RETO FINAL
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.

El objetivo de este reto final es el desarrollo de un sistema de control PID para el control de la velocidad de un motor DC, que es controlado por una computadora externa, un microcontrolador, en nuestro caso un ESP-32 y un puente H. Retomando todos los conocimientos adquiridos previamente en ROS 2 y micro-ROS, para la creación de los nodos, tópicos y mensajes, todo esto con el fin de llevar satisfactoriamente el Reto. Para esto se creó un nodo `/Setpoint` que corre en nuestra computadora para la generación de la señal de entrada y un nodo `/Controller` que corre dentro del MCU, la cual recibe el duty cycley a su vez publica la velocidad estimada en radianes/segundo, así como la dirección de giro.

II. Objetivos.

- Usar parámetros para poder modificar las señales del setpoint en tiempo real.
- Aplicación de un PWM para poder controlar la velocidad de un motor.
- Realizar comunicación entre ros y micro-ros.
- Realizar la caracterización del motor DC.
- Creación de un control PID.
- Obtención de las constantes proporcional, integral y derivativa.
- Calcular la velocidad con ayuda de un encoder.

III. Introducción.

Después de haber realizado diversas actividades con ROS 2, motor en DC con encoder y recordando los conceptos de los sistemas de control, se ha podido realizar este reto final. Esta práctica se basa principalmente en poder controlar la velocidad de un motor en DC con un sistema de control de lazo cerrado. Para el buen desarrollo del mismo se ha realizado código en python el cual se encarga de publicar una señal, la cual será el duty cycle; al mismo tiempo se tendrá corriendo otro nodo en un ESP32 el cual será encargado de leer la señal otorgada por el código de python, esta señal será la salida deseada del sistema de control y la ESP32 será la encargada de usar el encoder del motor para obtener la velocidad real y así tener retroalimentación, del mismo modo será el responsable de hacer el PID para tener un mejor comportamiento. Probablemente, la implementación de esta actividad suena muy poco útil en la vida real, pero comprender bien el comportamiento y modelado de esta problemática será muy útil al momento de trabajar con cualquier robot, ya que se está utilizando uno de los sensores más importantes en la robótica que es el de la posición, este sensor es muy importante debido a que es muy sencillo obtener velocidades y aceleraciones con ayuda de las derivadas. Al momento de utilizar estas variables, será mucho más sencillo poder conocer las posiciones globales y locales de cualquier brazo de los robots, claramente con la ayuda de diferentes análisis de cinemática y dinámica de robots.

La metodología de trabajo para la obtención de los resultados propuestos se muestra a continuación:

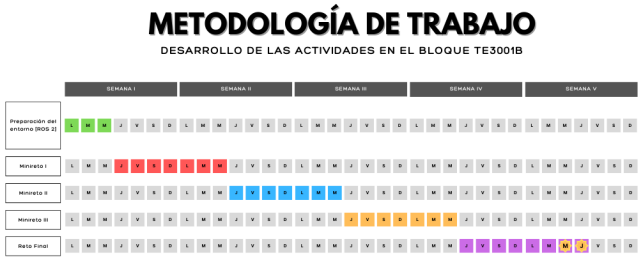

Fig. 1. Diagrama de Gantt sobre el desarrollo del curso. Se observan todos los pasos realizados y orden para el desarrollo del reto.

Nota: El diagrama en mejor resolución se encuentra en la sección de anexos.

En lo que respecta a la solución del reto final (semana 4 a 5), debido a la complejidad del mismo, sumado a el impedimento de no contar con la hoja de especificaciones de nuestro motor

(elemento que hubiese facilitado de sobremanera el problema) decidimos implementar un proceso de tipo heurístico para generar el modelo funcional del motor. A continuación se detalla el proceso generalizado en cinco puntos importantes para el desarrollo de la actividad, partiendo de la caracterización hasta el modelo ajustado del PID, demás está decir que esto es una generalización, pues cada punto requiere de más o menos elementos, mismos que serán contemplados texto abajo en este mismo documento.

Fig. 2. Pasos seguidos por el equipo para la resolución del reto [generalizado].

Pasos del proceso

- *Caracterización del motor:* La caracterización del motor es esencial para comprender su comportamiento. Se basa en medir la cantidad de pulsos por minuto que el motor genera, así como otros parámetros como el voltaje de entrada y el comportamiento general. Estos datos nos permiten comenzar a obtener variables que se aproximan al comportamiento real del motor. En nuestro caso, recurrimos a actividades previas de práctica en el entorno de Arduino con el ESP32 y el encoder para determinar estos datos. La caracterización nos ayuda a entender cómo responde el motor ante diferentes entradas y condiciones.
- *Función de Transferencia:* Una vez que tenemos los datos de caracterización, podemos derivar una función de transferencia que describa el comportamiento del motor. Esta función matemática relaciona la entrada (el set point) con la salida (como la velocidad angular del motor). Es fundamental para diseñar un controlador efectivo.
- *Desarrollo en Arduino:* Con la función de transferencia en mente, llega el momento de introducir MicroROS. Aquí desarrollamos el control para nuestra planta, considerando el set point (la referencia deseada) como entrada y la velocidad angular medida por el encoder como salida. La retroalimentación debería ser proporcionada por el encoder, lo que nos permite comparar la salida real con la deseada y ajustar el control en consecuencia
- *Coeficientes del PID:* Para implementar un controlador PID (Proporcional-Integral-Derivativo), necesitamos determinar los coeficientes que corresponden a las ganancias K para cada componente del controlador. Recurrimos al método de Ziegler-Nichols, que es una técnica empírica para aproximar estos coeficientes. Ajustamos los valores de K para lograr un buen rendimiento del controlador.
- *Modelo PID ajustado:* El paso anterior nos lleva al modelo de PID ajustado y funcional para la caracterización de nuestro motor. Este modelo nos permite predecir y controlar el comportamiento del motor en función de las entradas y las condiciones del sistema.

Los puntos anteriores pueden verse como un proceso iterativo de medición, modelado y ajuste para lograr un controlador eficiente y preciso.

Procedemos con la presentación de varios conceptos que fueron útiles para poder realizar esta actividad.

A. ¿Qué es ROS 2?

El Sistema Operativo de Robots (RO S, por sus siglas en inglés) es un conjunto de bibliotecas/librerías de software además de herramientas para construir aplicaciones de robots. Su primera versión, ROS 1, ha sido un elemento clave de un gran número de sistemas robóticos durante más de una década, década que trajo cambios significativos en el paradigma de la robótica y en el propio ROS, como resultado de ello surge ROS 2, una mejora de las limitaciones de ROS 1 y manteniendo lo funcional de dicha primera versión así como adaptándose a las necesidades crecientes de la industria misma. [1].

B. Estructura básica de funcionamiento.

La estructura de funcionamiento de ROS2 se puede sintetizar como se expresa a continuación:

- **Nodos:** Son la unidad más pequeña en el funcionamiento de ROS, cada nodo es un programa con tareas específicas.
- **DDS Middleware:** ROS 2 emplea Data Distribution Service como middleware para comunicarse entre los nodos, permitiendo su comunicación.
- **Estructura de grafo:** La estructura nodo, tema, mensaje y descubrimiento es la estructura distribuida de ROS 2.
- **Capa de abstracción RMW:** Usa una capa de abstracción propia en lugar de un middleware.
- **Publicadores y Suscriptores:** Los nodos de ROS son capaces de publicar mensajes de un tema o suscribirse a estos y recibir mensajes.
- **Herramientas y Bibliotecas:** ROS 2 incluye bibliotecas de código abierto empleadas para diferentes propósitos dentro del robot.

C. Elementos que integran su red de comunicación.

*1) **A nivel de sistema de archivos:**:* ROS se compone de paquetes que son la unidad principal para organizar el software. Los metapaquetes representan un grupo de paquetes relacionados. Los manifiestos de paquete proporcionan metadatos sobre un paquete. Los repositorios son colecciones de paquetes que comparten un sistema VCS común. ROS utiliza tipos de mensajes y tipos de servicio para definir las estructuras de datos para los mensajes y servicios respectivamente.

*2) **A nivel gráfico de cálculo:**:* ROS se compone de nodos que son procesos que realizan cálculos. El Maestro ROS proporciona registro de nombres y búsqueda para la comunicación entre nodos. El servidor de parámetros permite almacenar datos en una ubicación central. Los nodos se

comunican entre sí pasando mensajes a través de un sistema de transporte con semántica de publicación/suscripción, conocido como temas. Para las interacciones de solicitud/respuesta, se utilizan servicios, que están definidos por un par de estructuras de mensajes. Finalmente, las bolsas son un formato para guardar y reproducir datos de mensajes ROS.

3) ***A nivel de comunidad:****:* Las distribuciones ROS son colecciones versionadas de pilas que facilitan la instalación de software y mantienen la consistencia de las versiones, similar a las distribuciones de Linux. ROS opera en una red federada de repositorios de código, permitiendo a diferentes instituciones desarrollar y lanzar sus propios componentes de software robótico. [3].

D. ¿Qué es Namespaces?

Namespace es un mecanismo que nos permitirá iniciar dos nodos similares sin incurrir en conflictos de nombre de nodo o de tema, con la condición de compartir un mismo prefijo, por ejemplo, para dos nodos, sensor1 y sensor2, si creamos un namespace "robot", el nombre de estos nodos se escribiría tal que *robot1/sensor1* y *robot1/sensor2* respectivamente.

1) Uso de namespaces: En general, como ya se mencionó anteriormente, esta funcionalidad es de gran utilidad para evitar conflictos de nombre entre los nodos, así mismo, puede emplearse para organizar nodos con relaciones lógicas.

2) Declaración de Namespaces: Para especificar el namespace, se puede crear desde el nodo en python de la siguiente forma:

```
class MyNode(Node):
    def __init__(self):
        super().__init__('my_node', namespace='my_namespace')
```

3) Visualizarlos: Con el comando en terminal *ros 2 node list* para ver los nodos enlazados en el mismo. [1].

4) Namespaces y múltiples nodos: Si bien definir namespaces para un pequeña cantidad de nodos puede que no implique una gran tarea, para grandes cantidades existe una acción que nos permite definir un namespaces global para cada descripción de archivo, donde cada nodo anidado heredará estas características.

E. ¿Qué es Parameters?

Un parameter puede verse como un valores de configuración de nodos los cuales pueden almacenar parámetros como *bool, int64, float64, string, byte[], bool[], int64[], float64[] o string[]*. En general, son usados al inicio para la configuración de nodos y durante el tiempo de ejecución sin modificar el código, en este sentido, el parámetro tiene la misma vida útil que el nodo. Su estructura general consta de una clave, un valor y un descriptor.

1) Declaración de parámetros: El nodo necesitará declarar todos los parámetros que aceptará desde el inicio, esto para reducir posibles errores más adelante. Esto dependerá del tipo de nodo, pues en algunos casos se pueden crear instancias que permiten configurar y obtener parámetros.

2) Comportamiento de los parámetros: Cada parámetro puede comportarse de diferente forma, de forma predefinida necesitará declararse desde el inicio del nodo, sin aceptar modificaciones durante la ejecución, esto con el fin de evitar el ingreso de un valor diferente al tipo de valores que puede recibir el parámetro. Sin embargo, si el código lo permite y es necesario, podemos declarar el parámetro como uno dinámico, siendo este el segundo caso.

3) Establecer valores:

- **Al ejecutar un nodo:** Se puede hacer mediante la línea de comandos individuales o mediante archivos YAML.
- **Al lanzar un nodo:** Se puede modificar a través de la función de lanzamiento de ROS 2.

4) Interactuar con parameters:

- `/node_name/describe_parameters`: devuelve una lista de descriptores asociadas a los parámetros.
- `/node_name/get_parameter_types`: a partir de una lista de nombres de parámetros, devuelve una lista con los tipos de parámetros asociados a estos.
- `/node_name/get_parameters`: a partir de una lista opcional de prefijos, devuelve los parámetros disponibles con ese prefijo.
- `/node_name/set_parameters`: dada una lista con nombres y valores de parámetros, intenta establecer parámetros en el nodo. Devuelve una lista del resultado de esta acción (puede ser exitosa o no). [2]
- `/node_name/set_parameters_atomically`: dada una lista con nombres y valores de parámetros, intenta establecer parámetros en el nodo. Devuelve un único resultado, por lo que si uno falla, todo falla. [2]
- `ros2 param`: es el comando que comúnmente se utiliza para interactuar con los nodos que se encuentran en ejecución.

5) Devoluciones de llamada:

- **Establecer parámetro:** Su objetivo es brindar al usuario la posibilidad de inspeccionar el próximo cambio de parámetros y rechazarlo.
- **Evento de parámetro:** Su objetivo principal es brindarle al usuario la capacidad de reacción a los parámetros que han sido aceptados.

F. ¿Qué son los Custom messages?

Los mensajes personalizados son mensajes que el usuario define, los cuales le permiten extender el conjunto de tipos de datos admitidos por ROS 2, permitiendo la transmisión de información específica entre nodos fuera de los tipos de mensaje admitidos.

1) Mensajes incorporados: Los custom messages no solo permiten crear nuevas definiciones de mensaje, también pueden modificar los incorporados con nuevas definiciones.

2) Recomendaciones para la creación de mensajes personalizados: Siempre es importante verificar si existe un mensaje incorporado que puedas emplear antes de crear un mensaje personalizado, y en caso de existir coincidencia, considera si es posible emplearlo de forma directa o si es necesario adaptarlo. [6]

G. ¿Qué es Micro-Ros?

Se trata de una extensión de ROS centrada en microcontroladores, se puede ver como una conexión entre las unidades de procesamiento de recurso limitado (microprocesadores) y los sistemas más grandes para su uso en el campo de la robótica, esto se traduce en la posibilidad de programar un robot con limitadas capacidades de hardware sin mayor inconveniente.

1) Características:

- **Optimizada para microcontroladores:** permite utilizar los conceptos más importantes y útiles de ROS en microcontroladores.
- **Middleware:** Aunque se encuentra de manera limitada, este es extremadamente flexible, perfecto para micros.
- **Soporte multi-RTOS:** permite el uso de varios sistemas operativos en tiempo real. [1]

2) Importancia de Micro-Ros: Micro-Ros ha permitido construir robots pequeños que resultan ser mucho más económicos frente a las opciones del mercado sin limitar al robot de las funcionalidades que ROS ofrece. Particularmente esta necesidad de aplicación en el área de la robótica viene muy bien en el desarrollo de drones, robots móviles y el control industrial. El uso de este recurso nos permite el desarrollo de estos y más proyectos sin necesidad de restricciones de memoria o de procesamiento. [7]

H. ¿Qué son los módulos ADC?

Los módulos ADC (Analog-to-Digital Converter) puede verse como uno de los componentes en electrónica y en el área de la robótica más importantes para la construcción de módulos o robots. El propósito fundamental de un ADC es convertir señales analógicas, caracterizadas por poder adquirir valores distintos en un rango de valores continuos, en señales digitales permitiendo así poder ser representadas en forma de números discretos. Esta conversión facilita al microprocesador, microcontrolador o cualquier dispositivo digital que lo use, el procesamiento y uso de estas señales.

1) Funcionamiento: El funcionamiento del módulo ADC puede verse de la siguiente manera: el ADC observa una tensión analógica producida por algún sensor, a lo cual este módulo deberá convertir dicha tensión en un código binario para un tiempo determinado, esta emplea el muestreo sobre la tensión analógica en un tiempo x, y determinará el valor binario que deberá tener como salida. Algo importante a mencionar es que la frecuencia de muestreo, es independiente en cada dispositivo ADC, por lo que deberá consultarse su hoja de especificaciones.

2) Tipos de arquitecturas:

- **Registro de Aproximaciones Sucesivas o SAR:** para determinar un valor digital aplicable a ese valor analógico se realiza una serie de aproximaciones.
- **Delta-Sigma:** hace uso de técnicas de sobremuestreo y filtrado, esto aproxima mejor sus valores, lo que se traduce en una mayor precisión.
- **Convertidor de Canalización:** se divide a la señal en ”canales” los cuales se analizan por separado para determinar el valor.

3) Importancia: Los módulos ADC son altamente importantes en el uso de sensores con parámetros de salida análogos, pues facilita la lectura de los mismos y reduce su complejidad, aumentando el rango de uso en equipos. Así mismo, es una puerta entre el mundo analógico y el digital, permitiendo el diseño de dispositivos capaces de emplear ambos sin complicaciones. [8]

I. ESP32

Es la denominación para un Sistema en Chip de bajo costo y consumo de energía, el cual cuenta con tecnología WiFi y Bluetooth. Tiene un módulo WIFI+BT+BLE para sensores de baja potencia, una corriente de reposo de 5uA y velocidad de 150 Mbps. [9] Esta placa cuenta con el siguiente pinout:

- 19 canales de convertidor analógico a digital.
- 3 interfaces SPI (transferencia y comunicación).
- 3 interfaces UART (transferencia y comunicación).
- 2 interfaces I2C (transferencia y comunicación).
- 16 canales de salida PWM.
- 2 convertidores de digital a analógico.
- 2 interfaces I2S (transferencia y comunicación).
- 10 GPIO de detección capacitiva.

J. PWM en ESP32

El PWM (Pulse Width Modulation) permite el control de la intensidad de la corriente que fluye en un dispositivo electrónico a través de la variación de la duración en los pulsos de una señal de tipo digital. En microcontroladores como el ESP32, es comúnmente empleada para controlar la velocidad en los motores. El ESP32 cuenta con múltiples pines que admiten el uso de PWM, lo cual permite el control de diferentes dispositivos mediante estos. Sin embargo, a diferencia de lo que hace Arduino, el ESP32 emplea modulación ledc para generar salidas PWM, si bien ldc es utilizada comunmente para la intensidad de LEDs, se puede emplear para generar ondas, empleadas como generadores de PWM.

1) Configurar PWM en el ESP32: En una primera instancia deberemos configurar una serie de parámetros empleando la función `ledcSetup()`:

- Canal PWM entre 0 a 15.
- Pin al que el dispositivo esta conectado.
- Frecuencia de modulación en Hz.
- Resolución en bits de 1 a 15.

Una vez estos sean configurados, se establece el ciclo de trabajo, el cual nos pemitirá controlar la intensidad de la señal de salida, podremos configurar este parámetro empleando la función `ledcWrite()`. El ciclo de trabajo puede recibir un valor entre 0 (que significa que el pin siempre estará en estado ”bajo”) y 255 (el cual contrario al 0, implica que su estado siempre estará en ”alto”). [10]

K. Motor en DC.

Estos motores de corriente continua son muy útiles debido a la suavidad a la hora de controlarlos. Algo que de la misma manera es muy bueno es su torque a la inercia del rotor; por lo tanto, responderán muy rápido a los cambios.

En este caso se ha utilizado un motor con imán permanente. Como su nombre lo dice, sus campos son generados gracias a los imanes; lo que sucede es que, cuando la corriente eléctrica circula por la bobina del rotor, se genera un campo electromagnético que interactúa con el campo magnético del imán. Se generará un giro cuando dos polos iguales estén juntos; por lo tanto, empezará a girar y así seguirá hasta que se corte la alimentación.

L. Puente H

Se trata de un tipo de circuito electrónico empleado en el control de motores DC. El objetivo de este dispositivo es el de permitirle al motor girar en ambas direcciones, esto es posible mediante conmutadores (mecánicos como relés o electrónicos como transistores). En líneas generales, el diseño de un puente H presenta lo siguiente:

- **Interruptores:** generalmente son 4, pueden ser transistores, relés o MOSFETs. Estos permiten la conmutación de la corriente.
- **Motor:** Es el dispositivo al cual se le aplica en control.
- **Fuente de alimentación:** Proporciona la energía para el funcionamiento del motor.
- **Circuitos de control:** Proporcionan las señales a los interruptores de dirección.

1) Funcionamiento: Su funcionamiento se basa enteramente en como se activan o desactivan los interruptores, a continuación se mencionan 3 de ellas:

- Giro hacia adelante: Se activan los interruptores diagonales (superior izquierdo e inferior derecho), los demás aparecen desactivados.
- Giro hacia atrás: Se activan los interruptores opuestos a los del giro hacia adelante (superior derecho e inferior izquierdo).
- Detener el motor: Será posible al desactivar los 4 interruptores.

2) Ventajas y consideraciones:

- Flexibilidad de control: control bidireccional en un solo circuito.
- Compacto: puede diseñarse en un tamaño reducido.
- Eficiente: minimiza la perdida de energía.
- Protección contra cortocircuitos: evitar activar interruptores opuestos al mismo tiempo.
- Disipación de calor: dependiendo de la corriente, nuestro puente puede o no necesitar disipadores.

[11]

M. Encoder

Dispositivo de detección que proporciona una señal como respuesta, convierte el movimiento en esta señal eléctrica, empleada para determinar la posición, como un contador de velocidad o incluso de dirección. Emplea diferentes tipos de tecnología para crear la señal, aunque la óptica es la más común. [5]

1) Ejemplos de encoders:

- **Aplicación de corte a medida:** la combinación de un encoder en una rueda le otorga control sobre cuanto material debe cortar.
- **Observatorio:** un encoder puede proporcionar la posición de un espejo móvil para su posicionamiento.
- **Gruas:** un encoder proporciona información del posicionamiento, por lo que sabrá cuando recoger o soltar.
- **Ascensor:** le permiten saber al ascensor en que piso están y si ha llegado o no al correcto.

2) ¿Incrementales o absolutos?: Los encoders podría producir señales incrementales o absolutas. Las señales incrementales no indican posiciones exactas, solo el cambio en la posición. En cambio, los absolutos emplean un valor diferente para cada posición. [12]

N. Encoder de Cuadratura

Se trata de un tipo de encoder rotativo de tipo incremental el cual puede indicar tanto la posición, la dirección y la velocidad del movimiento.

1) Funcionamiento: Este tipo de encoder genera dos señales de onda cuadrada con un desfase de 90 grados entre ellas. Para la detección de cambios en la placa codificada hace uso del principio del efecto Hall, es decir, se genera tensión al detectar un campo magnético, indicando así al encoder una lectura. Dicha lectura no es sino una señal digital única para cada movimiento. Estas señales nos permiten determinar cuantos grados o distancia se ha avanzado.

2) Partes del encoder de cuadratura:

- **Placa codificada metálica o de plástico:** presenta perforaciones a lo largo de su extensión, las cuales indican el movimiento.
- **Sistema de detección:** emplea emisores y receptores para detectar la codificación de las placas.
- **Circuito de control:** lee las señales generadas y las convierte en desplazamiento.
- **Carcaza:** aunque esta puede no estar presente, le permite al encoder proteger sus componentes.

3) Tipos de encoder:

- **Giratorio:** permite medir el giro y la velocidad angular de los ejes giratorios.
- **Lineal:** Mide la distancia recorrida de un objeto sobre un riel.

[13]

O. Resolución de un encoder

La resolución de un encoder es medida en pulsos por revolución (PPR), dicho parámetro determina cuantos pulsos eléctricos se emiten por cada giro completo del eje del encoder. La forma de medirla dependerá del tipo de encoder, por ejemplo:

- **Encoders incrementales:** su resolución está directamente relacionada con el número de impulsos por vuelta.
- **Encoders absolutos:** su resolución está definida por el número de bits por mensaje de salida. Como cada uno es

distinto, su resolución está directamente relacionada con este número de bits.

[14]

P. Cálculo de la posición con Encoders

Para poder calcular la posición empleando un encoder de tipo incremental es requisito fundamental poder leer la tensión del encoder y transformarla a su equivalente en pulsos, estos pulsos representan un número de grados en un diagrama polar, por lo que a partir de ellos es posible estimar su posición. En un encoder de tipo magnético, los sensores de efecto Hall permiten detectar la orientación de un imán fijo, el cual será utilizado por un microprocesador para estimar la posición. En cuanto a los encoders lineales encontraremos un cambio en la forma de medición, pues en lugar de medir movimientos rotacionales, medirán movimiento lineal mediante una cinta magnética y un cabezal magnético que permitirá la lectura del desplazamiento lineal, esto en el caso de los magnéticos, mientras que los ópticos emplearán una regla graduada y un cabezal de lectura para determinar este movimiento. [15]

Q. Estimación de velocidad mediante el método de Euler

Se trata de un procesamiento de tipo numérico empleado para encontrar soluciones aproximadas en ecuaciones diferenciales de orden uno, esto claro siempre que se conozca la condición inicial del sistema. Si conocemos la velocidad (la cual es conocida como la razón de cambio de posición con respecto al tiempo), podemos usar el método de Euler para estimar la posición en intervalos discretos de tiempo.

1) ¿En qué consiste?:

- Con el método de Euler se pretende encontrar una solución numérica en el intervalo de x_0 y x_f.
- La discretización comprende un intervalo de $n+1$ puntos, es decir, $x_0, x_1, x_2...x_n$ donde $x_i = x_0 + ih$ siendo h el paso entre intervalos.
- Es inherente a conocer la condición inicial, conocer la derivada del inicio $y'(x_0) = f(x_0, y_0)$.
- Se hace el cálculo para aproximar un valor a la función en el siguiente punto.
- El procedimiento se repite con siguientes puntos hasta llegar al final.

[16]

R. Interrupciones en el ESP32

Las interrupciones en el ESP32 son una herramienta de hardware que, en general, permite que el microcontrolador procese eventos cortos y responda a estos de manera inmediata y eficiente. Otro uso común es para evitar la revisión constante del estado de los sensores. Básicamente, al ocurrir uno de estos ”eventos”, el programa principal se detendrá momentáneamente y se ejecutará mediante una función denominada *Callback*.

1) Implementación de interrupciones: **Definir la función Callback**

- La función se ejecuta cuando ocurra interrupciones en un pin.
- Contiene un tipo de dato *void* y no recibe argumentos.
- Se debe etiquetar con el atributo `IRAM_ATTTR`para ubicarlo en la RAM interna del ESP32, lo que le permite hacerle un proceso rápido.

Configurar la interrupción

- Haciendo uso de *attachInterrupt()* podremos configurar la interrupción en el pin.
- Necesita de tres argumentos: el número del pin, la función callback y su modo de interrupción.
- **Acerca de los modos:** *RISING* se activa cuando el estado del pin pasa de bajo a alto.
- **Acerca de los modos:** *FALLING* se activa cuando el estado del pin pasa de alto a bajo.
- **Acerca de los modos:** *CHANGE* se activa en cualquiera de los dos.

[17]

IV. Solución del problema

A. Implementación de ROS

En pocas palabras se ha creado un nodo que correrá desde la computadora, el lenguaje principal de este será python. A el nodo se le llamará `riSetpoint` (se le puso ”ri” por la abreviación de roboinges que es el nombre del equipo y así no tener transferencia con los datos de los otros equipos); como su nombre lo dice, este nodo será capaz de mandar el valor deseado de salida de nuestro sistema por medio del tópico `ri_setpoint`. De la misma manera por métodos prácticos se han declarado parámetros para poder modificar el tipo y valores de las señales en tiempo real y así observar el comportamiento del sistema en diferentes casos. Por último se ha creado un launch file, todo esto para que sea más sencillo correr todo más rápido y con un click. A continuación se muestra más a detalle todos los componentes de la implementación en ros.

1) Nodo que manda setpoint.: Este código es el más importante en la implementación de ros en python. Lo primero que se hace es declarar tanto los nodos, parámetros, tópicos y timer para declarar frecuencia de publicación. En este caso se ha declarado una frecuencia de 10Hz ya que si esta era mayor, la lectura de micro-ros en cada nuevo dato mandaba mucha basura, y tomando en cuenta que la frecuencia de nuestras señales eran de uno, según el teorema de muestreo de nyquist, debe ser mínimo dos veces la frecuencia de la señal, pero en nuestro caso para estar seguros y como el hardware lo permitió tomamos una frecuencia diez veces mayor, por lo tanto los 10Hz se han encontrado como la frecuencia adecuada, en el caso de las frecuencia en micro-ros se ha hecho de la misma forma diez veces mayor, para estar seguros que la lectura de datos que mandara el nodo creado por python sea buena. Para declarar los parámetros se han declarado 4 tipos diferentes, donde una es una señal constante la cual se le puede modificar

la amplitud y las otras 3 son señales de seno, pulso cuadrado y diente de sierra, las cuáles se les podrá modificar la amplitud, frecuencia, offset y cambio de fase; por lo tanto se debe de declarar un parámetro para cada variable. Por último queda comentar que se ha decido usar el tipo de dato float32 para toda la comunicación de los tópicos por diversas razones, la primera es la eficiencia, ya que se están trabajando muchos datos, no sería buena idea usar un float64, debido a que tomaría muchos recursos y en este caso no es necesaria la doble precisión. La segunda razón es que según la documentación de ros existe una buena compatibilidad con float32 al momento de usar `rqt_plot`. En pocas palabras este tipo de dato podrá brindar un gran desarrollo debido a su precisión suficiente y la compatibilidad con los recursos de ros 2.

```
Método __init__():|
 Inicializar nodo "ri_Setpoint"
 Declarar parámetros para controlar
  señales
 Crear publicador "ri_setpoint"
 Crear temporizador que llama al método
 timer_callback() cada 0.1 segundos
```

Después de esto se debe de realizar el callback el cual será llamado cada 0.1 segundos y mandará la señal correspondiente del valor de los parámetros del tipo de señal que se encuentre en ese momento.

```
Método timer_callback():
 Obtener el tiempo actual
 Leer el parámetro ri_type
 Generar la señal en función del tipo
 de señal
  especificado
 Leer parámetros solamente de tipo
 señal actual
 Publicarseñal en tópico "ri_setpoint"
```

Lo único que queda por hacer es crear una instancia para ros en la función main para que se pueda correr el nodo creado.

```
Función main():
 Inicializar el entorno de ROS
 Crear una instancia de la clase
 My_Talker_Params
 Entrar en el bucle de eventos de ROS
 Liberar recursos al salir del bucle de
 eventos
```

2) Config file.: Antes que nada se tuvo que crear una nueva carpeta a la altura de src del motor, dentro de esa carpeta crear un archivo con terminación yaml.

```
mi_paquete/
 config/
    params.yaml
 src/
    mi_paquete/
        __init__.py
        gener.py
 package.xml
 setup.py
```

Este archivo será el encargado de otorgar el valor inicial a nuestros parámetros, debe de llevar el nombre del nodo al que se declararañ y después asignar los valores de cada parámetros.

```
NombreNodo:
 rosParameters:
  type:
  sine:
    param1 a n
  square:
    param1 a n
  sawtooth:
    param1 a n
  constant:
    amplitude
```

3) Launch file.: El último código creado fue el encargado de lanzar los códigos creados de una manera sencilla y de la misma manera abrirá el `rqt_plot` y el `rqt_graph`, todo eso solamente con una línea de comando, todo esto se ha aplicado más que nada por practicidad, ya que agiliza la manera de correr el código y poder observar su comportamiento. Este nuevo archivo se debe de crear dentro de una nueva carpeta llamada launch y el archivo debe de tener la terminación launch.py para su funcionamiento correcto. El workspace se debe de ver algo así.

```
mi_paquete/
 config/
    params.yaml
 launch/
    init_launch.py
 src/
    mi_paquete/
        __init__.py
        gener.py
 package.xml
 setup.py
```

Con esta configuración lo único que se debe de hacer es llamar a un launch todos los archivos que se deseen correr y de la misma manera los plots y graph si es que se desea.

4) Modificación de package.xml y setup.py: Cabe destacar que para que el paquete funcione de manera correcta con los launch y config file se deben de agregar el ejecutable de ros2launch en el xml e importar los data files de carpetas nuevas con ayuda de importar os.

Después de haber realizado todo lo antes mencionado se pudo dar como concluido el código de python, ya que podrá mandar de manera exitosa la señal deseada con los parámetros otorgados.

Si se desea ver el código creado, se puede consultar en el repo del equipo [20]

B. Diseño del control PID

Para el control de nuestro sistema, escogimos un PID, debido a que el modo proporcional nos ayudó a tener una respuesta rápida, el integral nos permitió reducir el error para ser más precisos y el modo derivativo ayudó a mejorar la estabilidad del sistema al prevenir oscilaciones no deseadas. Haciendo uso del control PID en este caso fue de gran utilidad para proporcionar una respuesta más precisa, estable y adaptable a una variedad de sistemas y condiciones operativas, lo cual puede ser crucial en sistemas sensibles.

1) Caracterización del motor: Para obtener el control PID de un motor DC se puede hacer por medio de una función de transferencia o con ayuda de la caracterización del motor, pero para la función de transferencia se necesita la hoja de especificaciones del motor, desafortunadamente no contamos con las especificaciones, por lo que optamos por hacer la caracterización de motor y obtener una ecuación polinomial que realice el modelo del motor. Para obtener la modelación del motor dc, hicimos pruebas graficando el PWM contra las revoluciones por minuto (RPM), para estas pruebas usamos un voltaje de 8 volts en una fuente de voltaje. Primero notamos que el motor empezaba a moverse a partir de un PWM de 152 en una escala de 0 a 255, por lo que las pruebas fueron desde ese PWM hasta 255 y las revoluciones por minuto fueron desde 1 hasta 14, como se puede ver en la tabla I

TABLE I
DATOS PARA LA CARACTERIZACIÓN DEL MOTOR DC.

PWM	RPM
153	1
165.75	3
178.5	4.75
191.25	6.65
204	8.5
216.75	10.4
229.5	12.3
242.25	13.2
255	14

Las mediciones no fueron del todo exactas, debido a que las revoluciones por minuto calculadas las obteníamos en enteros y los decimales los estimamos. A continuación se muestra la gráfica de los datos, así como la línea de tendencia, como se puede ver en la figura 3

La línea de tendencia que mejor se adaptaba al modelo fue la polinómica, con la cual obtuvimos la siguiente ecuación que modelaba la velocidad angular del motor.

$$y = -9x10^{-6}x^3 + 0.0049x^2 - 0.7601x + 34.24 \quad (1)$$

2) Obtención de las constantes PID por medio del método de Ziegler-Nichols: A partir de los datos obtenidos en la sección anterior comenzamos a buscar los valores de las constantes del PID por medio del método Ziegler-Nichols. Para este método lo primero que se necesita es tener todo el sistema y poner los valores de la integral y derivativa en 0, de ahí modificar la variable proporcional de tal manera que las oscilaciones del sistema fueron aumentando hasta que las

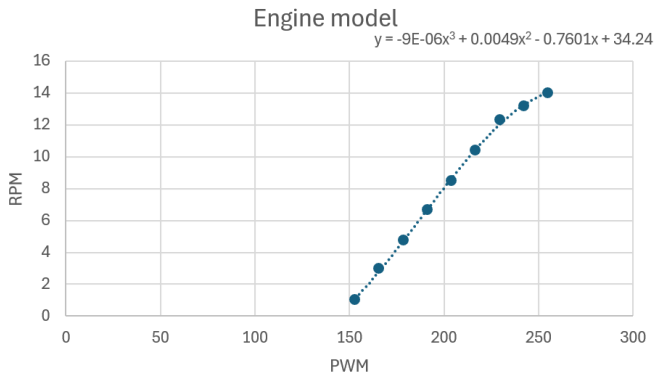

Fig. 3. Gráfica para la caracterización del motor obtenida con ayuda de línea de tendencia.

oscilaciones tengan una amplitud constante, siendo sostenidas, en la figura 4 se muestra la respuesta de nuestra sistema con un valor en Kp (valor proporcional) igual a 60 y se puede observar el valor de ”Pu”, el cual la diferencia da igual a 0.2145.

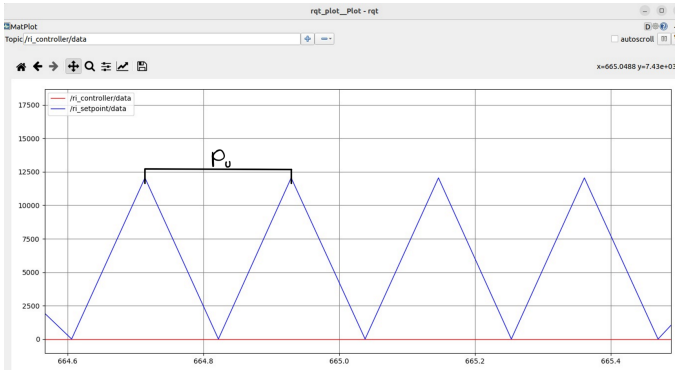

Fig. 4. Gráfica de oscilaciones sostenidas de nuestro sistema.

Se busca que el sistema tenga oscilaciones sostenidas con la variable proporcional, porque cuando el sistema llega a la ganancia crítica o ganancia límite (oscilaciones sostenidas), indica que está cerca de la inestabilidad. En un diagrama de polos y ceros, nos indicaría que los polos del sistema de lazo cerrado se encuentran sobre el eje imaginario, y un mínimo incremento en la ganancia proporcional provocaría la inestabilidad del sistema. Continuando con el cálculo de las ganancias, al obtener los valores de ”Kp” y ”Pu” y con ayuda de las ecuaciones de la tabla de la figura 5 del método que usamos, pudimos calcular los valores de nuestro PID.

Después de sustituir nuestros valores en las ecuaciones y eligiendo un control PID, obtenemos las siguientes ganancias para nuestro PID: `P = 60`, `I = 0.10745` y para `D = 0.0268625`.

C. Arduino con micro-ROS

En nuestro código de Arduino se empezó por la inclusión de todas las librerías necesarias para el trabajo con mirco-ROS, empezamos por la definición de los pines:

Control	K_p	τ_i	τ_d
P	$0.5K_u$	∞	0
PI	$0.45K_u$	$\frac{1}{1.2}P_u$	0
PID	$0.6K_u$	$0.5P_u$	$0.125P_u$

Fig. 5. Tabla para el método Ziegler-Nichols.

- **puente H:** Pin PWM, pines del encoder y pines del enable.
- **Definición PWM:** Frecuencia de PWM, resolución y canal PWM.
- **Definición PID:** Umax y Umin, representando los límites físicos, y tiempo de sampleo.

Posteriormente realizamos la declaración de todas las variables requeridas, empezando por aquellas requeridas para el funcionamiento del encoder y el cálculo de la velocidad, ejemplo de las mismas son el valor PWM, delta tiempo, posición, resolución del encoder, así como sus pulsos. Posteriormente se realizó la declaración de las variables utilizadas en el PID, estos incluyen los valores obtenidos de la ecuación característica del motor y las constantes proporcional, integral y derivativa obtenidas con el método Ziegler Nichols. Por último se realiza la declaración de las variables requeridas para la lectura del tópico y correcto funcionamiento de micro-ROS

1) Funciones: La función `Encoder()` es utilizada para la lectura del encoder, incrementando una variable contador con cada pulso registrado, para esto se toma en cuenta la dirección de giro del motor, realizando una comparación de la lectura de ambos canales del encoder, añadiendo un valor al contador de detectar ambos canales y restando de lo contrario. De igual forma se realiza el cálculo de delta tiempo, tomando el tiempo actual en micro segundos y restandolo al tiempo anterior.

```
void IRAM_ATTR Encoder(){
  //Lectura de los canales del encoder
  //Si la lectura es verdadera
    contador++;
    encoderDirection = true;
  //De lo contrario
    contador--;
    encoderDirection = false;
  //Se calcula delta tiempo
  tiempo_act = micros();
  delta_tiempo = tiempo_act - tiempo_ant;
  tiempo_ant = tiempo_act;
}
```

La función `pose()` verifica la dirección de rotación y realiza la comparación del contador con los pulsos del encoder para determinar el número de revoluciones del motor, añadiendo o restando según corresponda, de igual manera realiza el cálculo de la posición en grados, al multiplicar el contador por la resolución en grados de cada pulso `posición = contador * resolución;` de igual forma se calculan las RPM al dividir los micros en un segundo por el valor de pulsos por delta tiempo `velocidad = 60000000/(pulsos * delta_tiempo);`

```
Si rotación positiva y contador >= pulsos
    revoluciones++;
    contador = 0;
De lo contrario
    revoluciones--;
    contador = 0;

posicion = contador * resolucion;
velocidad = 60000000/(pulsos * delta_tiempo);
```

La función `controlPID()` es la encargada de calcular el error, así como obtener la salida de los componentes del Controlador PID, es decir, el resultado de cada uno de sus elementos tomando en cuenta las constantes Kp, Ki y Kd obtenidas previamente. Para posteriormente poder ser usada en la ecuación característica obtenida para el motor DC y así poder determinar la velocidad del mismo en términos de la variable PWM.

```
void controlPID(){
Componentes del PID
  P = Kp*error;
  I = I_prec + T*Ki*error;
  D = (Kd/T)*(valorEncoderRpm - valorEncoderRpm_anterior);
  U = P + I + D;

Ecuación característica del motor DC
  velocidadMotor = p1*pow(U,3) + p2*pow(U,2) + p3*U + p4;

Salida PWM
  ledcWrite(PWM1_Ch, velocidadMotor);
}
```

La función `timer_callback_1` es el timer encargado de llamar al publisher, dentro de la misma se llama a la función `pose()`, de igual manera calcula la velocidad angular en base a las RPM y envía este valor por el publisher.

```
void timer_callback_1(){
Si timer es diferente de cero
    //Se calcula la velocidad con el encoder
    pose();

Se verifica la dirección de giro
    //Se envía la velocidad en rad/s
    msg.data = velocidad*2*pi/60;
    valorEncoderRpm = velocidad;
  }
}
```

La función `subscription_callback()` es aquella encargada de llamar al subscriber, dentro de la misma se realiza la definición del setpoint para el cálculo del error, de igual manera se configura los pines In1 e In2 del puente H dependiendo del signo del duty cycle para realizar el ajuste de dirección del motor, así como actualiza las banderas de dirección, tomando en cuenta un pequeño delay entre cambios para evitar dañar el motor.

```
void subscription_callback(){

Definicioón del setpoint
  setpoint = abs(msg->data) * 14;
Se llama a la función del PID
  controlPID();

Configura los pines In1 e In2
 dependiendo del signo del duty cycle

  // Si msg->data es negativo
    mantiene su dirección
  //De lo contrario se detiene y cambia de dirección
    Para el motor
    delay(10);
    Cambio de dirección
}
```

En la función `setup()` se hace la declaración de los pinMode para la salida PWM, así como In1 e IN2, de igual manera se configuran las funciones PWM, la configuración del encoder, se crean las init options,así como la creación del nodo publisher y el subscriber. Por último se crea el timer donde se establece la frecuencia de muestreo, que es mayor a dos veces la frecuencia de la señal generada desde nuestra computadora en Python, debido al teorema de muestreo o teorema de nyquist. A su vez se llaman a los executors para ambos nodos.

```
PinMode salida PWM y encoder
Configuracion PWM
Configuracion encoder
Create init_options
Creación de publisher
Creación de subscriber
Creación de timer1
Creación de executors
```

En la función `loop()` se hace un delay de 100 milisegundos y se hace un `RCSOFTCHECK`.

D. Circuito

Para la elaboración del circuito utilizamos los siguientes elementos:

- ESP32
- Protoboard
- leds
- Puente H
- Resistencias
- Motor DC con encoder
- Fuente de voltaje de 6.5 v

Lo primero que hicimos fue conectar la ESP32 y la salida de sus pines para el PWM ya con el PID y la inversión de corriente a tres leds, de tal forma que los leds representaran los inputs del motor y poder visualizar los cambios de dirección y el PWM en los leds. Es importante mencionar que los leds deben ir conectados a una resistencia de 330 KOhms para no quemarlos. El primer led lo conectamos al pin 12, el cual en arduino es la señal del PWM ya con un control PID, por lo que el led aumentaba y disminuia la luminosidad dependiendo de la entrada del sistema, después el segundo led lo conectamos al pin 33 el cual corresponde al ”In2” del puente H y el tercer led lo conectamos al pin 27 siendo el ”In1” del puente H, estos últimos dos son para la inversión de corriente con ayuda del puente H, dependiendo de a que pin se mande una señal, es la dirección de la corriente del motor. Los leds nos ayudaron a visualizar mejor la salida del sistema.

Luego continuamos con conectar el motor al puente H y el puente H a los mismos pines conectados a los leds. El motorreductor con encoder tiene 6 pines, dos para la energía del motor, dos para la energía del encoder y los otros dos para la señal de salida del encoder, la terminal positiva para la energía del puente H la conectamos a 6.5 v y todo lo demás, encoder como leds, los conectamos al voltaje del ESP32 el cual es de 3.3 v, es importante tomar en cuenta que que al usar dos fuentes de voltaje las tierras se deben unir, ya que la unión de las tierras ayuda a mantener una referencia común de 0V y evita diferencias de potencial que podrían causar interferencias. Después se conecta el ”ENA” del puente H al led del PWM y los pines de ”IN1” e ”IN2” a los leds correspondientes para la inversión de corriente. Por último conectamos los pines del motor al puente H para el motor 1 y conectamos los pines de señal de salida del encoder al pin 25 y 26 del ESP32 para obtener la resolución del encoder y la velocidad angular del motor. A continuación se muestra el circuito en la figura 6

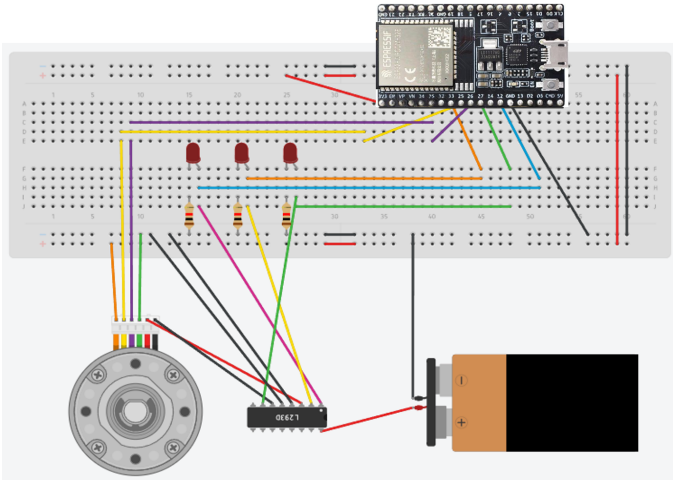

Fig. 6. Circuito.

V. Resultados

Después de aplicar muchos conceptos y hacer la conexión de los mismos se ha podido obtener un buen resultado.

Se pudo tener la seguridad que los tópicos estaban conectados entre sí y mandando datos debido al resultado que se tuvo al mandar `rqt_graph`, ya que se puede observar que se está teniendo una comunicación en el orden correcto. La imágen se puede observar en la figura 7

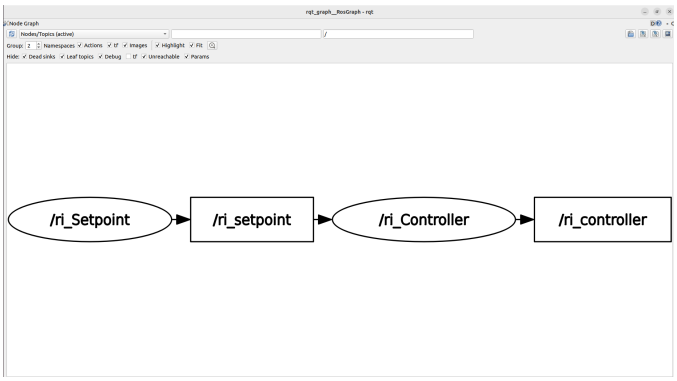

Fig. 7. Resultados del comando rqt graph.

Para poder observar la calidad de los resultados se ha realizado un video en el que se puede apreciar el comportamiento en tiempo real del motor dependiendo de los valores que se le otorgan al setpoint. Se ha considerado muy importante poner

un video en vez de diversas gráficas porque se cree muy importante observar el tiempo de respuesta del motor y así observar el comportamiento más orgánico. Lo primero que se hace en el video fue usar una señal constante y modificar su amplitud en valores tanto positivos como negativos, la razón principal de meter una señal constante fue para poder observar que el duty cycle del setpoint sí se adaptará a la velocidad del motor, algo que sería muy difícil de apreciar con una señal de pulsos o senoidal. Después de haber observado que la velocidad aumenta, disminuye y cambia de dirección con la señal senoidal, y de la misma manera que se manda correctamente la velocidad angular del motor en el tópico controller. Obteniendo un buen resultado ya se modificó el tipo de señal a la senoidal, cuadrado y diente de sierra en dónde se pudo observar un comportamiento esperado respecto a cada señal otorgada y del mismo modo cambios de signo y amplitud de la velocidad angular calculada del motor. Con este resultado se puede dar como cumplido el reto, como siempre con áreas de mejora, pero de la misma manera con un buen resultado. Algo que cabe recalcar es que se han puesto dos leds de color rojo para poder visualizar la salida de la posición del motor y de la misma forma un led de color azul para poder observar como es que cambia el pwm según el PID para poder llegar al setpoint deseado.

A continuación se pueden observar los videos en tiempo real desde diferentes ángulos. En las figuras 8 y 9

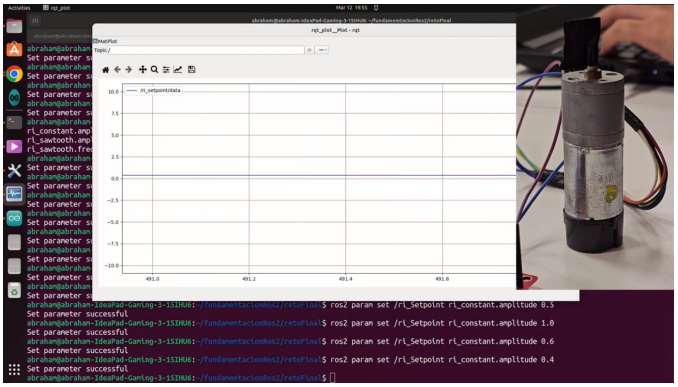

Fig. 8. Evidencia terminal y motor visto de frente.

VI. Conclusiones

A. Conclusiones grupales

Tras concluir este reto nos es claro que, los procesos previos al desarrollo de esta actividad final, tales como la investigación y un desarrollo teórico llevado a la práctica, son algo que resulta desafiante pero enriquecedor para todos nosotros.

Los retos en el camino abordan limitaciones tanto en el hardware como en el software, los cuales no pudimos ver sino hasta iniciada la implementación del control sobre el motor DC en ROS, si bien la teoría de control no es algo nuevo para nosotros, la forma en que esta debía ser aplicada a la situación resulto ser el mayor de los problemas, lo que nos orilló a investigar aún más en este campo. A lo anterior se

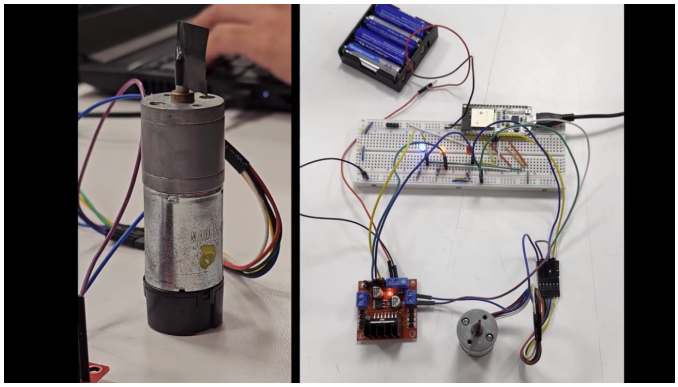

Fig. 9. Motor visto de frente y desde arriba.

le suma la escasa resolución de nuestros encoders y la falta de datasheets de nuestros motores, lo cual género que, lo que podría hacerse de manera rápida y sin muchas complicaciones más allá del cálculo de los valores de K, se complicara, y no solo eso, sino también afectara la calidad del modelo pasando a ser estimaciones y aproximaciones de comportamiento.

Dentro de este entorno caótico, hemos podido encontrar áreas que pueden optimizarse para un mejor rendimiento del sistema de control en ROS, particularmente creemos que podría ser buena idea un controlador de bajo nivel, uno más simple y cercano a nivel de hardware reduciendo así la latencia y aumentando la precisión del mismo, así como el uso de controladores en nodos destinados únicamente a esta labor, permitiendo una mayor agilidad.

Algunas formas en las que creemos podríamos mejorar el controlador actual recaen en la incorporación de filtros para solo emplear aquellos parámetros dentro de un rango, evitando así aquellas mediciones fuera de los parámetros normales, así mismo, `rqt_plot` es un aliado fundamental para monitorear el encoder en tiempo real, permitiendo hacer ajustes más precisos al modelo. De igual forma, `dynamic_configure` facilitaría el proceso de calibración si necesidad de reinicios constantes en los nodos. Además, a partir de investigaciones externas realizadas, `robot_localization` podría ser útil en caso de implementar más de un sensor a nuestro control en búsqueda de una estimación más exacta, aunque claro, es mucho mejor conseguir un motor con una hoja de especificaciones presente.

Este proyecto no solo ha fortalecido nuestra comprensión teórica, sino que también ha mejorado nuestra capacidad para adaptarnos y evolucionar frente a los desafíos prácticos, preparándonos para afrontar con propiedad futuros proyectos en este y otros ámbitos.

B. Conclusiones individuales

1) Abraham Ortiz Castro: Este reto me ha servido mucho para poder aplicar los conocimientos y conceptos vistos en el bloque, pero de la misma manera se han usado muchos de los temas abarcados en control en semestres anteriores, y aunque se tuvo un buen resultado, es notable que existen muchas áreas de oportunidad para el funcionamiento óptimo

de este sistema. Un problema muy significativo que se tuvo fue no tener la hoja de especificaciones del motor. El primer problema que encontramos fue que no se pudo tener una función de transferencia, debido a que desconocimos valores importantes para su cálculo, esto nos orilló a hacer el control del PID con el método heurístico de ziegler y nichols, algo que no está mal, pero sería mejor poder obtener los coeficientes del control con ayuda de la función de transferencia, porque de esa manera sería mucho más preciso el acercamiento y estabilidad con la referencia. El segundo problema que se tuvo fue con el encoder, ya que aunque se hicieron estimaciones de los pulsos que daba por revolución, fueron nada más que un acercamiento a la realidad, el cual estuvo muy influenciado de la imprecisión humana, con ello la lectura de la posición no fue tan precisa como se desea; y si es que no se tiene precisión en la posición tampoco se tendrá en la velocidad debido a que es obtenida desde la posición. Con esta problemática se propone algo muy simple, pero un poco costoso en relación al costo actual, ya que la razón de que no teníamos la hoja de datos era porque fue un motor prestado del laboratorio de la institución, la solución sería comprar un motor de una fuente confiable y tener facilitada toda su información, de la misma manera buscar una buena relación calidad/precio para el encoder ya que aunque conozcamos los pulsos por revolución especificadas por el fabricante, si es un sensor de muy poca calidad, este seguirá siendo poco exacto y por lo tanto se seguirá teniendo una mala retroalimentación. En cambio si es de calidad y se tienen la hoja de datos se podrá calcular la función de transferencia y con ello poder calcular los coeficientes del PID de una manera más robusta, por lo tanto se tendrá un sistema del mismo tipo. Otra área de mejora que yo observo en el reto es la de hacer mensajes personalizados, debido a que las señales en sí tienen muchos valores para poder modificar el comportamiento de la señal, teniendo mensajes personalizados sería más ordenada la manera de manejar estos datos y se tendrían las características de cada señal de una manera agrupada, además en el caso de querer mandar esos datos en un tópico también sería mucho más sencillo; una de las razones principales por la cual no se ha implementado de esta manera es por la limitación que se tuvo en el tiempo, ya que aunque no es una tarea muy difícil, por el hecho de tener que hacer un nuevo paquete de c porque python no soporta el tipo de mensaje personalizados, sería una gran inversión de tiempo el cual desafortunadamente no se tuvo en este caso. Por último, otro aspecto que podría otorgar un mejor resultado sería el de correr todos los nodos y tópicos en micro-ros, debido a que se ha observado en diversas ocasiones que existe una considerable pérdida de datos al momento de trabajar a altas frecuencias desde ros en una laptop, por lo tanto si es que se trabajara con puro micro-ros se espera poder usar frecuencias más altas con las cuales no existan ninguna o la mínima pérdida de datos, obteniendo así un resultado más preciso, ya que claramente al momento de usar frecuencia bajas en la publicación de los datos de señales, los comportamientos de estas pueden llegar a ser diferentes y cuando la frecuencia sea alta se acerca más al comportamiento que se tendría analógicamente. Teniendo así, sistemas más robustos.

En conclusión aunque fue un reto demasiado completo y que exigía de las conexiones de muchos conocimientos vistos en la carrera, se ha concluido de manera satisfactoria, y todas las áreas de mejora se ven muy viables, siempre y cuando se haya tenido el tiempo y la economía necesaria para su implementación.

2) Alan Iván Flores Juárez: Puedo concluir que este reto ha sido un gran desafío, especialmente ya que incorpora los conocimientos vistos en ROS a lo largo de las 4 semanas previas, en conjunto con la integración del proyecto en micro-ROS, con el control PWM para el control de las velocidades angulares. Un gran desafío en este último paso fue la obtención de la ecuación característica del motor DC, ya que no se contaba con el datasheet requerido para obtener su función de transferencia, por lo que se opto por obtener la ecuación característica. Otro de los aspectos complicados dentro del control fue la obtención de las constantes de proporción, integral y derivación, ya que en un inicio se optó por una metodología de prueba y error, pero al no observar un control adecuado, se optó por usar el segundo método de Ziegler Nichols, obteniendo un control que aunque no perfecto, se acercaba al valor de referencia definido sin introducir muchas oscilaciones. Siendo este un aspecto que se podría mejorar en un futuro.

Dentro de los aspectos que nos limitó en los resultados obtenidos en el reto, fue la cantidad de tiempo disponible para el desarrollo e implementación de la solución, ya que considero este fue muy justo tomando en cuenta todos los requisitos de la entrega, tanto el desarrollo en ROS como en micro-ROS, así como la integración y tuneo del controlador PID de los valores recibidos por el encoder, así como las presentaciones y videos requeridos.

Dentro de los elementos y enfoques que se podrían mejorar en el futuro, se podría optar por el uso de un motor DC cuyas especificaciones sean conocidas para así obtener su función de transferencia y de esta manera definir con ayuda de MatLab y el PID Tuner valores más precisos para las constantes Kp, Ki y Kd, resultando en un mejor control. Otra de los enfoques sería tomar otro tipo de control en lugar de un PID, que se adecue mejor a las necesidades del problema, dado que se asume se conocería la función de transferencia del motor a utilizar.

3) Jesús Alejandro Gómez Bautista: Si bien es cierto que todo conocimiento nuevo y actividad en ello implica un reto, la actividad final de este bloque es para mí una de las más desafiantes en cuanto a cantidad de información, tiempo y de destreza se refiere, sin embargo, es una de las más cercanas a lo que en un entorno real podrían desarrollarse, siendo aquí donde entran nuestras capacidades de resolución y análisis critico para hacerles frente.

Como ya lo hemos comentado con anterioridad no solo en este documento, sino también en el video presentado para Manchester Robotics, el desafío más grande durante la implementación del control DC en ROS recayó en el propio motor,

el cual hemos aprendido que, sin hoja de especificaciones, la construcción no solo de su función de transferencia, sino también del control en sí mismo cae en una complejidad muy grande, que si bien, puede ser abordada de otras formas con el suficiente tiempo, implica una ardua investigación, cálculos y aproximaciones (por qué claro, no podemos llegar a un modelo exacto y preciso) para medianamente tener un modelo cercano al funcionamiento real que nos ofrece el motor, todo esto apoyado en la descripción de un modelo caracterizado por variables precisas para cada motor. El uso de aproximaciones por abajo y/o por arriba de los valores ”reales” hace que tengamos siempre un error que nos aleje de esa reducción del error.

Esto nos lleva también a mencionar la resolución del encoder, otro elemento que afecto en la precisión de la medición de la posición real, aumentando nuestros problemas. Por lo anterior, creo que no es sorpresa que todos en el equipo insistamos en buscar una mejor resolución del encoder así como motores con especificaciones claras y concisas, pues al menos estas fueron dos de las razones que hicieron pesar nuestro desarrollo. Así mismo, el emplear el uso de elementos personalizados (como los mensajes) en ROS pudieron haber ajustado las características de nuestros mensajes a elementos exactos y de mayor utilidad. De igual forma, la documentación y difusión de este problema en los foros de la comunidad podría hacer que, personas en nuestro lugar, puedan prepararse ante esta clase de dificultades, pues para infortunio nuestro no contamos con los recursos suficientes como para prever esta situación.

Pese a estas dificultades, creo que podemos sacar muchas áreas de oportunidad que nos permitirán crear modelos más robustos y adaptativos a las problemáticas que se intenten abordar y solucionar, demás esta decir la necesidad e importancia que es el consultar la documentación y teoría de estos temas cuando encontremos estas dificultades en el camino.

4) Ulises Hernández Hernández: En conclusión, durante el desarrollo del proyecto nos enfrentamos a diversos desafíos, principalmente debido a la falta de una hoja de especificaciones para el motorreductor DC con encoder, lo que dificultó la obtención de la función de transferencia y de la velocidad angular máxima para diseñar un control PID preciso. También otro problema que parte del anterior, fue la caracterización del motor resultó en estimaciones, debido a que la resolución del encoder igual eran estimaciones y la velocidad angular calculada nos la daba en enteros en código, por lo que no era exacto, lo que limitó la precisión del control PID diseñado utilizando el método de Ziegler-Nichols. Además, las restricciones de tiempo impuestas al proyecto generaron presión adicional, dificultando la realización adecuada del diseño y la implementación del control.

Para mejorar el rendimiento del control en ROS, es fundamental optimizar el controlador PID, ajustando sus parámetros de manera precisa, lo cual se logra con la hoja de especificaciones de los componentes utilizados, en este caso, del motor y del encoder. Asimismo, se debe optimizar la configuración de la red para reducir la latencia de comunicación entre los nodos de ROS2, utilizando protocolos de comunicación eficientes como DDS y optimizando la configuración de red en la ESP32. Además, se pueden explorar técnicas avanzadas de control, como el control adaptativo o predictivo, para mejorar aún más el rendimiento y la robustez del sistema; y por último según el paradigma de la robótica y ROS, los controladores de bajo nivel, como el controlador PID para el motor, deben ubicarse lo más cerca posible del hardware. En este caso, ejecutar el controlador PID en la ESP32, directamente conectada al motor, esto garantizaría una respuesta más rápida y precisa del sistema de control. Estas estrategias combinadas pueden ayudar a superar los desafíos encontrados durante el desarrollo del proyecto y mejorar significativamente el rendimiento del control en el entorno de ROS2.

REFERENCES

[1] ROS 2 Documentation — ROS 2 Documentation: Foxy - documentation. (s.f.). link.
[2] Kutluca, H. (2022, Marzo 14). Robot Operating System 2 (ROS 2) Architecture - Software Architecture Foundations - Medium. Medium. link.
[3] es/ROS/Conceptos - ROS Wiki. (s.f.). link.
[4] Managing large projects — ROS 2 Documentation: Foxy documentation. (s.f.). link.
[5] About parameters in ROS 2 — ROS 2 Documentation: Foxy documentation. (s.f.). link.
[6] ROS Custom Message Support - MATLAB y Simulink - MathWorks América Latina. (s.f.).
[7] Micro-ROS. (s.f.). micro-ROS. link.
[8] Matan. (2023, Septiembre 21). ¿Cómo funcionan los ADC en un circuito? Electricity - Magnetism. link.
[9] Carranza, S. (2022, Septiembre 17). CONOCIENDO AL ESP32. TodoMaker. link.
[10] Llamas, L. (2023, Agosto 25). Cómo usar las salidas analógicas PWM en un ESP32. Luis Llamas. link.
[11] Matan. (2023, Septiembre 21). ¿Qué es un puente H y cómo funciona? Electricity - Magnetism. link.
[12] Company, E. P. (s.f.). ¿Qué es un Encoder? Lo Que Hay Que Saber Sobre Los Codificadores. Encoder Products Company, Inc. link.
[13] Schwartz. (2020, Marzo 7). Encoder - ¿Qué es? 4 tipos de encoder y su funcionamiento. LBA Industrial. link.
[14] Buxo, T. L. (2024, Febrero 5). Encoders: Qué son, tipos y ejemplos. Servomotors. link.
[15] Descripción y aplicación de encoders y sensores de inclinación — SICK. (s.f.) link.
[16] Zapata, F. (2023, Agosto 11). Método de Euler. Lifeder. link.
[17] Llamas, L. (2023, Septiembre 5). Cómo usar interrupciones en un ESP32. Luis Llamas. link.
[18] rqt plot - ROS Wiki. (s. f.). link.
[19] rqt graph - ROS Wiki. (s. f.). link.
[20] ManchesterRoboticsLtd. (s. f.). GitHub - ManchesterRoboticsLtd/TE3001B-Robotics-Foundation-2024. GitHub. link.
[21] Roboinges GitHub - RoboInges-Manchester. GitHub. link.

VII. ANEXOS

APPENDIX A
METODOLOGÍA DE TRABAJO

Acceda al archivo de la metodología usada a través de este link: metodología

APPENDIX B
VIDEO. DEMOSTRACIÓN DEL RETO

Acceda al archivo de video sobre la explicación del reto para Manchester Robotics mediante este link: video demostrativo

APPENDIX C
PRESENTACIÓN FINAL. MANCHESTER ROBOTICS

Acceda al archivo de la presentación final a través de este link: Presentación

APPENDIX D
CÓDIGO DE ARDUINO.

Acceda al código de arduino.

APPENDIX E
CÓDIGOS DE PYTHON.

Acceda al código generador de setpoint.
Acceda alcódigo launch.
Acceda al código terminación yaml.
Acceda al código terminación xml.
Acceda al código setup