Reto semanal 3. Manchester Robotics

Oscar Ortiz Torres A01769292

Yonathan Romero Amador A01737244

Ana Itzel Hernández García A01737526

Fundamentación de robótica

Grupo 101

Jueves 6 de marzo de 2025

Resumen

En este reto se desarrolló e implementó un nodo en una ESP32 utilizando Micro ROS, con el objetivo de controlar la velocidad y dirección de un motor DC de 12V mediante un puente H L298N. Para lograrlo, se estableció la comunicación con ROS2 desde una terminal en Ubuntu, permitiendo la recepción y procesamiento de comandos de velocidad a través del tópico `/cmd_pwm_ROSario`.

La metodología consistió en configurar la ESP32 con las bibliotecas necesarias para Micro ROS, desarrollar un nodo que se suscribieron al tópico, validar y procesar los valores de entrada en el rango [-1, 1], y generar la señal PWM adecuada para accionar el motor. Además, se implementó un mecanismo de control que reduce progresivamente la señal antes de cambiar la dirección del motor, evitando así arranques y frenados bruscos.

Objetivos

El objetivo general del reto es desarrollar e implementar un nodo en una ESP32 utilizando Micro ROS, capaz de recibir comandos de velocidad a través del tópico `/cmd_pwm_ROSario`, procesar los valores dentro del rango [-1, 1] y generar una señal PWM adecuada para controlar la velocidad y dirección de un motor DC de 12V mediante un puente H L298N, permitiendo la comunicación con ROS2 desde una terminal en Ubuntu.

Objetivos particulares

1. Configurar la ESP32 con Micro ROS para establecer la comunicación en un entorno ROS2 en Ubuntu.
2. Desarrollar un nodo en la ESP32 qué se suscriba al tópico `/cmd_pwm_ROSario`, recibiendo y validando los valores de entrada.

3. Implementar el procesamiento de la señal para convertir los valores de entrada en señales de control para el puente H L298N, manejando la dirección y la velocidad del motor.
4. Generar la señal PWM en la ESP32 y aplicarla a los pines del puente H para accionar el motor de acuerdo con el comando recibido.
5. Realizar pruebas de funcionamiento publicando valores desde la terminal de Ubuntu y verificando la respuesta del motor
6. Analizar el desempeño del sistema y documentar los resultados obtenidos.

Introducción

Para la realización de este proyecto se utilizó un motoreductor, estos son un mecanismo capaz de regular la velocidad de giro de un motor en una máquina para que funcione con un ritmo determinado. Estos están hechos a base de una cadena de engranajes, estos son los encargados de disminuir la velocidad del mecanismo sin que el motor lo resienta así aumentando su potencia mecánica. *(Garcia, 2023)*

Para poder utilizar los motorreductores tenemos que entender cómo funcionan, ya que estos se controlan a través de un módulo, comúnmente un puente H en robótica, donde este recibe una señal y la transforma en voltaje para el motor. Esta señal es conocida como PWM (Pulse Width Modulated) esta señal cuenta con una frecuencia fija, lo que varía la potencia del motor es su amplitud. (Bercial, 2023)

En este proyecto nuestro motor manejara 12V por lo que nuestro PWM ira de 0V a 12V dependiendo de la amplitud, en este caso será implementado por una ESP32 el cual es un microcontrolador con capacidades de internet el cual es versátil, la resolución de nuestro PWM es de 8 bits. (Llamas, 2023)

Para poder leer las señales analogicas y transformarlas a digitales se necesita de un módulo ADC, estos leen señales físicas como luz, temperatura, sonido, etc. y lo transforman en señales digitales, es decir una serie de valores discretizados, donde la señal se divide en secuencias las cuales dependen de nuestra tasa de muestreo. Estos módulos son necesarios para que nuestros microcontroladores puedan procesar los datos, actualmente muchas tarjetas ya tienen integrados un módulo ADC como nuestro ESP32. (Electronics, 2024)

Micro ROS es una librería hecha para poder utilizar ROS2 en microcontroladores, esto para poder extender las redes creadas por ROS a microcontroladores, para esto se optimizan recursos utilizados así como memoria para poder entrar en nuestros sistemas embebidos, en este caso utilizaremos micro ROS para nuestra ESP32 y poder crear nodos dentro de esta. (*micro-ROS*, s. f.)

Solución del problema

Para cumplir con los objetivos del reto, se desarrolló e implementó un nodo de Micro ROS en la ESP32, la metodología empleada y los elementos utilizados son los siguientes:

Estructuración del código

1. Configuración del ESP32: Se utilizaron las bibliotecas necesarias como `micro_ros_arduino.h` para la implementación de Micro ROS con la ESP32, permitiendo la comunicación con ROS2.
2. Suscripción al tópico `/cmd_pwm_ROSario`: Se desarrolló un nodo en el ESP32 que aseguraba la recepción y validación de los valores de entrada dentro del rango [-1, 1].
3. Definición de pines y parámetros:

   a. Se configuraron los pines de salida para la señal PWM y para el control de dirección del puente H.
   b. Se definió un LED indicador de conexión.
   c. Se establecieron los parámetros de frecuencia (5000 Hz), resolución (8 bits) y canal del PWM.
   d. Se declararon variables de almacenamiento y una bandera para detectar cambios en el sentido de giro.

4. Recepción y procesamiento del dato recibido
   a. Se valida que el valor recibido esté dentro del rango permitido [-1,1].
   b. Se extrae el signo del valor para determinar la dirección de giro.
   c. Se detecta si hay un cambio de sentido de giro comparando el signo con el valor previo.
   d. Se activa la bandera state si hay un cambio de giro.
5. Control del motor mediante PWM
   a. Si hay un cambio de sentido, se reduce progresivamente la señal PWM hasta llegar a cero antes de invertir la dirección del motor.
   b. Se establece la dirección de giro configurando los pines del puente H según el signo del valor recibido.
   c. Se convierte el valor de entrada en un entero de 8 bits (0-255) para la señal PWM.
   d. Se ajusta progresivamente el PWM:
      i. Si hubo un cambio de sentido, se incrementa progresivamente desde 0 hasta el nuevo valor objetivo.

ii. Si no hubo cambio de sentido, se incrementa o decrementa de manera gradual según la diferencia entre el nuevo valor y el anterior.

6. Almacenamiento de datos para la siguiente iteración
    a. Se guardan el signo y el valor del PWM aplicado para usarse en la siguiente iteración del callback.

Implementación física

El sistema físico incluyó:

- Una ESP32 conectada al puente H L298N para controlar el motor.
- Alimentación externa para el motor, independiente de la ESP32.
- Conexión entre la ESP32 y la PC mediante USB para la comunicación con ROS2 en Ubuntu.

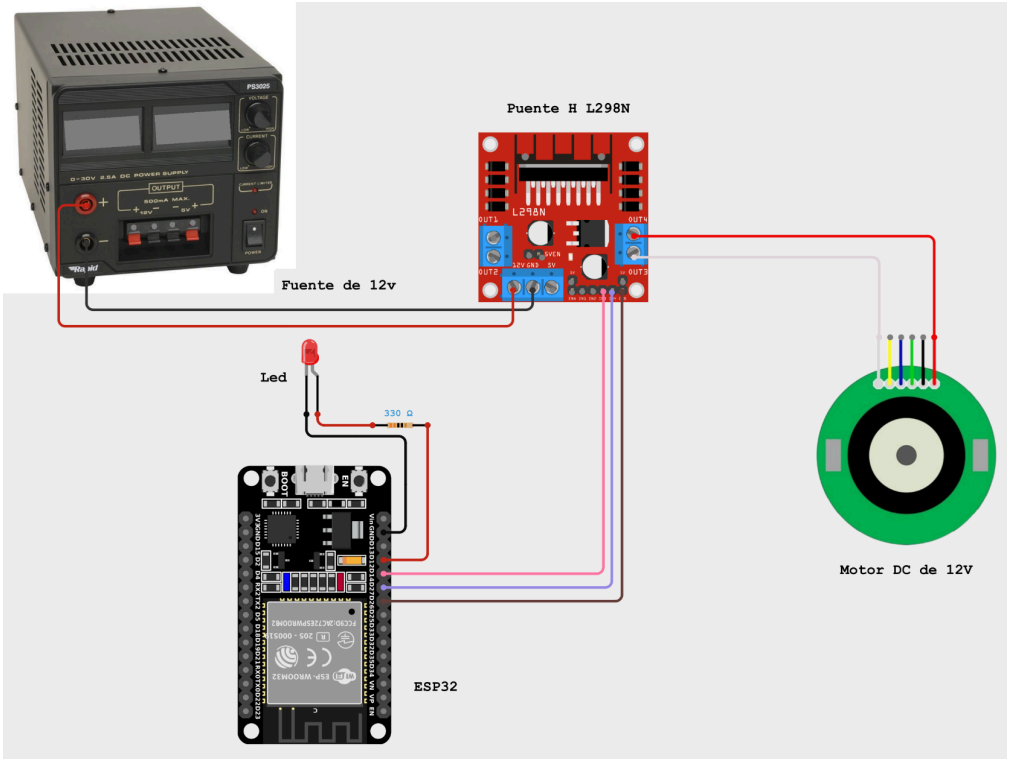

Figura 1. Diagrama de conexión de los componentes y el ESP32

Con esta implementación, se logró un control eficiente del motor, minimizando arranques y frenados bruscos y asegurando la dirección adecuada del giro del motor según los comandos recibidos en el tópico `/cmd_pwm_ROSario`. La comunicación en tiempo real entre la ESP32 y ROS2 permitió la supervisión y ajuste del comportamiento del motor a través de la terminal de Ubuntu.

Resultados

Para validar el correcto funcionamiento del sistema se realizaron diversas pruebas en las que se enviaron los datos desde la terminal de Ubuntu a través del tópico `/cmd_pwm_ROSario`.

Se observaron cambios en la velocidad y dirección del motor dependiendo de los valores enviados, es decir, al enviar un valor de 1 tiene el 100% del ciclo de trabajo, así que girará a su máxima velocidad hacia la derecha, mientras que con un valor de -1 sigue girando a su máxima velocidad pero con dirección hacia la izquierda.

Si el motor recibe un valor menor a $|0.6|$, el 60% de la velocidad, entra en lo que se conoce como “zona muerta”, lo cual quiere decir que se encuentra en el área de valores de entrada donde el sistema no responde porque los valores son demasiado pequeños para que el motor pueda reaccionar.

Se iniciaron las pruebas con valores de 0 a 1, es decir, el motor inicia en reposo y comienza a girar hacia la derecha con una velocidad creciente hasta alcanzar su velocidad máxima. Continuamos con una prueba de 1 a -1, como hay un cambio de dirección, el código detecta el cambio de signo y ejecuta el sistema de frenado donde reduce gradualmente la señal PWM a 0, desactiva los pines `In1` e `In2` para detener el motor momentáneamente, se invierte la polaridad activando el pin `In1` y comienza a aumentar el PWM en la nueva dirección.

La siguiente prueba va de -1 a -0.6, donde el motor ya está girando en dirección a la izquierda, se empieza a reducir progresivamente la velocidad hasta el 60% sin cambiar la dirección. Continuamos con los valores de -0.6 a 0.6, en el cual hay un cambio de signo que el código detecta como necesario de invertir la dirección, aplicando la misma estrategia de frenado que en la segunda prueba, a diferencia de que aquí el motor gira en dirección hacia la derecha con un 60% de la velocidad máxima.

La última prueba fue con valores de 1 a 0, por lo tanto, el motor está girando con el 100% de su velocidad en dirección a la derecha, se reduce progresivamente la velocidad hasta detenerse por completo.

Con ayuda de las pruebas comprobamos que el sistema responde de manera eficiente a los datos recibidos sin generar algún retraso significativo, además de que verificamos que, en cambios de dirección, el código implementa un breve frenado para evitar los movimientos bruscos en el motor, a la par, en cambios de velocidad en la misma dirección la transición es más suave y progresiva.

El video de demostración con las pruebas realizadas se encuentra disponible en la sección de anexos de este documento.

Conclusiones

La creación y diseño de nuestro nodo en una ESP32 utilizando Micro ROS permitió cumplir nuestros objetivos establecidos para este reto. Se logró crear una comunicación en un entorno ROS2 en Ubuntu, permitiendo la recepción de comandos a través de un tópico `/cmd_pwm_ROSario`, esto para posteriormente procesarla y generar una señal PWM para poder así controlar la dirección y velocidad de un motorreductor de 12V mediante un puente H L298N.

Con esto se pudo identificar los límites de nuestra zona muerta del motor, lo cual nos permite saber el rango funcional de nuestro motor, siendo de aproximadamente de 7 a 12 volts.

Al finalizar este reto pudimos concluir algunas posibles mejoras futuras para así tener un control más eficiente de nuestro motor. Para esto debemos cambiar nuestro mapeo de nuestro motor para no tener en cuenta nuestra zona muerta. Aumentando así nuestra resolución de voltaje teniendo un mayor control, además se podría reducir las iteraciones necesarias para cambiar de un PWM a otro, para así reducir retardos en la respuesta de nuestro sistema. Esto para evitar daños en el motor ocasionados por cambios abruptos en el PWM.

Bibliografía o referencias

Bercial, J. (2023, 28 febrero). *¿Qué es el PWM y para qué sirve?* GEEKNETIC. https://www.geeknetic.es/PWM/que-es-y-para-que-sirve

Electronics, A. (2024, 20 marzo). *Aspectos básicos de los convertidores de analógico a digital*. Arrow.com. https://www.arrow.com/es-mx/research-and-events/articles/engineering-resource-basics-of-analog-to-digital-converters

Garcia, E. (2023, 16 junio). Definición de un motorreductor y cómo funciona. *Ferretería y Suministros industriales Online. - Todoparalaindustria.com*. https://todoparalaindustria.com/blogs/blog/definicion-de-un-motorreductor-y-como-funciona

Llamas, L. (2023, 25 agosto). *Cómo usar las salidas analógicas PWM en un ESP32*. Luis Llamas. https://www.luisllamas.es/esp32-pwm/

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

Anexos

Repositorio: https://github.com/oscarOT09/motor_node_ESp32_ROSario

Video evidencia: challenge3_evidencia.mp4