Instituto Tecnológico y de Estudios Superiores de Monterrey

TE3001B Fundamentos de Robótica(Gpo 101)

Reto semanal 2. Manchester Robotics

Autores:

Yestli Darinka Santos Sánchez // A01736992

Emiliano Olguín Ortega // A01737561

Erik García Cruz // A01732440

Profesores:

Juan Manuel Ahuactzin Larios

Rigoberto Cerino Jiménez

Alfredo García Suárez

Jueves 20 de Febrero de 2025 Semestre (6) Feb-Jun 2025

Campus Puebla

Resumen

Este reto tiene como objetivo principal la implementación de un controlador PID en ROS2 para un motor simulado, se requiere la creación de un nodo denominado "/ctrl" que suscriba a los tópicos "set_point" y "motor_output_y" y publique en "motor_input_u", además, se deben utilizar herramientas como rqt_plot, rqt_graph y rqt_reconfigure para analizar y ajustar el comportamiento del sistema, se enfatiza el uso de namespaces y la configuración de parámetros en tiempo de ejecución.

Objetivos

General

Diseñar e implementar un controlador PID para regular la salida de un motor simulado en ROS2

Específicos

- Crear un nodo controlador en ROS2 que implemente un control PID.
- Configurar y modificar parámetros en tiempo de ejecución mediante rqt_reconfigure.
- Utilizar herramientas de análisis como rqt_plot y rqt_graph.
- Implementar namespaces para gestionar múltiples instancias del controlador.
- Desarrollar un sistema capaz de recibir diferentes señales de referencia y generar respuestas adecuadas.

Introducción

El presente reporte tiene como objetivo analizar y describir detalladamente el proceso de implementación y utilización del Sistema Operativo para Robots (ROS), particularmente en el contexto de la simulación y control de sistemas dinámicos. ROS es una plataforma ampliamente utilizada en la robótica moderna debido a su capacidad para proporcionar herramientas y estructuras que permiten a los desarrolladores crear, probar y gestionar aplicaciones robóticas de manera más eficiente. En este reporte, se explorarán diversos aspectos fundamentales de ROS, tales como el manejo de namespaces para evitar colisiones de nombres en nodos y temas, la configuración de parámetros dentro de los nodos para asegurar una ejecución adecuada, la utilización de servicios para facilitar la comunicación entre nodos de manera eficiente, y la implementación de interfaces personalizadas para satisfacer necesidades específicas. El análisis se centra en cómo estos elementos contribuyen al desarrollo ágil y organizado de aplicaciones robóticas, permitiendo que el sistema se

adapte a distintas exigencias y escale según sea necesario sin comprometer la modularidad y la integridad del sistema.

Solución del problema

Implementación del Controlador PID

Se creó un nodo denominado controller_robotronicos, el cual implementa el algoritmo PID en ROS2. Se declararon y configuraron los parámetros del controlador para permitir ajustes en tiempo real mediante rqt_reconfigure. A continuación, se presenta una parte importante del código implementado:

```
def __init__(self):
    super().__init__('controller_robotronicos')

    # Parámetros del PID
    self.declare_parameter('kp', 1.0)
    self.declare_parameter('ki', 0.1)
    self.declare_parameter('kd', 0.01)
    self.declare_parameter('dt', 0.01)

    self.kp = self.get_parameter('kp').value
    self.ki = self.get_parameter('ki').value
    self.kd = self.get_parameter('kd').value
    self.dt = self.get_parameter('dt').value

    # Suscripciones y Publicaciones
    self.subscription_setpoint = self.create_subscription(Float32, '/set_point_robotronicos', self.setpoint_callback, 10)
    self.subscription_output = self.create_subscription(Float32, '/motor_speed_y_robotronicos', self.output_callback, 10)
    self.publisher = self.create_publisher(Float32, '/motor_input_u_robotronicos', 10)

    self.add_on_set_parameters_callback(self.parameters_callback)

    # Variables PID
    self.setpoint = 0.0
    self.output = 0.0
    self.integral = 0.0
    self.prev_error = 0.0
    self.anterior = 0.0

    # Timer para control en tiempo real
    self.timer = self.create_timer(self.dt, self.timer_callback)
```

```python
def timer_callback(self):
    error = self.setpoint - self.output
    self.integral += error * self.dt
    self.integral = np.clip(self.integral, -6, 6)

    proportional = self.kp * self.prev_error
    integrative = self.ki*self.integral
    derivative = self.kd * ((self.prev_error - self.anterior) / self.dt)

    control_signal = proportional * integrative + derivative

    self.anterior = self.prev_error

    msg = Float32()
    msg.data = control_signal
    self.publisher.publish(msg)

    self.motor_msg = Float32()

    self.motor_msg.data = msg.data
    self.publisher.publish(self.motor_msg)
```

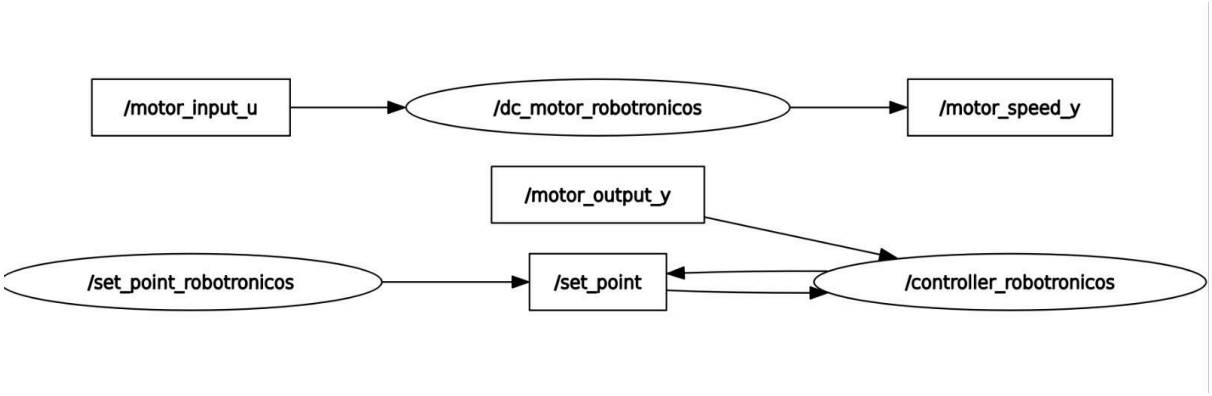
Implementación del Motor Simulado

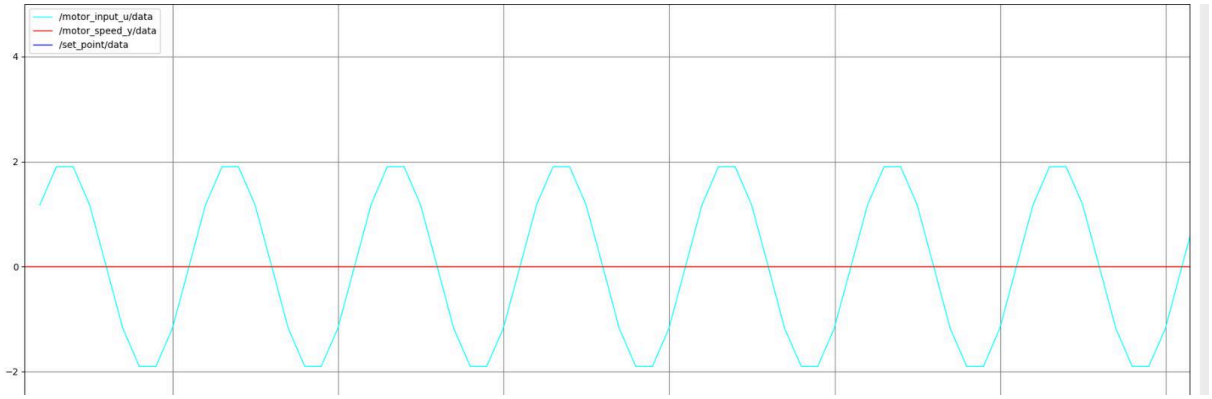
Se creó un nodo denominado dc_motor_robotronicos, el cual simula un motor de corriente continua basado en una ecuación diferencial de primer orden:

```python
def __init__(self):
    super().__init__('dc_motor_robotronicos')

    # Parametros del motor
    self.declare_parameter('sample_time', 0.01)
    self.declare_parameter('sys_gain_K', 2.16)
    self.declare_parameter('sys_tau_T', 0.05)
    self.declare_parameter('initial_conditions', 0.0)

    self.sample_time = self.get_parameter('sample_time').value
    self.param_K = self.get_parameter('sys_gain_K').value
    self.param_T = self.get_parameter('sys_tau_T').value
    self.output_y = self.get_parameter('initial_conditions').value

    # Publicación y suscripción
    self.motor_input_sub = self.create_subscription(Float32, '/motor_input_u_robotronicos', self.input_callback, 10)
    self.motor_speed_pub = self.create_publisher(Float32, '/motor_speed_y_robotronicos', 10)
    self.timer = self.create_timer(self.sample_time, self.timer_cb)
    self.add_on_set_parameters_callback(self.parameters_callback)

    # Mandar mensajes
    self.motor_msg = Float32()
```

```python
def timer_cb(self):
    self.output_y += (-1.0 / self.param_T * self.output_y + self.param_K / self.param_T * self.input_u) * self.sample_time
    self.motor_msg.data = self.output_y
    self.motor_speed_pub.publish(self.motor_msg)
```

Resultados





Los resultados obtenidos no coinciden completamente con los esperados, aunque el controlador PID logró generar una respuesta del sistema, se observó que la señal de salida no seguía correctamente el setpoint y presentaba oscilaciones no deseadas. En comparación con los resultados esperados, donde se observa una respuesta más estable y rápida, los datos obtenidos indican que el sistema necesita una mejor sintonización de los parámetros PID para mejorar la estabilidad y el tiempo de establecimiento.

Conclusión

El desarrollo del controlador PID en ROS2 permitió regular la salida del motor simulado, pero los resultados obtenidos muestran que aún hay margen de mejora.s

¿Se lograron los objetivos? Parcialmente. Se implementó el controlador PID y se validó su desempeño, pero la respuesta del sistema no fue óptima.

¿No se cumplieron completamente los objetivos? La señal del motor presentó oscilaciones y no siguió correctamente el setpoint, lo que sugiere que la sintonización del PID no fue adecuada.

¿Cuál sería una posible mejora? Se recomienda realizar un ajuste más preciso de los parámetros PID, utilizando métodos avanzados de sintonización, como optimización basada en algoritmos genéticos o el método de ensayo y error con métricas de desempeño más detalladas.

Bibliografía

Fmrico. (s. f.). *GitHub - fmrico/book_ros2*. GitHub. https://github.com/fmrico/book_ros2

Understanding nodes — ROS 2 Documentation: Humble documentation. (s. f.). https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html

Creating a package — ROS 2 Documentation: Humble documentation. (s. f.). https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html

Creating a workspace — ROS 2 Documentation: Humble documentation. (s. f.). https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html