Instituto Tecnológico de Estudios Superiores de Monterrey

Fundamentación de robótica & Manchester Robotics

Manchester Robotics: Challenge 2

Profesores:

Rigoberto Cerino Jiménez

Dr. Mario Martinez

Integrantes

Daniel Castillo López A01737357

Emmanuel Lechuga Arreola A01736241

Paola Rojas Domínguez A01737136

27 de Febrero de 2025

Índice

Resumen

El desafío 2, se centra en el desarrollo de un sistema de control basado en el sistema de comunicación ROS 2, para un motor simulado. Donde debemos de crear un controlador P, PI o PID, que reciba una señal de referencia y ajuste la entrada del motor para seguir esa referencia de la manera más precisa posible. Por ello empleamos herramientas de ROS 2, como los namespaces, parámetros configurables y mensajes personalizados, lo que nos permite una implementación flexible y modular, además de ordenada. La actividad nos permite experimentar con las configuraciones y el ajuste en tiempo real de un controlador PID.

Objetivos

Los siguientes objetivos buscan una comprensión integral de cómo desarrollar sistemas básicos utilizando ROS 2 y sus herramientas.

- ➔ Revisar los conceptos introducidos en las sesiones anteriores
- ➔ Reforzar los conceptos sobre controladores
- ➔ Crear un controlador PID para un motor DC simulado por ROS
- ➔ Implementar namespaces y parámetros

Introducción

Ros o mejor conocido como Robot Operating System, es un framework (entorno de trabajo) abierto para el desarrollo de software en robótica, lo cual lo convierte en un conjunto de herramientas bibliotecas y convenciones que facilitan la creación de sistemas robóticos complejos en los cuales se pueden crear redes de comunicación multiplataforma. (ROS Wiki 2023).

Namespaces

Los namespace en ROS2 sirven para organizar y estructurar la comunicación entre nodos, evitando conflictos de nombres y permitiendo una modularidad en sistemas robóticos grandes; Son similares a las carpetas en un sistema de archivos, donde los nodos, temas y servicios pueden estar agrupados bajo un mismo espacio de nombres.

- Forma de trabajo con namespaces:
    - Todo lo que es trabajar con namespaces es declararlo o ocuparlos en el launch (lugar donde se encuentran los nodos), ahora al momento de organizar los namespaces se les debe de incluir un identificador. por ejemplo si tienes múltiples motores, a uno le nombrará motor1 y motor2, y así sucesivamente, sin necesidad de andar declarando muchas veces lo que es un nodo para cada motor, sino que con el namespaces tomas el nodo y lo duplicas y le agregas un

nombre, numero que pueda hacer que lo identifiques de una manera más eficaz y eficiente.

Parameters

En ROS 2, los parámetros están asociados a nodos individuales y permiten configurar su comportamiento en el inicio y en tiempo de ejecución sin modificar el código. Su duración está ligada a la del nodo, aunque pueden persistir para recargarse tras un reinicio. Algunas de sus características son:

- Se identifican por nombre de nodo, espacio de nombres, nombre de parámetro y espacio de nombres del parámetro (opcional).
- Constan de una clave (string), un valor (de tipo bool, int64, float64, string, byte[], bool[], int64[], float64[], string[]) y un descriptor (opcional).
- Un nodo debe declarar los parámetros que usará, salvo que permita parámetros no declarados (allow_undeclared_parameters = true).

Callbacks de Parámetros

ROS 2 permite registrar tres tipos de callbacks para gestionar cambios en los parámetros:

- Pre-set parameter callback: Se ejecuta antes de modificar parámetros, permitiendo ajustes o cambios en otros parámetros relacionados.
- Set parameter callback: Evalúa los cambios antes de aplicarlos y puede rechazarlos.
- Post-set parameter callback: Se ejecuta después de un cambio exitoso, permitiendo reacciones al nuevo valor.

Configuración de Parámetros

- Se pueden definir valores iniciales en la ejecución mediante la línea de comandos o archivos YAML.
- En lanzamientos con ROS 2 launch, se pueden establecer parámetros en la configuración del nodo.

Custom messages

En ROS 2, los mensajes personalizados son definiciones creadas por el usuario que amplían el conjunto de tipos de mensajes disponibles en el sistema. Estos mensajes permiten definir estructuras de datos específicas que se ajustan a las necesidades particulares de su aplicación robótica, facilitando la comunicación entre nodos con formatos de datos adaptados a casos de uso concretos.

Para crear un mensaje personalizado en ROS 2, se sigue generalmente este proceso:

- Crear un paquete de interfaces: Se genera un nuevo paquete dedicado a contener las definiciones de mensajes y servicios personalizados. Este enfoque modular ayuda a

mantener una arquitectura limpia y facilita la reutilización de las interfaces en otros paquetes.

- Definir los archivos de mensaje (.msg): Dentro del directorio msg del paquete, se crean archivos .msg que describen la estructura del mensaje.
- Modificar los archivos de configuración: Es necesario actualizar los archivos CMakeLists.txt y package.xml del paquete para incluir dependencias y configuraciones relacionadas con la generación de interfaces.
- Compilar el paquete: Después de definir los mensajes y ajustar las configuraciones, se compila el paquete utilizando herramientas como colcon.

Controlador PID

El controlador PID (Proporcional-Integral-Derivativo) es un sistema de control ampliamente utilizado en la industria para regular variables como temperatura, presión, velocidad y posición. Su popularidad se debe a su capacidad para minimizar el error de un sistema de control a lo largo del tiempo ajustando tres términos clave: proporcional (P), integral (I) y derivativo (D).

Término Proporcional (P)

Representa la respuesta inmediata del sistema ante un error, está determinado por la ecuación:

$$u(t) = K_p \cdot e(t)$$

donde K_p es la ganancia proporcional y $e(t)$ es el error en el tiempo actual. Si K_p es demasiado alto, puede generar oscilaciones en la respuesta del sistema.

Término Integral (I)

Acumula el error a lo largo del tiempo para corregir el error estacionario, está representado con la siguiente ecuación:

$$u(t) = K_i \cdot \sum e(t) \cdot T$$

donde K_i es la ganancia integral y T es el tiempo de muestreo. Si K_i es demasiado alto, puede causar sobrecorrección y oscilaciones.

Término Derivativo (D)

Anticipa futuros errores analizando la velocidad de cambio de error. Su ecuación es:

$$u(t) = K_d \cdot \frac{de(t)}{dt}$$

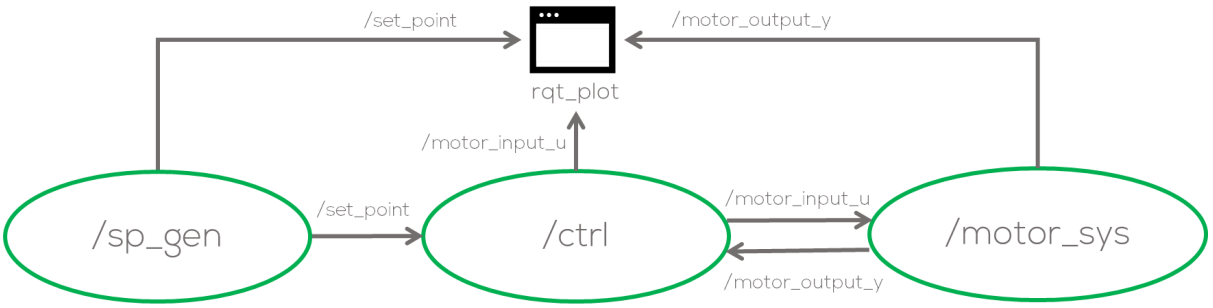
donde K_d es la ganancia derivativa. Su función principal es reducir la sobresaturación y mejorar la estabilidad del sistema.

Método Prueba y Error

Para obtener los valores correspondientes a K_p, K_i y K_d existen distintos métodos, pero para este reto utilizaremos el de prueba y error. Este es un método empírico que se basa en ajustar manualmente los valores de las constantes mientras se observa la respuesta del sistema.

Se comienza ajustando K_p, luego K_i y por último K_d hasta obtener una respuesta del sistema apropiada.

Solución del problema



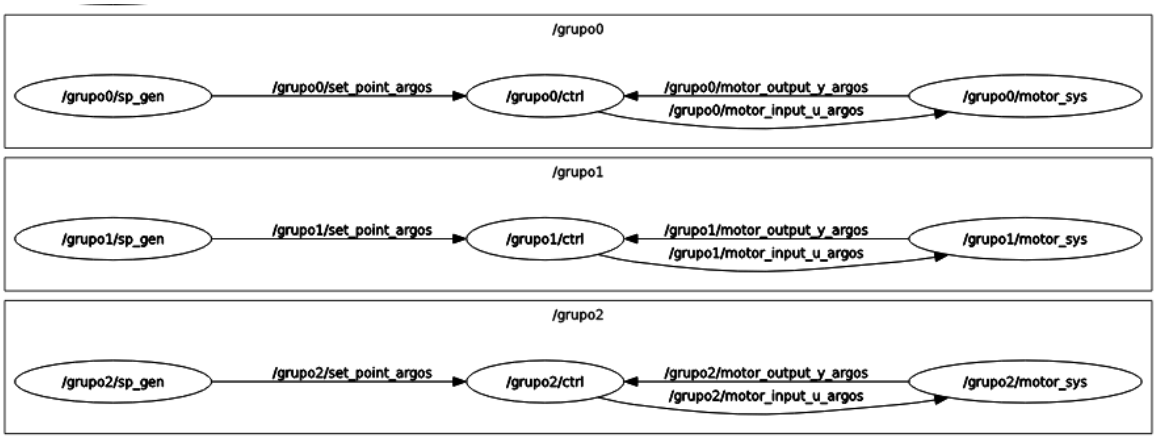
1. Nodo sg_gen
    - Nuestro nodo, creamos un publisher con el nombre /set_point_argos, que será enviado para el nodo ctrl y este siendo una onda de referencia para el control , este mensaje se desarrolló como un tipo Float32
    - Definimos dos parámetros importantes para la señal, siendo la amplitud y el periodo. La definición del periodo se establece aproximadamente 2π lo que define la duración de un ciclo completo de nuestra señal
    - La inda se establece como una onda cuadra, iniciamos con el cálculo de la señal en su valor actual y con un % se determinó en qué parte del ciclo está recorriendo, si el tiempo transcurrido dentro del ciclo es menor a la mitad del periodo, la señal se vuelve en un valor positivo de amplitud, de lo contrario se vuelve en un negativo, lo que nos da la forma de la onda cuadrada
2. Nodo ctrl
    - Definimos los parámetros del controlador, con las ganancias Kp, Ki, Kd y el tiempo de muestreo, se definen como parámetros dinámicos, lo que permite ajustarlos durante la ejecución sin detener el nodo
    - Nuestro nodo Ctrl, está suscrito a la señal de /set point argos para recibir el valor de referencia y /motor_output_y_agos, que designará el valor de salida de nuestro motor, como también definimos un publisher de nombre /motor_input_u argos para publicar la señal del de control que ajustará el motor
    - Se establecen valores del controlador PID con valores predeterminados, esto se calcula el error como la diferencia entre el valor de referencia y la salida del

motor.Luego calculamos el término integral acumulando el error en el tiempo, con una limitación entre -10 y 10 para evitar alguna saturación. El término derivativo se calcula usando la diferencia entre el error actual y el anterior, entrando también una división entre el tiempo de muestreo, dando como resultado la suma entre estos la señal que se le asignará

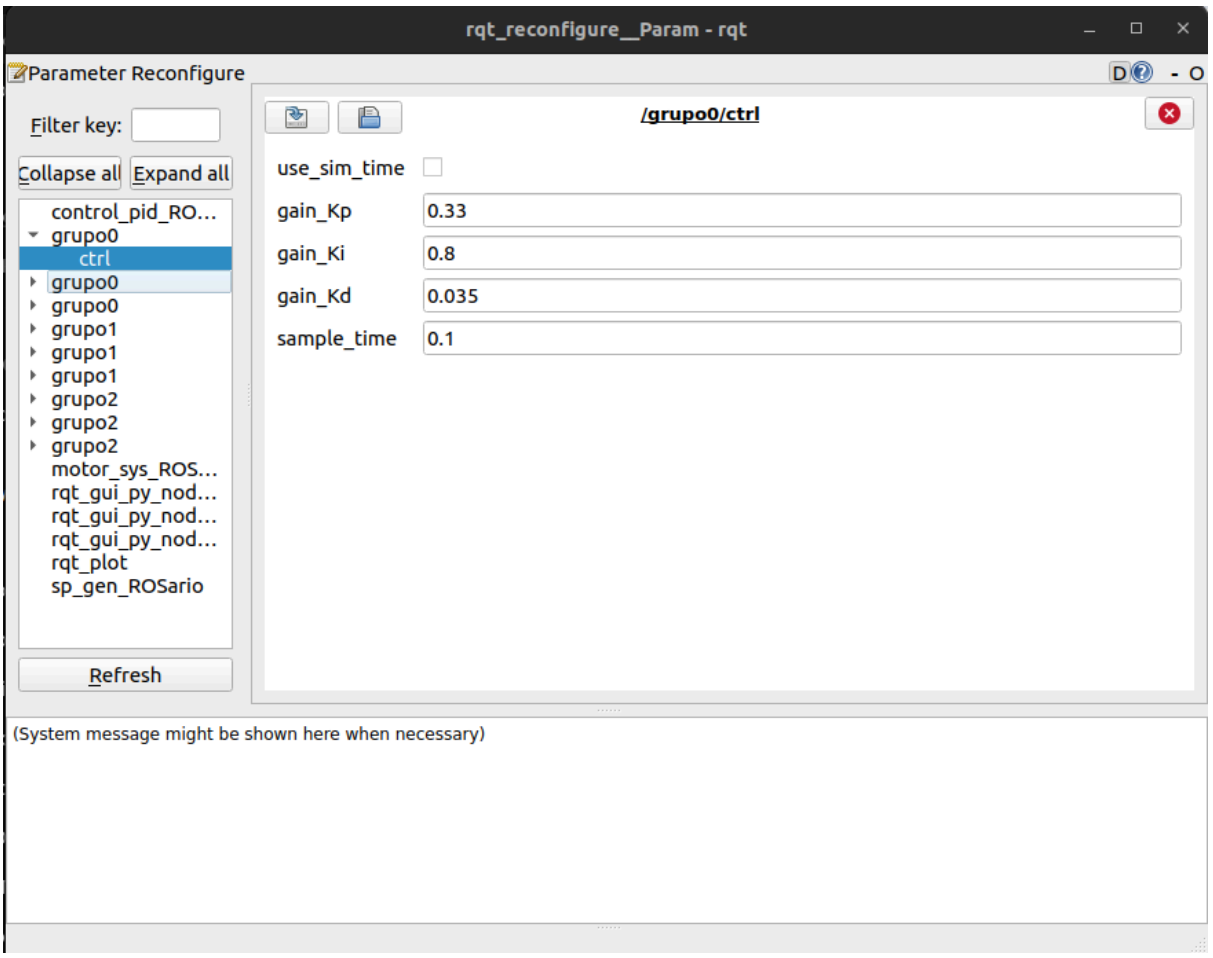
- Como también da la capacidad de actualizar los parámetros en la función parameters_callback(), revisando los valores que sean positivos antes de aplicarlos.

3. Nodo motor_sys
    - Esto simula el comportamiento de un motor de corriente,utilizando una ecuación diferencial discreta, que se le designan parámetros para el motor,sample_time,sys_gain_K,sys_tau_T e initial conditions
    - EL motor está suscrito al canal /motor_input_u_argos, recibe las señales de entrada del nodo Ctrl y también tiene relacionado un publisher /motor_output_y_argos
    - En la simulación del control se genera la simulación del motor de corriente continua usando una ecuación diferencial discreta
4. Launch
    - En la parte del launch, declaramos los nodos para el motor controller, además de eso añadimos los namespace para que de esta manera ocuparamos las mismas declaraciones que se realizaron de un solo nodo y duplicamos o en este caso triplicamos los nodos, y se dividieron en grupo, que sería desde el grupo 0 al 2, dandonos como resultado la imagen presentada a continuación en la cual se puede observar como tenemos los nodos y estos encerrados cada uno en un grupo, y esto se desarrolla cuando tenemos más de un motor, sensor, etc conectado al sistema.

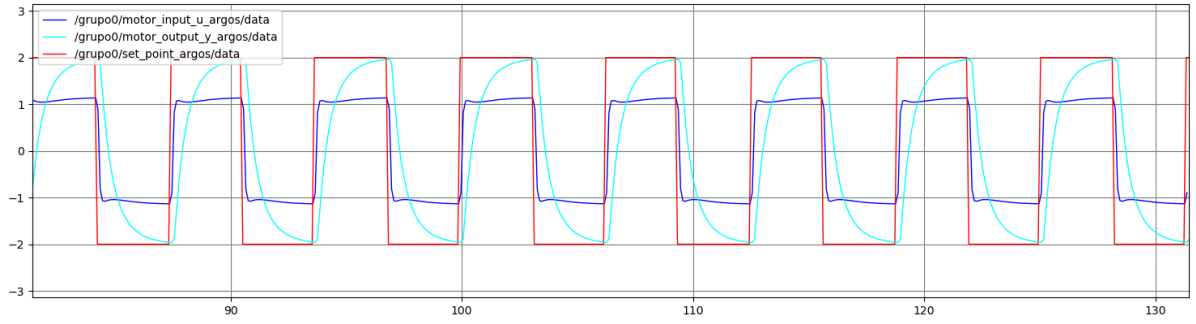
Resultados



En representación de la arquitectura de nuestro sistema de control, se divide en tres grupos, la arquitectura interior muestra la conexión de lo 3 nodos y de cómo en el nodo ctrl, recibe retroalimentación del motor , siendo la salida del motor de velocidad



Se abre la interfaz de rqt_reconfigure, que permite ajustar dinámicamente los parámetros del nodo en ejecución, en este caso es el del grupo 0 el nodo de control, facilita el ajuste fino del controlador para optimizar el desempeño.



Siendo la respuesta a la definición de nuestros parámetros el PID ajusta continuamente la señal del control para buscar minimizar el error del setpoint y la salida.Todo este análisis se puede observar en el grupo cero con las variables previamente establecidas.

Conclusiones

En este desafío se nos ofreció la oportunidad a los estudiantes de aplicar conocimientos de ROS2 en el entorno de Linux, donde podemos practicar, implementar y ajustar las características adecuadas para el uso del programa ROS2, A través de herramientas como lo son rqt_plot y rqt_graph, donde podemos visualizar y analizar el desempeño del sistema, ajustando parámetros en tiempo real para lograr un control más eficiente, además de la implementación de los namespaces que ayuda a la gestión de los nodos, los parámetros que ayudan a modificar el comportamiento de los nodos , los mensajes personalizados para enviar y recibir datos y la sintonización de controladores PID.

En pocas palabras el Challenge 2 es una excelente oportunidad para que desarrollemos nuestras habilidades esenciales con ROS2 y así comprendamos la importancia de la integración de sistemas de control robóticos, cumpliendo tanto con el objetivo teórico de investigación como el objetivo práctico donde desarrollamos el sistema de control

Referencias

Parameters — ROS 2 Documentation: Rolling documentation. (s. f.). https://docs.ros.org/en/rolling/Concepts/Basic/About-Parameters.html

Microchip Technology Inc. (s.f.). *Discrete PID Controller on tinyAVR and megaAVR (Application Note AVR221)*. Microchip Technology Inc. Recuperado el 27 de febrero de 2025, de https://onlinedocs.microchip.com/oxy/GUID-58BE6228-CBD9-413C-8876-FE2E0B8F1DB3-en-US-3/index.html

Creating custom ROS 2 msg and srv files — ROS 2 Documentation: Crystal documentation. (s. f.). https://docs.ros.org/en/crystal/Tutorials/Custom-ROS2-Interfaces.html