Instituto Tecnológico de Estudios Superiores de Monterrey

Fundamentación de robótica & Manchester Robotics

Manchester Robotics: Challenge Final.

Profesores:

Rigoberto Cerino Jiménez

Dr. Mario Martinez

Integrantes

Daniel Castillo López A01737357

Emmanuel Lechuga Arreola A01736241

Paola Rojas Domínguez A01737136

14 de Marzo de 2025

Índice

Resumen

El resumen presenta una síntesis de los aspectos más relevantes del proceso de investigación y desarrollo llevado a cabo, destacando los objetivos, la metodología empleada y los principales resultados obtenidos.

El presente proyecto tiene como objetivo el desarrollo de un sistema de control con velocidad para un motor de corriente continua (DC), utilizando un enfoque basado en ROS 2 y ejecutando en una máquina (computadora) externa y un microcontrolador (MCU).

Proceso de investigación.

- Modelado del motor DC
    - Obtención de la función de transferencia mediante parámetros eléctricos y mecánicos del motor como el datasheet.
    - Medición de valores resistencia, inductancia, constante de torque, momento de inercia y coeficiente de fricción viscosa.
- Diseño e implementación de los nodos en ROS 2.
    - Nodo /input: pública señales de referencia en /set_point con diferentes tipos de onda (senoidal , cuadrada y escalón unitario).
    - Nodo / Control: Ejecutando en la MCU, recibe la referencia, la convierte en señales PWM para el driver del motor y estima la velocidad con los datos del encoder.
- Implementación del controlador
    - Diseño y ajuste de un controlador P, PI o PID, en nuestro caso ocupamos PID basado en la función de transferencia obtenida por las mediciones y documentación del motor, junto con la creación de un esquemático en matlab de simulink para obtener valores más precisos para el PID.
    - Implementación de un sistema del cual se pueda ajustar los parámetros en tiempo real.
- Pruebas y validación de los recursos.
    - Evaluación del comportamiento del sistema con diferentes entradas (señales), como.
        - Senoidal.
        - Cuadrática.
        - Escalón unitario.

- Uso de herramientas como rqt_plot y rqt_graph para visualizar la respuesta del control del motor en base a cómo este reacciona con el pwm asignado además de mostrar los tópicos, nodos .

Objetivos

En esta sección se presentan el objetivo general y los objetivos particulares del reto, los cuales están enfocados en el diseño e implementación de un sistema de control PID para el PWM de un motor de corriente continua a traves del uso de ROS 2 y Micro-ROS

Objetivo General: Desarrollar un sistema de control de velocidad para un motor DC, utilizando una arquitectura basada en ROS 2, donde la computadora externa y el microcontrolador trabajen en conjunto para recibir una referencia de velocidad angular, generar señales PWM y estimar la velocidad angular del motor a partir de datos del encoder.

Objetivos particulares:

- Implementar el nodo /Input para generar señales de referencia en /set_point_argos, permitiendo la selección de diferentes tipos de onda (escalón, senoidal, cuadrada).
- Diseñar e implementar el nodo /Motor_control_argos en la MCU (ESP32), encargándose de:
    - Convertir la señal de /set_point_argos en señales PWM y dirección para el motor.
    - Leer los datos del encoder y estimar la velocidad angular del motor.
    - Publicar la velocidad estimada en /motor_output_argos.
- Modelar la dinámica del motor DC mediante la obtención de su función de transferencia y los parámetros característicos (L,J,R,B,Km,Ka).
- Desarrollar un controlador PID para regular la velocidad del motor asegurando estabilidad y un tiempo de respuesta óptimo.
- Configurar el sistema para permitir el ajuste dinámico de parámetros mediante archivos de configuración en ROS 2 .
- Validar el desempeño del sistema a través de pruebas experimentales y visualización en herramientas como rqt_plot.

Introducción

La introducción presenta el tema de investigación, proporcionando el contexto necesario para comprender la relevancia del reto a realizar. Se ofrece una visión general que facilita la construcción de un esquema mental sobre las metodologías utilizadas, además de los conceptos y tecnologías que se abordarán en el desarrollo del reporte.

Metodología de trabajo: Extreme Programming (XP)

En el desarrollo del presente proyecto, se ha seleccionado la metodología de trabajo Extreme Programming, la cual permite el desarrollo continuo de códigos funcionales mediante iteraciones cortas, integración frecuente y retroalimentación constante. Se escogió esta estrategia para este proyecto ya que se enfoca en la mejora continua del código, empezar con las funciones más simples para posteriormente ir adaptando a cambios en el desarrollo de los nodos en ROS2 y la implementación en micro-ROS. La metodología Xp se implementó en el desarrollo del reto siguiendo los siguientes principios:

- **Desarrollo incremental:** El desarrollo se realizó en pequeñas iteraciones, comenzando con las funciones más básicas e ir mejorando progresivamente.
- **Comunicación y reuniones:** Este punto es para garantizar la alineación del equipo y permite la rápida toma de decisiones.
- **Pruebas frecuentes:** Cada módulo y modificación fue probado antes de su integración en el sistema final. Las herramientas como rqt_plot y rqt_graph fueron esenciales para analizar el comportamiento del sistema y verificar la respuesta del controlador PID
- **Retroalimentación:** Cada iteración fue evaluada para detectar posibles mejoras en la implementación de los nodos y la comunicación entre ROS2 y micro-ROS

Para la organización del equipo, se asignaron roles específicos a cada uno de los tres integrantes, además de las actividades colaborativas para garantizar la integración de diferentes módulos del sistema:

- Emmanuel Lechuga: Enfoque principal en encontrar la función de transferencia del motor y realizar el ajuste de los parámetros del controlador PID
- Daniel Castillo: Desarrollo de los nodos en ROS2 y configuración de la comunicación en micro-ROS.
- Paola Rojas: Implementación del steck en micro-ROS para la ejecución en ESP32.

Aunque se asignaron actividades particulares, todos los integrantes del equipo colaboraron en conjunto en la solución de errores, integración de módulos y verificación del funcionamiento del paquete en ROS, conociendo así el sistema por completo.

El proyecto se desarrolló en cuatro fases basadas en la metodología XP, a continuación se muestran estas fases en la Figura 1.

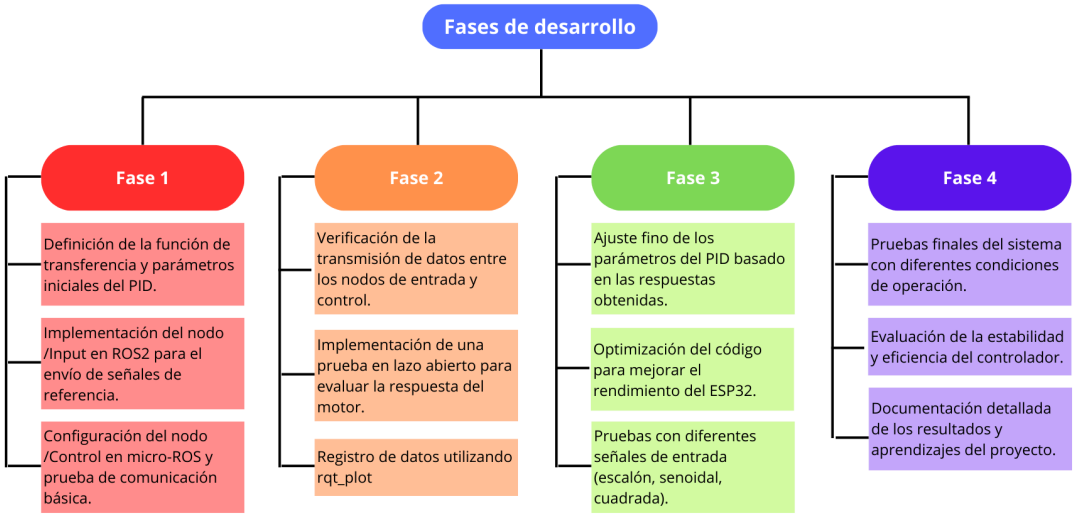

Figura 1. Diagrama de fases de desarrollo

La implementación de Extreme Programming permitió un desarrollo más organizado del reto, asegurando la calidad en el código. Con iteraciones cortas, pruebas constantes y refactorización continua, para obtener un sistema robusto que cumpla con los requerimientos del desafío propuesto.

Descripción del reto

El control de velocidad de motores de corriente continua (DC) es un aspecto fundamental en el diseño de sistemas robóticos, ya que permite optimizar el rendimiento de dispositivos que dependen de movimientos precisos y regulados. En este proyecto, se desarrolló un sistema de control de velocidad para un motor DC, utilizando una arquitectura basada en ROS 2 y ejecutada en una computadora externa y un microcontrolador (MCU).

Para llevar a cabo esta tarea, se diseñaron e implementaron dos nodos principales en ROS 2 en la computadora externa:

Nodo /Input, ejecutado en la computadora externa, encargado de generar y publicar señales de referencia (/set_point_argos) con distintas formas de onda ya sea: senoidal, cuadrada y escalón, permitiendo probar el desempeño del controlador con diferentes tipos de entrada.

Nodo /Control, ejecutado en la MCU (ESP32), responsable de recibir el set_point, convertirlo en señales PWM para el motor y estimar su velocidad a partir de los datos del encoder, publicando la información en el tópico /motor_output_argos.

El controlador implementado en el nodo /Control fue diseñado para operar en un esquema de lazo cerrado, empleando un enfoque P, PI o PID, con parámetros ajustables en tiempo real. Para garantizar la estabilidad y el desempeño óptimo del sistema, se estableció un proceso de ajuste y calibración basado en la observación de las señales en rqt_plot, permitiendo evaluar la respuesta del motor en distintas condiciones de operación.

Además aplicamos técnicas de procesamiento de señales para mejorar la estimación de la velocidad del motor, considerando factores como el ruido en la medición del encoder y las limitaciones en la resolución de los pulsos. El sistema también fue diseñado para cumplir con las restricciones establecidas en el desafío, utilizando únicamente la bibliotecas estándar así como NumPy y evitando el uso de controladores predefinidos en el código o algún uso de control remoto no especificado en el reto.

Uno de los principales desafíos del proyecto radica en diseñar un sistema de control eficiente para un motor DC, uno en el que se garantice la estabilidad y precisión de la velocidad.

En cuanto a la parte antes mencionada de garantizar un sistema de control. fue una de las partes más laboriosas con la obtención de los parámetros característicos de la función de transferencia, debido a que al momento de realizar la investigación y encontrar el datasheet del motor RK-370CA-2217, no proporciona muchos de los datos necesarios como lo podría ser la viscosidad, la inductancia, la resistencia entre otros, es por eso mismo es que se tuvo que realizar investigación sobre los métodos heurísticos necesarios para la obtención de las variables faltantes, y al momento de obtenerlos pudimos finalizar los parámetros que necesitábamos para la FT (Función de transferencia).

Los resultados obtenidos demostraron la efectividad del sistema de control, evidenciando una respuesta estable y precisa ante los cambios en la referencia de velocidad. Se realizaron múltiples pruebas experimentales para evaluar el comportamiento del controlador, comparando las respuestas del motor en cada una de las configuraciones implementadas. Finalmente, se documentó el desempeño del sistema mediante análisis gráfico y registros en ROS 2, permitiendo una mejor comprensión de su funcionamiento y posibles mejoras futuras.

Este proyecto no solo permitió la implementación de un sistema de control en hardware real, sino que también reforzó el uso de ROS 2 como una herramienta para la integración y gestión de sistemas embebidos avanzados con uso específico en la robótica. Trasladándose a un escenario real el uso de este código puede ser escalado e implementado en distintas industrias gracias al uso de ROS 2, que es un framework (entorno de trabajo) con el cual es sencillo aplicarse para el desarrollo de software en robótica. y este proyecto si se aumentara o especializada, se podría alcanzar un control de velocidad precisa una en la que garantice un desplazamiento estable, para brazos robóticos (ejemplo), lo que permitiría movimientos suaves, precisos y seguros en tareas de manipulación.

Solución del problema

A continuación, se describe la metodología empleada para alcanzar los objetivos del reto, detallando los elementos utilizados, las funciones implementadas y el desarrollo del código de programación.

Para desarrollar una metodología nos dividimos la estructura del proyecto en diferentes partes, que nos faciliten el entendimiento de diferentes áreas que puedan al final unirse en conjunto con los elementos y funciones del proyecto y así poder presentar nuestro objetivo final con sus partes implementadas como la programación del código que abarca mayor parte del reto.

Tomando como base lo que se presenta en la Figura 2, la estructura que llevaremos a cabo se presenta como un control de velocidad para un motor DC, siguiendo las características de tener dos nodos que en este caso sería /Input y /Control el que generará la retroalimentación de nuestro control.

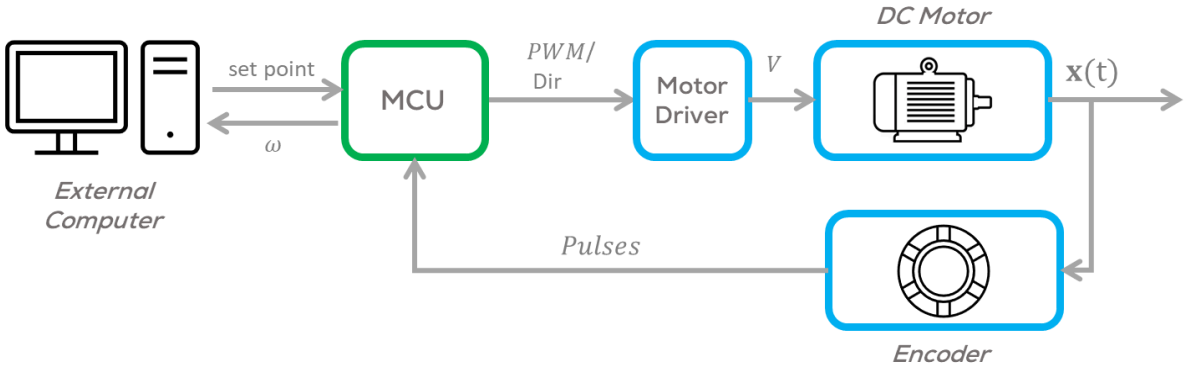

Figura 2. Diagrama de estructura

Input node

Para iniciar con la estructura, se creó un nodo con las de ROS2 con el nombre /Input este nodo se encargará de publicar la salida del /set_point_argos, dicho esto el set_point será la onda de entrada que definirá el transcurso de nuestro motor, para este desarrollo se establece los parámetros que usaremos en nuestra onda.

Como también existe una suscripción que tiene este nodo para recibir los valores del /motor_ouput_argos, este valor es definido por el nodo control que será la que publicara la velocidad angular, este cambio dará acceso a que el usuario pueda ver los valores cambiantes de nuestra velocidad de nuestro encoder por medio de una computadora externa.

Después de definir los valores de entrada y salida establecimos parámetros para que el usuario pueda desarrollar cambios en nuestra onda en tiempo real, dichas variables son: amplitud, periodo y frecuencia estas variables son para la creación de nuestra onda, pero también el usuario podrá seleccionar el tipo de onda de su preferencia en el parámetro ”type” que hace cambios para poder generar una onda cuadrada, senoidal o un escalón unitario,

también encontramos un parámetro observador que imprime el valor que esté recibiendo de las subscripciones que en este caso es nuestro /motor_output_argos.

Creamos una función que generará la onda y se utilizará por el usuario, tomando en cuenta los parámetros y calculando el tiempo transcurrido, con esto se presenta en seleccionar el tipo de onda que se busca realizar. Para terminar con la construcción de nuestro nodo, usamos una función llamada parameters_callback que analizará las respuestas de los cambios en los parámetros que el usuario haga, y generará una respuesta dependiendo los valores que estén permitidos.

Launch file

Con el nodo ya creado lo agrupamos con un launch con el nombre final_launch_argos que se encarga de agrupar el input y los parámetros, estos se encuentran descritos en un YAML que define los valores de iniciación y permite modificarlos en cualquier momento, con esto adjuntamos la entrada del plot_node que se encarga de abrir la gráfica de plot,está con los valores de los canales /set_point_argos, /control_output_argos y /pwm_argos, como se encuentra en la Figura 9, también el grafo que se demuestra en la Figura 12 que es encargado de señalar la conexión de nuestros canales y nodos, como último detalle el reconf_node que abre de inmediato la configuración de los valores con un rqt_reconfigure, esto nos permite una pestaña que sea mucho más fácil de modificar los parámetros en tiempo real. Todos los valores generados en la función son agrupados y unidos para utilizarlos inmediatamente al iniciar el código del programa.

Control node

Siendo el apartado donde se desarrollan mayor parte de las acciones de nuestro motor, esta sección es desarrollada en el microcontroller unit (MCU), esto presenta una ventaja por ser un sistema embebido dedicado esto manteniendo en mejor estado nuestro muestreo y controlador puede ser estable,siendo el medio de comunicación MIcroRos el encargado de poder enviar nuestro nodo a dentro del microcontrolador.

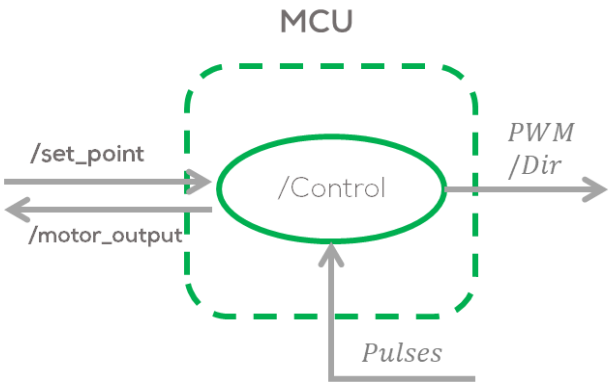

Figura 3. Diagrama del nodo de control

Como lo describe la Figura 3, nuestro control tiene una subscripción del set_point_argos que establece el nodo del Input, pero también crea 2 publicadores que en este caso será el pwm_argos el que enviará las indicaciones al motor DC y publicará el valor de la velocidad angular con los datos que reciba del Encoder, dichos datos serán recibidos con con la entrada de pulses vistos en la Figura 3, estos valores de entrada y salida definirán la estructura de nuestro nodo.

Primero declaramos los nodos que serán usados, esto siendo la representación del nodo de ROS 2 corriendo en el microcontrolador, también registrando los ejecutores que nos ayudará en los timer y callback para el código, definiendo esto podremos establecer los suscriptores y publicadores, que tendrán relación con la declaración de los mensaje utilizados con el tipo de mensaje Float32. La especificación que viene siendo los pines que estableceremos con el puente H y los que recibirán los del encoder están basados en la Figura 4, además con el mensaje del PWM designamos un valor de la frecuencia y resolución del PWM siendo esto 980 Hz nuestro estándar y la señal del duty cicle se establece los 8 bits para su resolución.

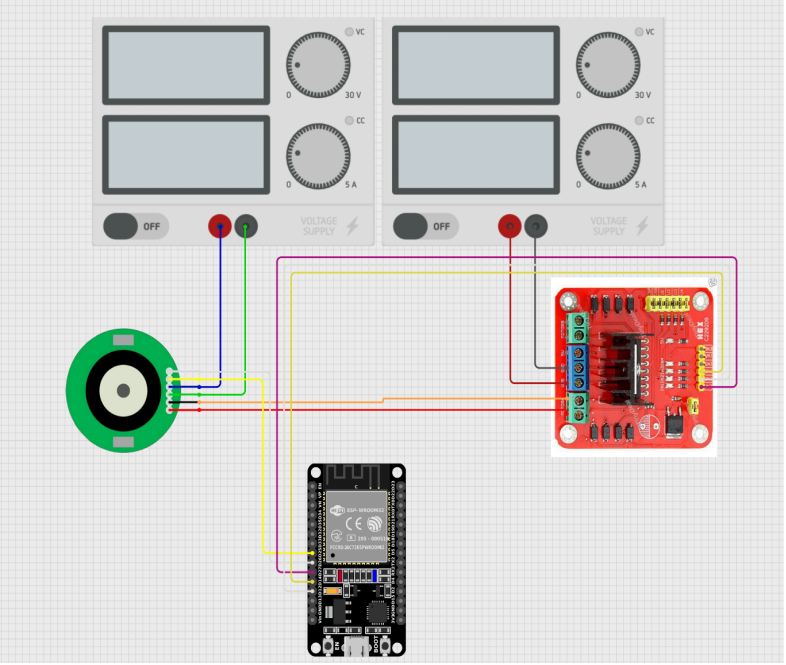

Figura 4. Diagrama de conexión

Para nuestro void setup() que es para establecer las primeras instrucciones inicializamos con la comunicación entre nuestro microcontrolador (Esp32) y el agente de ROS 2, para pasar en establecer el funcionamiento la designación de los pines en el microcontrolador, estos pines se agregara una attachInterrupt que nos ayudará en el cálculo de los valores que reciba del Encoder, después desarrollamos el nodo en su totalidad designando su nombre siendo mismo para los valores de los canales de suscripción y publicadores, agregamos un ejecutor que desarrollara en el que cada que tiene un dato nuevo ejecuta la función el callback.

Se establece también una función subscription_callbak que se ejecuta por cada mensaje des suscriptor , siendo la representación de nuestro mensaje recibido, ser interpretado como un puntero constante a un mensaje del tipo msg. Posteriormente llama la función PID que le

envía el dato nuevo del set_point al PID que se encargará de ajustar el control del motor usando como base lo que está recibiendo del set_point.

Definimos una función callback que ejecuta periódicamente la activación de un temporizador en nuestro entorno con ROS2, dentro de esta llamamos a la función pose que calculará la posición del robot y si el temporizador no es NULL, se le asigna un nuevo valor al set_point. Para después establecer el dato a través del pwm y finalizamos en gestionar los errores al publicar el mensaje.

Para la función de PID implementa las partes de proporcional, integral y derivativo para ajustar la velocidad del motor en función de un valor de referencia (set_point), Calcula el error con la diferencia el valor nuevo de set point y la velocidad actual del motor, esto lo recibe de la función pose, Luego, se calcula la derivada del error asumiendo un intervalo de muestreo. Luego la señal del control se obtiene sumando las componentes de Kp, Ki, Kd, estos valores fueron desarrollados con el método de Atmel aunado con el método comprobado en clase, Lo primero es modelar el circuito del motor presentado en la Figura 5.

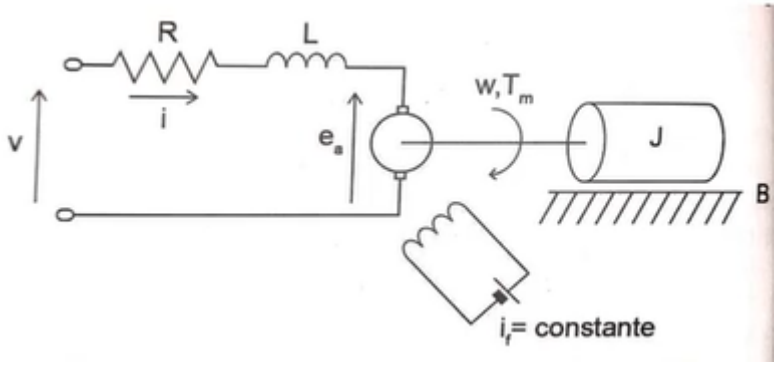

Figura 5. Circuito motor DC

En donde realizando un análisis podemos obtener la función de transferencia que se encuentra a continuación, de la cual sería reemplazar valores obtenidos en el datasheet y de manera heurística del motor.

$$\frac{km}{J*L\cdot s^2+(L*B+R*J)s+(R*B+kb*km)}$$

Los valores proporcionados por la hoja de datos (datasheet) del motor fueron los siguientes: Voltaje: 6v, Resistencia: 40 ohms, Km (Torque: 0.0134Nm y Corriente: 0.62 A): 0.0216 Nm/A , Kb (Voltaje: 6V/ W(s): 3500 RPM):0.0095 V/Rad*segundo. En cuanto a los datos faltantes como lo es la inercia, el coeficiente de fricción y la inductancia se obtuvieron con mediciones físicas utilizando aparatos para estas mismas mediciones de los cuales se ocupó un osciloscopio, voltímetro y un inductómetro. Una vez realizadas las mediciones se puede obtener valores con los que se pueden realizar los siguientes cálculos, para estos cálculos el Datasheet nos proporciono lo que es el peso del rotor que tiene un valor de 0.0002714346 Kg y el radio del rotor en metros que tiene el valor de: 0.003m.

$$J = 1/2(0.0002714346)(0.003s)^2$$

$$J = 1.22x10^{-9}$$

Una vez obtenido el valor de la inercia, podemos utilizarla para reemplazar el valor en la del coeficiente de fricción: lo que nos deja el cálculo de la siguiente manera.

$$B = J * 5.565s - 1$$

$$B = 1.22x10^{-9} * 5.565s - 1$$

$$B = 6.79x10^{-9} Nm/s/rad$$

Estos valores se reemplazan en el recuadro de la función de transferencia y eso sería todo para la misma.

Sistema de control

Para el armado del sistema de control en simulink desde matlab, realizamos un esquemático común para incorporar un sistema de control como se presenta en la Figura 6 donde el módulo con nombre de "PID controller", incorpora una función en la que automáticamente realiza una configuración de los valores del PID, tomando en cuenta sólo la función de transferencia y el resultado final de la gráfica.

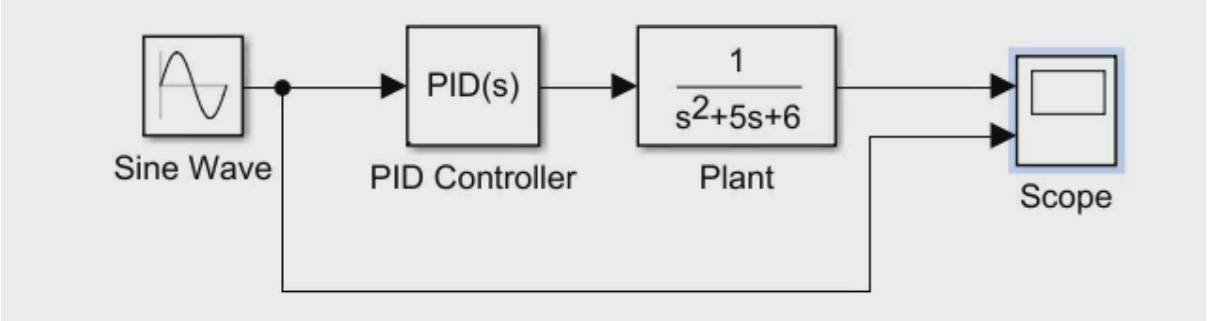

Figura 6. Sistema de control

Una vez obtenido ese resultado, lo siguiente es tomar los términos de PID e irlos configurando como se explica en el documento de ATMEL (APPLICATION NOTE, n.d., sec. Proportional, Integral, and Derivative Terms Together) y obtuvimos un esquema como el que se presenta en la Figura 7, donde explica que primero debe de irse moviendo la proporcional para que alcanzando un punto máximo pero no excedido ni tampoco tan bajo, ahora vayamos a configurar el integral para que no haya oscilaciones y al final el derivativo con el cual regulamos el sistema, aunque viene más explicado en el siguiente artículo de Atmel (AVR221: Discrete PID Controller on tinyAVR and megaAVR devices), donde también vienen otros métodos para la configuración y obtención del PID.

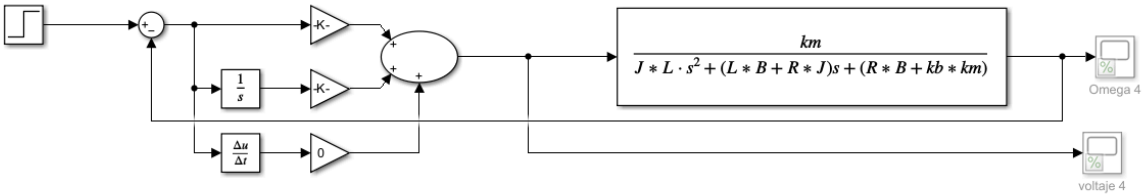

Figura 7. Ajuste de parámetros

Primero se configuró el PID en base al escalón unitario, debido a que este es una señal sencilla a diferencia de lo que es la señal senoidal y la cuadrática; entonces una vez que ajustamos los datos de la manera antes explicada, fuimos reemplazando las señales y para lo que es la senoidal, solo tuvimos que mover lo que es la integral donde aumentamos el valor en una unidad, mientras que para la señal cuadrática fuimos moviendo lo que es la derivativa, hasta alcanzar valores que para las tres señales fuera óptimo el seguimiento, obtenido resultados como los de las figuras (9,10 y 11), donde se puede apreciar como la línea de la señal (línea color morado), va siguiendo la línea del PID (línea color azul claro).

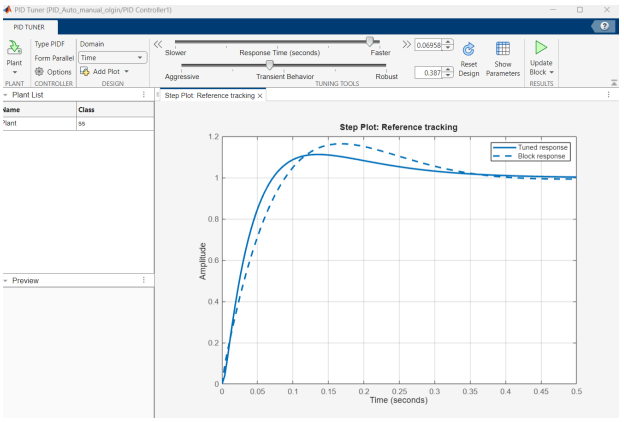

Figura 8. Ajuste de parámetros

La señal del control se limita con un contrain en base a la amplitud y el valor de pwm se es calculado con map ajustado en el rango de 0 a 255. Según la señal de control, se determinan las salidas de In1 e In2 para controlar la dirección del motor en avance, si la señal es positiva, o retroceso, si la señal es negativa. Si la señal termina cero se detiene, Finalmente el valor del PWM se envía al canal del control del motor, que se convertirá en /pwm_argos.

La función Encoder es una rutina de servicio de interrupción (ISR), lo que permite desde la RAM en la ESP32 para un desempeño más rápido,este leer los estados de los pines EncA y EncB siendo la salida del encoder como en la Figura 4, determina la dirección del sentido, si Aset BSet son el mismo valor indica un giro hacia la derecha, en contro caso de ser diferentes es a la izquierda, se guarda una determinación del tiempo en microsegundos.

La función pose calcula la posición, las revoluciones y la velocidad angular del motor a partir de los pulsos del encoder. Esto es básico para monitorear el movimiento del motor y así poder enviar la velocidad. Para evitar interferencias en la captura de datos, se desactivan temporalmente las interrupciones mientras se copian los valores actuales del contador de

pulsos, la dirección del giro y el tiempo en microsegundos. Luego se calcula la posición del motor multiplicando el número de pulsos por la resolución del encoder. Si el número de pulsos acumulados alcanza el valor máximo definido por el número de pulsos por revolución, se incrementa o decrementa el contador de revoluciones, dependiendo de la dirección del giro.

Posteriormente, se calcula la velocidad angular en radianes por segundo, dividiendo la cantidad de pulsos entre el tiempo transcurrido tal y como se demuestra con la siguiente fórmula.

$$motor\_speed = \frac{2\pi \times contador\_safe}{pulsos \times (\frac{\Delta t}{10^6})}$$

Finalmente, la velocidad que se encuentra en revoluciones por minuto (RPM) es publicada en el tópico /motor_output_argos, permitiendo su monitoreo en el sistema de ROS2.

Resultados

Como una conclusión al desarrollo de nuestro proyecto en este apartado se busca establecer las evidencias del funcionamiento de nuestro reto, con una especificación en los resultados obtenidos con el nodo Input y el Motor en base a un análisis de resultados , para exponer los puntos satisfactorios o por otro lado una interpretación a los resultados.

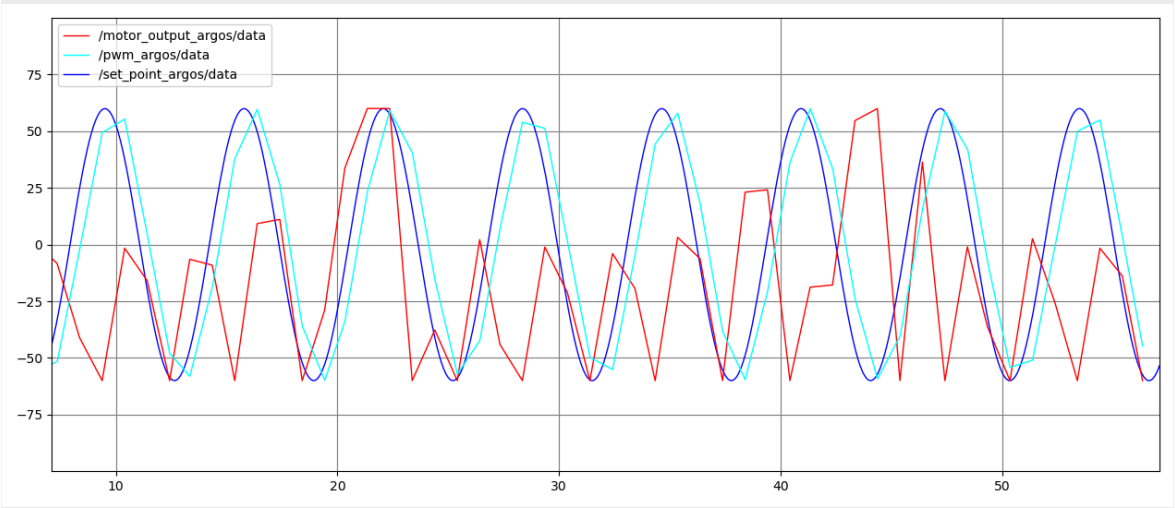

Figura 9. Onda senoidal

Cuando el parámetro de type está establecido en 1, la onda cambiará en senoidal, esto lo muestra el plot como en la Figura 9, que aquí se compara la estructura de del pwm, esté llevando el PID y se identifica como el azul claro, en este plot podemos observar la figura de la velocidad angular, identificada como la roja se muestra las elevaciones de nuestro motor DC, dichas elevaciones muestran cómo influye nuestra velocidad angular en la estructura de nuestro motor, teniendo repercusión en su forma errante al graficar.

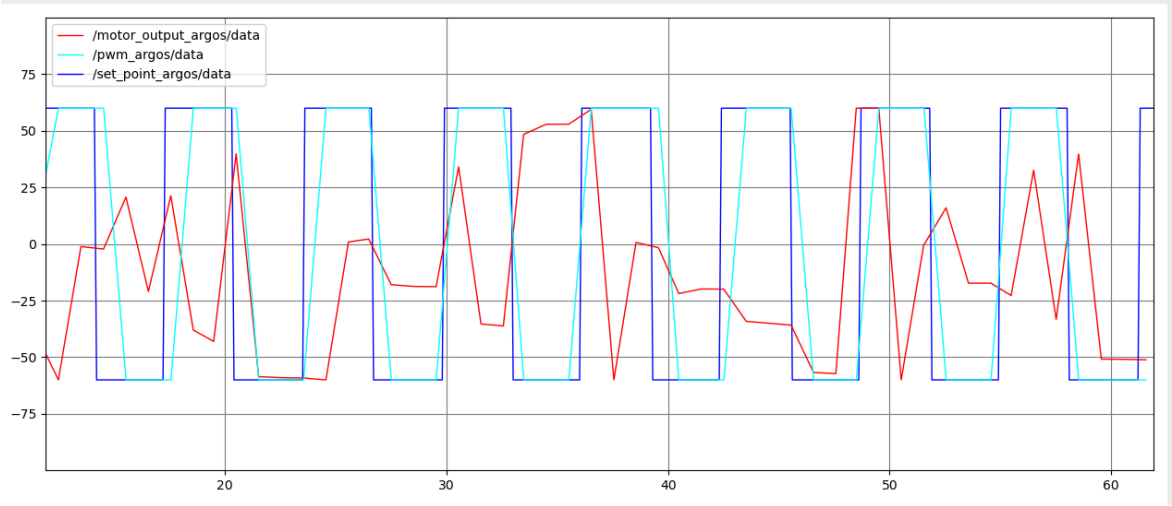

Figura 10. Onda cuadrática.

La Figura 10 muestra el caso cuando el valor 2 está definido en los parámetros de la onda, presentando pulsos rápido donde nuestro pwm se encarga de seguirlo, esto en la práctica muestra un cambio repentino de velocidad del motor haciendo que pase de un segundo el cambio de dirección en nuestro motor, como observamos la graficación de el motor_output se sigue mostrando pico , pero que van estando acorde a los cambios repentinos.

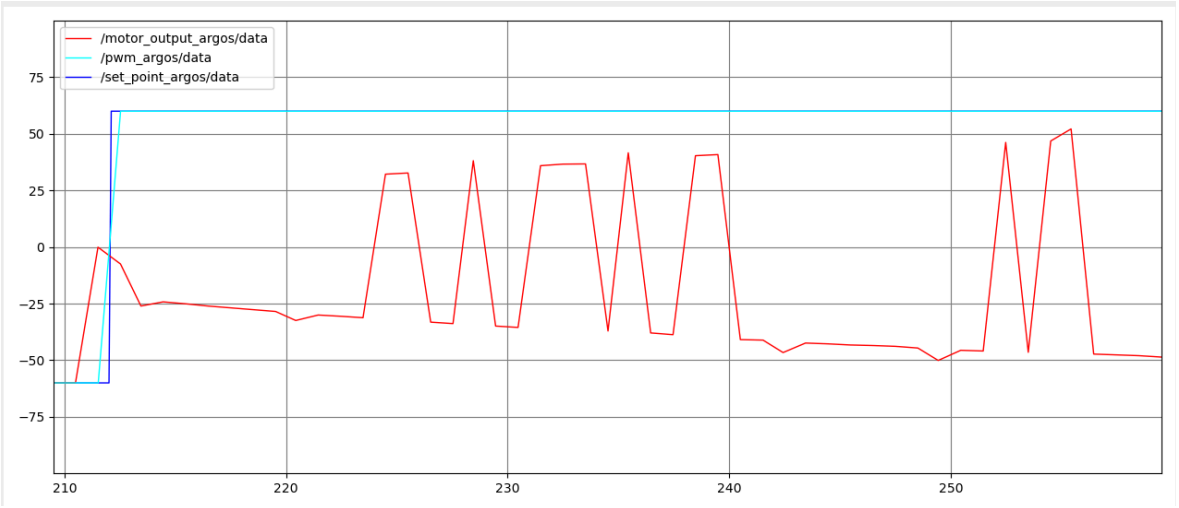

Figura 11. Escalón unitario.

En el escalón unitario se presenta como cambia de una manera perpetua el set_point a como se ve en la Figura 11 esto siguiendo este cambio el pwm, pero como podemos ver la forma del motor_output, va teniendo diversas variaciones que no se debería formar debido a que es perpetuo el movimiento, dando una posible muestra sobre que los datos que pasan de microro a ROS2 pueden tener fluctuaciones que cambian estos valores.

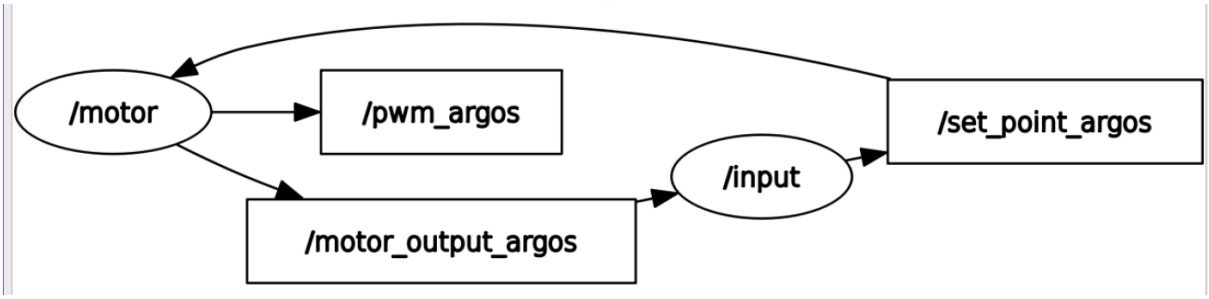

Figura 12. Gráfico de canales y nodos.

Como podemos observar en la Figura 12 los que están dentro de un cuadrados son los canales de comunicación y los círculos son los nodos, en el nodo de input, se muestra como publica el /set_point_argos y este mismo se suscribe al nodo motor, este nodo pública los valores de /pwm_argos que lo utilizaremos para graficar como se muestra en la Figura 9, como también regresa el /motor_output_argos al nodo de input para poder observar la velocidad angular como es visto en la Figura 13.

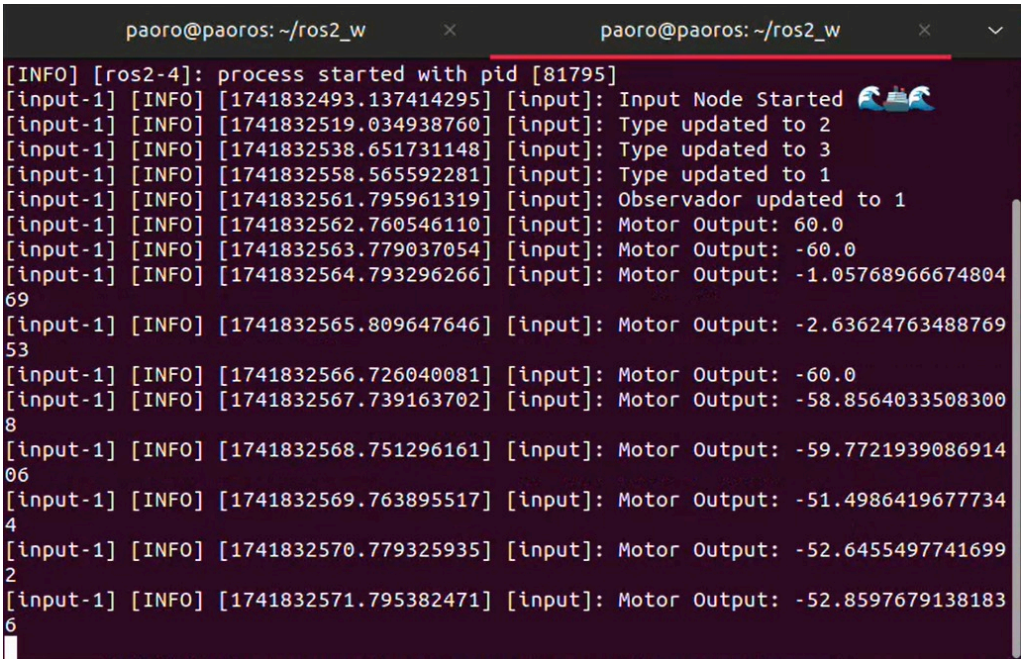

Figura 13. Observación de velocidades en terminal.

Como resultados, la terminal que se usó para correr el launcher, al modificar el valor del parámetro "type" a uno, empieza a imprimir el valor que actualmente se encuentra la velocidad, si la velocidad está en positivo el motor está girando a la derecha, si está en negativo está en un giro a la izquierda, los valores que se están mostrando más bajos significa que el pwm está bajando, significa que el voltaje del motor DC está disminuyendo consecuente hace su velocidad de caer.

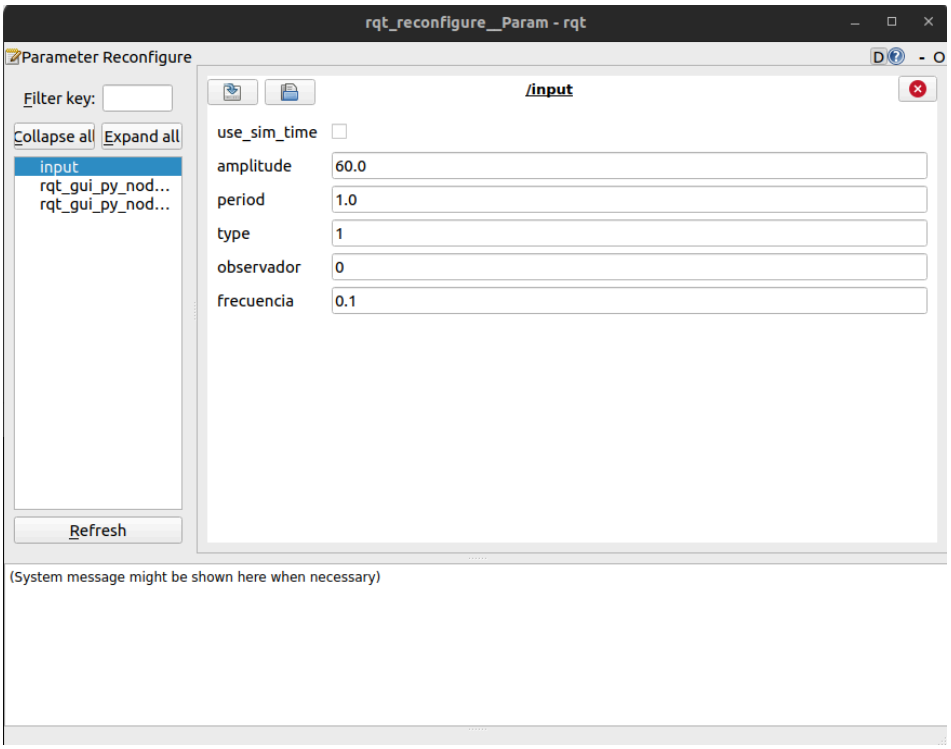

Figura 14. rqt_reconfigure.

En la FIgura 14, encontramos como se el rqt_reconfigure muestra los parámetros ya previamente establecido en ROS 2 para poder modificar la amplitud, periodo tipo, observador y frecuencia, designando al usuario una libertad para poder modificar las variables en tiempo real

Conclusiones

Por último, se presentan los logros alcanzados a lo largo del desarrollo del reto, analizando el grado de cumplimiento de los objetivos planteados. Se evalúa si estos fueron alcanzados en su totalidad, identificando las razones detrás de su éxito o, en caso contrario, los factores que pudieron haber limitado su cumplimiento. Asimismo, se proponen posibles mejoras a la metodología utilizada, con el propósito de optimizar los resultados obtenidos. A través de este análisis individual, se busca profundizar en la comprensión de los procesos involucrados y extraer aprendizajes clave para futuras implementaciones.

Emmanuel Lechuga

Con la implementación de este proyecto pudimos darnos cuenta que algunas cosas pudieron mejorar, al igual que algunas cosas no teníamos previstas como lo fue el fallo de uno de los motores, debido a que este no se podía comunicar de manera correcta con la Esp 32, lo que nos hace tener en cuenta cosas imprevistas como estas en un futuro. En cuanto a la implementación de todos los conocimientos pasados podemos decir que todo lo antes aprendido a lo largo de la carrera nos ha servido para poder aplicar muchos de esos conocimientos hoy con este proyecto. además de poder aprender otro lenguaje aunque es más como un espacio de trabajo bajo el sistema de Linux, pero especializado para la comunicación de robots, lo cual dentro de todo podemos darnos cuenta que el PID, los microcontroladores, la programación en arduino, en lenguajes de programación en Lenguaje máquina y prácticamente el uso de herramientas, nos ha ayudado en este proyecto para poder presentar una buena composición del proyecto. Aunque hubo un problema con el motor el proyecto se desarrolló de manera óptima.

Daniel Castillo

En el desarrollo de este proyecto, y en vista de la implementación de ROS 2 y micro-ROS fue una situación que presentó un reto con mayor desarrollo, mostrando cómo los temas de materias pasadas se van conectando con con un concepto más avanzado del modelado, usando herramientas como el control del PID y la comunicación entre un microcontrolador ESP 32. Pero es desarrollo de esto con nuevas herramientas trajo otra perspectiva de cómo se pueden implementar ciertas tecnologías en esto, como el estimar la velocidad a partir de los datos del encoder y utilizar un análisis a este para poder conseguir la inercia, el coeficiente de fricción y la inductancia, que son con herramientas con las que poco o casi nada he trabajado. Esto condujo una metodología que fue complicada de sobrellevar pero teniendo sus ventajas en un plazo corto siendo esto para un desarrollo ágil , permitiendo hacer pruebas constantes pero eso trajo desventajas en el desarrollo.

Los resultados obtenidos validan la eficacia del enfoque adaptado, mostrando como el PID implementado ofrece una respuesta estable a los diversos cambios, siendo bastante precisa pero ante distintas señales. estas implementaciones hacen que cuando el launch se active se

muestran de inmediato el plot, graph y rqt_reconfigure, nos muestra en su totalidad la comparación entre estos canales, como se había mencionado existe cierto conflicto en la gráfica del motor_output, que bien el funcionamiento del motor se adapta a lo que está pasando en el set_point deja algunas incongruencias sobre los picos que se presenta en el plot, esto muestra unos planteamientos como las señales que se envían de micro-ROS a ROS 2, sobre cómo el motor DC puede estar involucrado o como la metodología de trabajo pudo traer complicaciones al desviar la solución enfocándonos en la velocidad angular. Con esto concluyo que este proyecto logró proyectar dificultades que se pueden presentar en un área de trabajo donde los requerimientos de diversos temas es fundamental , en este momento es solo un motor pero la complejidad de estos elementos invita al análisis de diferentes estrategias para que como equipo podamos unificar diversas ideas y desarrollar un proyecto que pueda adaptarse con mejor eficacia a los diferentes cambios que se presentan , la posibilidad de explorar futuras mejoras, como un control adaptativo y la integración de estos son especiales para proyectos de este tipo.

Paola Rojas

Trabajar en este reto permitió la implementación de un sistema de control de velocidad para un motor DC basado en ROS2 y micro-ROS, logrando la comunicación entre una computadora externa y un microcontrolador para el control del motor en tiempo real. A lo largo de la actividad, se abordaron distintos temas, desde el modelado del motor hasta la implementación del PID, reforzando así conceptos fundamentales en la robótica.

Uno de los principales logros fue la integración de un control PID, la implementación de este controlador permitió la mejora de la respuesta dinámica del sistema, reduciendo el error y mejorando la estabilidad ante los distintos tipos de señales de referencia.

Entre los desafíos, se destaca la comunicación entre ROS2 y micro-Ros, la cual llegó a generar problemas a la hora del envío de datos. Otro reto fue la implementación de un sistema robusto para la captura de datos del encoder, específicamente la velocidad, la cual llegó a presentar fuertes variaciones.

Una posible mejora es el uso de filtros para eliminar el ruido presente en las mediciones del encoder y mejorar la estabilidad de la señal de la velocidad angular. Esto contribuiría a una mejor respuesta del controlador PID, reduciendo fluctuaciones no deseadas en el PWM y teniendo una mejor visualización de los resultados.

En conclusión, este proyecto no sólo permitió el desarrollo de un sistema de control PID para un motor DC, sino que también permitió conocer las posibilidades en la integración entre ROS2 y sistemas embebidos, sentando así los conocimientos fundamentales para la robótica.

Referencias

En este apartado se anexan los elementos consultados para el desarrollo del tema de investigación.

Fuentes, J. R. L. (2015). Desarrollo de Software ÁGIL: Extreme Programming y Scrum. IT Campus Academy.

Cerino Jiménez, R. (2025, 28 febrero). *Sensores y actuadores_sesión_6_PWM.pdf* [Diapositivas].

Cerino Jiménez, R. (2025, 4 marzo). *Sensores y actuadores_sesión_7_CONTADOR.pdf* [Diapositivas].

Cerino Jiménez, R. (2025, 7 marzo). *Sensores y actuadores_sesión_8_laboratorio_ENCODER.pdf* [Diapositivas].

ManchesterRoboticsLtd. (s. f.). *GitHub - ManchesterRoboticsLtd/TE3001B_Intelligent_Robotics_Implementation_2025: In this repository you will find all the resources required for the TE3001B*. GitHub. https://github.com/ManchesterRoboticsLtd/TE3001B_Intelligent_Robotics_Implementation_2025/tree/main

Micro-ROS. (s. f.). micro-ROS. https://micro.ros.org/

ROS Documentation. (s. f.). https://docs.ros.org/

APPLICATION NOTE. (n.d.). AVR221: Discrete PID Controller on tinyAVR and megaAVR devices. Microchip.com. Retrieved March 15, 2025, from https://ww1.microchip.com/downloads/en/Appnotes/Atmel-2558-Discrete-PID-Controller-on-tinyAVR-and-megaAVR_ApplicationNote_AVR221.pdf

Anexos

En esta sección se presentan materiales complementarios que respaldan la información del presente reporte. Se incluyen enlaces a los recursos digitales del proyecto, tales como el código fuente disponible en GitHub, donde se pueden encontrar los scripts y configuraciones utilizados, así como un enlace a YouTube con una demostración en vídeo del funcionamiento del sistema así como un vídeo explicativo. Estos anexos permiten una mejor comprensión del desarrollo y ejecución del proyecto sin sobrecargar el contenido principal del documento.

- El código del proyecto está disponible en :Repositorio de GitHub
- Para un mejor entendimiento del proyecto, ver el: Video explicativo
- El funcionamiento del proyecto se puede visualizar en: Video demostrativo