación de nuestro trabajo de la semana pasada, donde nos enfocamos en la definición de trayectorias para el PuzzleBot. En esta ocasión, el objetivo es que el robot pueda seguir una serie de trayectorias geométricas de manera continua. Es decir, una vez que el PuzzleBot complete una trayectoria, inmediatamente comenzará con la siguiente, hasta que se complete toda la lista de trayectorias.

Para lograr esto, nos apoyaremos en la odometría del robot, que nos proporciona información valiosa sobre su posición y orientación. Además, implementaremos un control de lazo cerrado, que nos permitirá ajustar las velocidades de las ruedas del robot en tiempo real para seguir las trayectorias de manera precisa.

El diseño y ajuste de nuestro controlador PID (Proporcional, Integral, Derivativo) será fundamental en este proceso. Este controlador nos permitirá ajustar las velocidades de las ruedas en función del error de control y sus derivadas e integrales, lo que nos permitirá seguir las trayectorias de manera precisa.

La combinación de estos nuevos conocimientos con los que ya hemos adquirido nos permitirá construir un sistema robusto para el diseño del movimiento del robot. Con este sistema, podremos hacer que el PuzzleBot siga una serie de trayectorias geométricas de manera precisa y eficiente.

II. Objetivos.

- Mejorar la interacción entre PuzzleBot y las herramientas provistas por ROS para crear comandos sólidos aplicados a tareas específicas.
- Conocer las coordenadas de la posición del robot mediante el uso de la odometría.
- Calcular el error de la posición en tiempo real para ajustar la locomoción del robot.
- Diseñar un controlador P, PI o PID para ajustar el movimiento del robot con base a el error en la posición.
- Implementar algoritmos de navegación de punto a punto unificados a un control de lazo cerrado.

III. Introducción.

A. Control de lazo cerrado para un robot móvil

El control de lazo cerrado o de retroalimentación, es un tipo de control en el que se utiliza la salida del sistema para ajustar la entrada con el objetivo de mantener un sistema en un estado deseado.

Para un robot móvil diferencial, el control de lazo cerrado implica el uso de la retroalimentación para ajustar las velocidades de las ruedas del robot con el fin de mantenerlo en una trayectoria esperada o deseada. Este proceso implica lo siguiente:

- **Modelación del sistema:** Implica entender el cómo las velocidades de las ruedas del robot afectan el movimiento, a partir de las ecuaciones de cinemática que describen al robot.
- **Medición del estado:** Esto puede ser logrado a partir del uso de sensores como el encoder en las ruedas, los cuales nos permiten medir la rotación de las ruedas y con algunos datos más propios del robot diferencial, calcular la distancia recorrida del robot.
- **Cálculo del error:** Conociendo el estado actual del robot, se puede conocer el error de la posición y del ángulo al conocer la diferencia entre esos valores deseados y los valores reales o actuales que conocemos nos dará el error existente para poder lograrlo.
- **Ajuste de las velocidades de las ruedas:** A partir del error, ajustamos las velocidades de las ruedas del robot, para nuestros fines a través de un controlador PID, lo cual permitirá ajustar la velocidad de las ruedas en función del error

[1]

B. PID aplicado a un robot móvil diferencial

Un controlador PID (Proporcional Integrador Derivativo) es una técnica de control empleada para sistemas de lazo cerrado. Enfocado en un robot móvil diferencial, el PID tendrá la tarea de ajustar las velocidades de las ruedas del robot en función de la medición del error. Cada una de las siglas de PID describe

el tipo de control que ejercerán sobre el sistema, estas acciones son:

- **Control Proporcional (P):** Basándose en el error actual del control, si el error medible es grande, el controlador de tipo P intentará corregirlo rápidamente al aumentar la velocidad de las ruedas al producir una salida proporcional al error actual. La constante de proporcionalidad determina cuánto cambio en la salida se produce por unidad de cambio del error.
- **Control Integral (I):** Este elemento del control se enfoca en los errores pasados, es decir, si el robot ha estado fuera de la trayectoria por un tiempo, el controlador de tipo I comenzará a acumular este error con el tiempo y lo solucionará al ajustar la velocidad de las ruedas, proporcionando así un control de acción lenta, pero contante, ayudando a eliminar el error residual que a menudo ocurre al usar el controlador de tipo P.
- **Control Derivativo (D):** Se basa en la tasa de cambio del error, en otras palabras, si el robot vuelve rápidamente a su trayectoria, el controlador de tipo D disminuirá la velocidad con el fin de evitar sobre ajustes. Este control proporciona una acción rápida y proporcional a la tasa de cambio del error, disminuyendo así la respuesta del sistema y evitando oscilaciones.

Si bien es cierto que el controlador PID puede ser muy efectivo dentro de un robot diferencial móvil, necesita de un ajuste cuidadoso de los parámetros P, I y D para un mejor rendimiento. Aunque los valores se pueden obtener mediante métodos heurísticos, podemos recurrir a técnicas más efectivas como el método Ziegler-Nichols.

Adicionalmente, a lo anterior, no debemos dejar de lado que el control PID asume un modelo lineal del sistema, por lo que ciertos comportamientos del robot móvil de tipo diferencial puede mostrar comportamientos no lineales como consecuencia del deslizamiento de las ruedas o la dinámica del robot. [2]

C. Cálculo del error

Para generar un sistema robusto de navegación y control del robot, el cálculo del error en un robot móvil de tipo diferencial es fundamental, y se basa fundamentalmente en la diferencia entre la posición actual del robot y la posición deseada u objetivo.

El error en un robot diferencial puede abordarse en dos componentes primordiales.

- **Error de posición:**

$$e_p = P_o - P_a \tag{1}$$

  Es la diferencia entre la posición deseada y su posición actual, básicamente, es la diferencia entre las coordenadas de la posición objetivo y las actuales del robot.
- **Error de orientación:**

$$e_\theta = \theta_o - \theta_a \tag{2}$$

  Es la diferencia entre la orientación deseada y su orientación actual, básicamente, es la diferencia entre el ángulo objetivo y actual del robot.

[3]

D. Robustez de un controlador

Se entiende a la robustez del controlador a la capacidad de mantener un rendimiento de forma adecuada pese a las variaciones o incertidumbres en el sistema donde se esta ejerciendo el control, se puede entender por ende que un controlador robuste es aquel capaz de manejar variaciones en los parámetros del sistmea con el fin de mantener los objetivos, su estabilidad y el rendimiento.

Particularmente en un sistema PID, la robustez hace referencia a la capacidad del controlador para manejar variaciones sin que su rendimiento se reduzca, es decir, pese a que el sistema experimeinte cambios en sus parámetros, un controlador robusto es capas de adpatarse al cambio mientras continua dando un control apropiado o mejor dicho, un control que menos sensible a cambios bruzcos del sistema. [4]

IV. Solución del problema

A. Código y lógica

Para dar solución a este proyecto se realizó la creación de tres nodos `odometry` el cual es el encargado de recibir los valores de la velocidad angular de cada una de las llantas y convertir dichos datos a desplazamiento en 'x', 'y' y theta, valores que posteriormente son enviados por un tópico del mismo nombre. El nodo `path_generator` el cuál es el nodo encargado de enviar las coordenadas correspondientes para cada una de las figuras, desde un triángulo hasta un hexágono, cuyas coordendadas se encuentran dentro de un círculo con diámetro 1, estas coordendas son enviadas de igual forma por otro tópico con el mismo nombre. Por último el nodo `controller`, recibe los valores del tópico `odometria` y del `path_generator` tomando esta información, calculando la posición actual, la posición objetivo, el ángulo actual y el ángulo objetivo. Para calcular el error como se muestra en las ecuacuiones 1 y 2. Segmentos de código implementado en este último nodo se muestran a continuación:

```
#Se hacen las suscripciones pertinentes
        self.subscription_odometria = self.create_subscription(
            Vector,
            'odometria',
            self.signal_callback1,
            rclpy.qos.qos_profile_sensor_data )

# Callback para recibir la posición
actual del robot
    def signal_callback1(self, msg):
        if msg is not None:
            self.Posx = msg.x
            self.Posy = msg.y
            self.Postheta = msg.theta
```

En el código anterior se crea el suscriber que escucha al tópico 'odometria' con tipo de dato 'Vector', el cual contiene la posición actual del robot en los ejes 'x', 'y' y theta, para que posteriormente dentro de la función `signal_callback1` sean asignadas a otras variables locales.

```
#Se hacen las suscripciones pertinentes
        self.subscription_path = self.create_subscription(
            Path,
            'path_generator',
            self.signal_callback2,
            rclpy.qos.qos_profile_sensor_data )

def signal_callback2(self, msg):
    if msg is not None:
        self.trayectoria = [(0,0),(msg.x1, msg.y1),
        (msg.x2, msg.y2),  (msg.x3, msg.y3),
        (msg.x4, msg.y4), (msg.x5, msg.y5),
        (msg.x6, msg.y6)]
```

Dentro de esta sección de código se crea el suscriber que escucha al tópico `path_generator` que contiene las distintas coordenadas de cada una de las figuras para ser guardadas dentro de un vector desde el cual las accesaremos posteriormente.

```
#Timer para realizar los calculos correspondientes
self.timer = self.create_timer(0.1, self.timer_callback)

def timer_callback(self):
    [...]
    # Obtener las coordenadas del punto actual
    en la trayectoria
    target_x, target_y = self.trayectoria
    [self.indice_punto_actual+1]

    target_x_ant, target_y_ant =
    self.trayectoria[self.indice_punto_actual]

    # Calcular las coordenadas polares del punto objetivo
    self.distancia = math.sqrt((target_x - self.Posx)**2 +
    (target_y - self.Posy)**2)
    self.angulo_objetivo =
    numpy.arctan2(target_y-target_y_ant,
    target_x-target_x_ant)

    self.errorTheta = self.angulo_objetivo - self.Postheta
```

Dentro de nuestra función `timer_callback` se toman los valores obtenidos de nuestros suscribers con anterioridad y se realizan las operaciones para el cálculo de las coordenadas actuales del robot, las coordenadas objetivo, el valor actual de theta y su valor objetivo. Para que con las restas de los mismos se obtenga el error de tanto la posición, así como del ángulo. De igual manera se envían los valores del error a través de un tópico con el mismo nombre.

De misma forma se llevo a cabo la implementación del control, en nuestro caso decidiendo por un control 'Proporcional' ya que observamos daba resultados correctos para las trayectorias planteadas. Con valores para la variable proporcional de la velocidad lineal de 0.4 y 0.25 para la velocidad angular. Siendo la implementación de un control 'PI' o 'PID' una de las áreas de oportunidad a mejorar en un futuro. Ejemplo de la implementación se muestra a continuación:

```
#Valores de k
    self.kpTheta = 0.25
    self.kpLineal = 0.4

#Control Angular
    self.PTheta = self.kpTheta*self.errorTheta
#Control lineal
    self.PLineal = self.kpLineal*self.error_distancia

#Velocidades
    self.velA = self.PTheta
    self.velL = self.PLineal

#Ceilings
    if self.velA > 0.2:
        self.velA = 0.2

    if self.velL > 0.2:
        self.velL = 0.2

    if self.errorTheta > 0.03 or self.errorTheta < -0.03:
        self.velL = 0.0
```

En el código anterior se toma en cuenta los valores proporcionales para ambas velocidades que son posteriormente multiplicadas por el error para obtener así el valor de las velocidades que serán enviadas posteriormente por el tópico `cmd_vel` estableciendo con anterioridad un límite de 0.2 para la velocidad lineal y un márgen de error angular de más menos 0.03.

V. Resultados

Fig. 1. Video funcionamiento

En el video del funcionamiento del reto se puede observar como el robot cumple satisfactoriamente las trayectorias propuestas presentando ligeras perturbaciones, especialmente en el cambio de las figuras en las que el robot requiera de una rotación angular mayor, ya que pudimos detectar un pequeño error menor al 5% en las lecturas de la odometría, error que aunque pequeño se va sumando al realizar las distintas figuras, de misma forma y para poder observar mejor las figuras realizadas se tuvo que reposicionar al robot al terminar una de las trayectorias, ya que el ligero error dentro de la posición provoco una ligera traslación del marco de referencia inercial. De igual forma en el video se puede observar el funcionamiento correcto del control proporcional implementado, ya que éste acelera a una velocidad mayor de detectar un error grande y realiza la desaceleración correspondiente al disminuir el mismo. Dentro de las áreas de oportunidad podemos encontrar la implementación de un control PI o PID, ya que aunque toma en cuenta la distancia al valor objetivo y realiza los ajustes correspondientes a la velocidad, nuestra implementación actual no toma en cuenta los ligeros errores que se van acumulando.

VI. Conclusiones

Haciendo una comparativa con el reto de la semana pasada, que en lo particular resultó ser mucho más complicado por el enfoque que aplicamos a la hora de resolver la problemática,

este desafío puede ser abordado con un mayor margen de tiempo (lo cual se vio significativamente en los tiempos de desarrollo y conclusión de la actividad) y de cierta forma, adelantados un poco con respecto desde donde partir nuestro enfoque, pues ya contábamos con aproximaciones para entender el cálculo del error y la odometría. Sin embargo, aún carecíamos de ciertos elementos.

El desarrollo de esta actividad permitió mejorar significativamente el desempeño del trazado de trayectorias, eliminando aquellos desajustes o rotaciones adicionales que nuestro modelo anterior realizaba con la intención de reajustar o corregir la trayectoria, los cuales posiblemente fueron el mayor problema presentado en el proyecto pasado. Ahora, con un mejor entendimiento de cómo el PuzzlBot registra estos componentes de su posición y rotación gracias a la prueba y error, hemos avanzado considerablemente en la optimización de su desempeño. Esto se debe en parte a la introducción de actividades intermedias entre el reto de la semana 2 y el de la semana 3, las cuales permitieron reducir el área de trabajo o enfoque, dejándonos solo con lo que deberíamos poner especial atención.

El producto de esta semana cubre los huecos dejados por el reto de la semana 2, sin embargo, aún hay espacio para mejorar la locomoción del PuzzleBot, especialmente en lo que respecta al sistema de control, particularmente en el Tuning del control PID, que aún estamos debatiendo. La solución de esta semana introduce una constante proporcional; sin embargo, esto no elimina del todo el error. A partir de este punto, nos enfocaremos en mejorar el cálculo del error y la respuesta ante él, revisando las posibles alteraciones que el nuevo acoplamiento integrado en el robot puede generar en el cálculo de la odometría. Además, consideraremos la incorporación, si es posible y necesario, del dominio de la derivada o de la integral en nuestro controlador con el fin de hacerlo más exacto y preciso al realizar trayectorias. Con estos ajustes, esperamos lograr un desempeño óptimo del PuzzleBot en futuros retos que están por venir.

REFERENCES

[1] Castellanos Pérez, M., Medina, A., Álvaro Hernández, S., Lester, S., Maza, A., León, I., y Revisores, O. (s.f.). Instituto Tecnológico de Tuxtla Gutiérrez Reporte Final Residencia Profesional: Desarrollo y Aplicación de Algoritmos de Cooperación en Robots Móviles. link.

[2] Cortés, U., Castañeda, A., Benítez, A., y Díaz, A. (2015). Control de Movimiento de un Robot Móvil Tipo Diferencial Robot link.

[3] Seguimiento de Trayectoria Mediante la Solución del Error Lateral en un Robot Tipo Diferencial. (s.f.). link.

[4] Mariana. (2024, Enero 4). Control robusto. Fisicotrónica. link.

[5] ManchesterRoboticsLtd. (s.f.). GitHub - ManchesterRoboticsLtd/TE3002B-2024: Intelligent Robotics Implementation. GitHub. link.