RETO SEMANAL III
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

Ahora que hemos aplicado los conceptos clave de ROS 2 en actividades anteriores, es el momento de adentrarnos en el mundo de los microcontroladores y ROS. En esta etapa, exploraremos micro-ROS, que actúa como un puente entre ROS y nuestro entorno de desarrollo en Arduino IDE. El ESP32 será nuestro microcontrolador, y enfrentaremos el desafío de trabajar con componentes físicos que nos brindarán una primera experiencia en este entorno. Es necesario retomar conceptos como el PWM aplicado a microcontroladores y la funcionalidad que un módulo ADC tiene con el fin de diseñar dispositivos sin importar la diferencia digital - analógica.

II. Objetivos.

- Poder realizar una analogía de todo lo aprendido y aplicarlo para microros2.
- Aplicación del pwm para control sencillo de luminosidad de los leds.
- Controlar la intensidad de un led con ayuda de un potenciómetro.
- Controlar la intensidad de otro led suscribiéndose a un tópico de ros que tenga el duty cycle de un pwm.

III. Introducción.

Ahora que entendemos la funcionalidad de ROS 2, es momento de colocar un reto más importante, la manipulación de voltaje mediante microcontroladores a través de micro-ROS, un nuevo entorno que debemos comprender y emplear para lograr el control dispositivos que nos encaminan hacia la resolución del reto. Es por ello que en esta ocasión nos enfocaremos en entender que es micro-ROS, los elementos que hacen a un ADC tan importante y como podemos emplear el PWM en un ESP32.

A. ¿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]

1) Características:

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

2) Importancia de Micro-Ros: Micro-Ros ha permitido contruir 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 aplicacion en el área de la robótica viene muy bien en el desarrollo de drones, robots moviles 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. [1]

B. ¿Qué son los modulos 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 modulos 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. [2]

1) Funcionamiento: El funcionamiento del módulo ADC puede verse de la siguiente manera: el ADC observa una tensión analógica producida por algun 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 sus hoja de especificaciones. [2]

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. [2]
- **Delta-Sigma:** hace uso de técnicas de sobremuestreo y filtrado, esto aproxima mejor sus valores, lo que se traduce en una mayor precisión. [2]
- **Convertidor de Canalización:** se divide a la señal en ”canales” los cuales se analizan por separado para determinar el valor. [2]

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. [2]

C. PWM en ESP32

El PWM (Pulse Width Modulation) permite el control de la intensisdad 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 comunmente 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. [3]

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 dispostivo 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”). [3]

IV. Solución del problema

A. Circuito

Para la solución de este problema se necesita realizar un circuito para poder mandar un PWM a un led y de esta manera controlar la intensidad de luz de un led. Los componentes utilizados fueron:

- ESP32
- Protoboard
- Potenciómetro
- Leds
- Resistencias

Primero tomamos en cuenta lo siguiente: el pinout de un potenciómetro nos dice que en los pines de los extremos, uno va conectado a tierra y el otro a un voltaje de 3.3 v, y el pin de en medio es la salida de la señal. En cuanto al led, la patita más larga es para el voltaje y la mas corta va a tierra, en este caso el voltaje será mandado por un pin del ESP32, para poder modular la intensidad del led con el potenciómetro. Es importante tomar en cuenta que para proteger el circuito y evitar daños el led se debe conectar con una resistencia como por ejemplo de 330 KOhms, debido a que resistencia limita el flujo de corriente que pasa a través del led, lo que ayuda a mantener la corriente a un nivel seguro y prevenir que se sobrecaliente o se queme.

Sabiendo lo anterior mencionado emepezamos con el circuito, primero compartimos del ESP32 la tierra y un voltaje de 3.3 v a la protoboard en los laterales como se indica en su simbología, tierra en ”-” y el voltaje en ”+”, después conectamos el potenciómetro, los pines de los extremos, uno a voltaje y el otro a tierra los que compartimos de la ESP a la protoboard, no importa el orden, solo cambia en cuanto a la dirección del giro, si va a aumentar la resistencia o dismunuir, ahora el pin de la señal lo conectamos al pin 34 para leer el valor analógico y convertirlo a digital. Luego conectamos el led, su terminal negativa (pin pequeño) a un resistencia de 330 KOhms y la resistencia a tierra, luego la terminal positiva la conectamos al pin 12 para poder modificar el duty cycle y mandar la señal a ese pin, controlando la luminosidad del led. Por último conectamos otro led, su terminal negativa a una resistencia y la resistencia a tierra, y la terminal positiva al pin 33 del ESP32 para modular la intensidad de luz del led con la señal digital del potenciómetro. A continuación se puede observar una foto del circuito realizado 1.

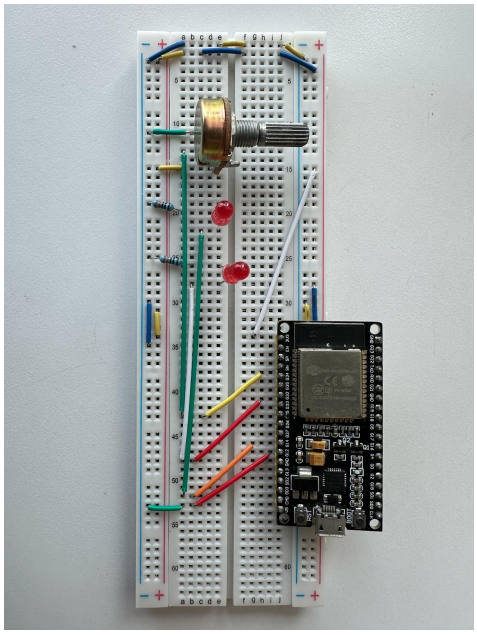

Fig. 1. Circuito

B. Código (Arduino IDE)

En nuestro código en arduino, se hizo la importación de todas las librerías requeridas para el funcionamiento adecuado de nuestra ESP-32 y ROS2, de misma forma se realizó la definición de los pines, tanto del pin del LED (Pin 12), el pin para el PWM (Pin 33) y por último el pin que recibe la entrada analógica del potenciometro (Pin 34). Por último la declaración del voltaje máximo a recibir a tráves del potenciometro, así como la resolución de medida.

```
// Se incluyen las librerias necesarias.
#include <micro_ros_arduino.h>
#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <std_msgs/msg/int32.h>
#include <std_msgs/msg/float32.h>
#include "driver/adc.h"
#include "driver/ledc.h"

// Definiciones de pines
#define POTENTIOMETER_PIN 34
//Pin que lee la entrada del pot.
#define LED_PIN 12
// Pin que sera modificado con por duty cycle
#define PWM_PIN 33
// Pin que será modificado con el pot

// Rangos de ADC y voltaje
#define ADC_RESOLUTION 4095
#define VOLTAGE_MAX 3.3
}
```

En las priemra secciones de nuestro código se realiza la verificación de errores con las funciones `RCCHECK()` y `RCSOFTCHECK()`, se hace la definición del valor inicial del potenciometro y se crea la función `error_loop()` que no recibe ningun argumento de entrada, esta función es la encargada de hacer prender alternamente el LED en caso de que detectar un error.

```
//Marco para verificación de errores
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn;
if((temp_rc != RCL_RET_OK)){error_loop();}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn;
if((temp_rc != RCL_RET_OK)){}}

// Se define lectura del potenciometro en global
int pot_value = 0;

// Indicador de error critico
void error_loop(){
  while(1){
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    delay(100);
  }
}
}
```

A continuación se crean las funciones `timer_callback_1()` y `timer_callback_2()` ambas recibiendo como argumentos un timer y la variable `last_call_time`, la primera de las cuales es la encargada de leer el valor del potenciometro que recibe a tráves del pin `PWM_PIN` declarado previamente y almacenarlo en una variable llamada `pot_value`. El segundo es el encargado de publicar el valor del potenciometro que recibe a través de `pot_value` y que almacena dentro de `msg.data`, para esto se utiliza el tópico `raw_pot_publisher`, a su vez se mapea el resultado del potenciometro para determinar el voltaje recibido, dicho calculo se realiza con la siguiente fórmula:

```
float voltage = pot_value * VOLTAGE_MAX / ADC_RESOLUTION;
```

Por último realiza la publicación de este voltaje calculado en `msg.data` con el tópico `voltage_publisher`. Por último se calcula el valor del PWM haciendo un mapeo del valor recibido a través del potenciometro, el valor de resolución `ADC_RESOLUTION` configuado de 0-4095 y una resolución de 8 bits para el PWM, es decir de 0 - 255. Escribiendo el valor obtenido del PWM al pin previamente asignado como `PWM_PIN`.

```
// Callback al que se le llamara
void timer_callback_1(rcl_timer_t * timer_1,
int64_t last_call_time)
{
  RCLC_UNUSED(last_call_time);
  if (timer_1 != NULL) {
    // Leer el valor del potenciómetro
    pot_value = analogRead(POTENTIOMETER_PIN);


  }
}

// Callback al que se le llamara 2
void timer_callback_2(rcl_timer_t * timer_2,
int64_t last_call_time)
{
  RCLC_UNUSED(last_call_time);
  if (timer_2 != NULL) {
    // Publicar el valor crudo del potenciómetro
    msg.data = pot_value;
    RCSOFTCHECK(rcl_publish(&raw_pot_publisher, &msg, NULL));

    // Mapear el valor del potenciómetro al rango de 0 a 3.3V
    float voltage = pot_value * VOLTAGE_MAX / ADC_RESOLUTION;

    // Publicar el voltaje mapeado
    msg.data = voltage;
    RCSOFTCHECK(rcl_publish(&voltage_publisher, &msg, NULL));

    // Calcula el valor de PWM a partir del valor del pot
    int pwmValue = map(pot_value, 0, 4096, 0, 255);

    // Configura el brillo del LED utilizando el valor de PWM
```

```
    analogWrite(PWM_PIN, pwmValue);
  }
}
```

En la siguiente sección se realiza el `subscription_callback`, una función que calcula el valor PWM2 a partir del Duty Cicle recibido a través del `msg.data` con un valor entre 0 y 100 transformándolo a una resolución de 8 bits o 256 valores. Escribiendo el resultado en `LED_PIN` con el valor obtenido en PWM2.

```
void subscription_callback(const void * msgin)
{
  const std_msgs__msg__Float32 * msg =
  (const std_msgs__msg__Float32 *)msgin;
  // Calcula el valor de PWM a partir del duty cycle recibido
  int pwmValue2 = map(msg->data, 0, 100, 0, 255);

  // Configura el brillo del LED utilizando el valor de PWM
  analogWrite(LED_PIN, pwmValue2);
}
```

Por último en la función estandar de Arduino `setup()` se realiza la inicialización de todos los pines previamente descritos, a su vez de la creación de las opciones de inicialización, la creación del nodo `smicro_ros_arduino_node`, el publisher `raw_pot_publisher` y `voltage_publisher`, encargados de publicar el valor leido directamente del potenciometro y del voltaje calculado respectivamente, la creación de un subscriber `pwm_duty_cycle_sub` que manejará el Duty Cycle y por último la creación de ambos timers con sus respectivos `timers_callbacks`.

```
void setup() {
  set_microros_transports();

  // Inicializa el pin del LED como salida
  pinMode(LED_PIN, OUTPUT);
  pinMode(PWM_PIN, OUTPUT);


  delay(2000);

  allocator = rcl_get_default_allocator();

  //create init_options
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // create node (string vacio sería el nombre del namespace)
  RCCHECK(rclc_node_init_default(&node,
  "micro_ros_arduino_node", "", &support));

  // create publisher para mandar entrada directa pot
  RCCHECK(rclc_publisher_init_default(
    &raw_pot_publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "ri_raw_pot"));
  // create publisher para mandar voltaje de pwm
  RCCHECK(rclc_publisher_init_default(
    &voltage_publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "ri_voltage"));

  // create subscriber que hará pwm ,según duty cycle obtenido
  RCCHECK(rclc_subscription_init_default(
    &pwm_duty_cycle_sub,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "ri_pwm_duty_cycle"));

  // create timer1
  const unsigned int timer_timeout = 10;
  RCCHECK(rclc_timer_init_default(
    &timer_1,
    &support,
    RCL_MS_TO_NS(timer_timeout),
    timer_callback_1));

  // create timer2
  const unsigned int timer_timeout2 = 100;
  RCCHECK(rclc_timer_init_default(
    &timer_2,
    &support,
    RCL_MS_TO_NS(timer_timeout2),
    timer_callback_2));

  // create executors para todos los publisher y suscribers
  RCCHECK(rclc_executor_init(&executor, &support.context, 3,
  &allocator));
  RCCHECK(rclc_executor_add_timer(&executor, &timer_1));
  RCCHECK(rclc_executor_add_timer(&executor, &timer_2));

  RCCHECK(rclc_executor_add_subscription(&executor,
  &pwm_duty_cycle_sub, &msg, &subscription_callback, ON_NEW_DATA));

  msg.data = 0;
}
}
```

Por último en la función `loop()` se llama a la función `RCSOFTCHECK()`, la cual es una herramienta para comprobar y verificar el estado del software y hardware sea correcto y añadir una capa más de confiabilidad.

```
void loop() {
  delay(100);
  RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

C. ROS 2

Dentro de la practica se uso ROS 2 para modificar el duty cycle y de esta manera modular la intensidad del led. Antes de correr lineas de código el circuito y el código ya estaban en funcionamiento, así como el ESP32 conectado a la laptop. Primero en una terminal se corrió la siguiente linea para abrir el puerto serial

```
ros2 run micro_ros_agent micro_ros_agent serial --dev/dev/ttyUSB0
```

la parte de `ttyUSB0` se obtiene revisando los dispositivos conectados e identificando cual es el conectado a la ESP32. Después nos suscribimos al nodo de valor del potenciómetro y del voltaje con las siguientes lineas

```
ros2 topic echo /micro_ros_esp32/raw_pot
ros2 topic echo /micro_ros_esp32/voltaje
```

Finalmente para modificar el valor del duty cycle, publicamos el valor del PWM en porcentaje con la siguiente linea, en el valor de data que se encuentra entre comillas dobles va el valor que se quiere para el duty cycle.

```
ros2 topic pub /micro_ros_esp32/pwm_duty_cycle
--once std_msgs/msg/Float32 "data:50"
```

Por último para visualizar la gráfica de como se modifica el voltaje que recibe el led, desde la terminal se corre la siguiente linea de código `ros2 run rqt_plot rqt_plot`

V. Resultados

Los resultados obtenidos se puden observar en la figura 2, en la que se demuestra como ajustando el valor del potenciometro, se ajusta el valor del PWM que se ve reflejado en el ajuste del brillo de nuestro LED, de igual manera se muestra la configuración y conexiones en físico realizadas en el ESP32 y la gráfica vista a través de `rqt_plot`.

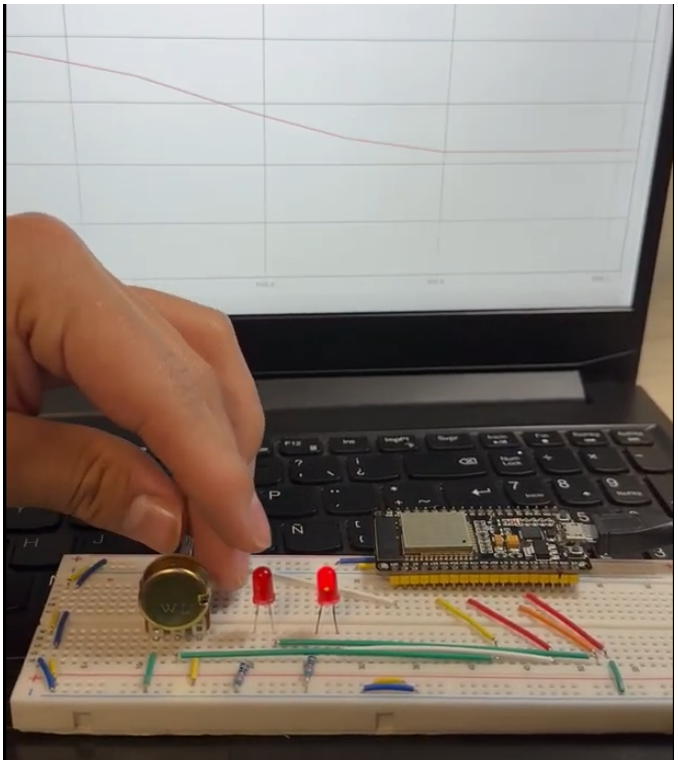

Fig. 2. Evidencia Video

A. *Interpretación de resultados*

En el video se puede observar la gráfica generada por el PWM, esta gráfica muestra en el eje Y el voltaje mandado al led y en el eje X se ve el tiempo, por lo que se aprecia como cambia el voltaje mandado al led al pasar el tiempo. Tambien se observa como se tiene una buena respuesta adaptando la luminosidad del led correspondiente tanto de cuando se modifica el valor del potenciómetro, como en el caso de que se cambie el duty cycle directamente en el tópico desde la terminal. Gracias a este análisis queda claro que la respuestas y la capacidad de tiempo de respuesta de un microcontrolador es mucho mejor que la de un dispositivo portatil.

VI. Conclusiones

Este reto ha resultado muy interesante, el hecho de tener la opción de usar ros2 con un microcontrolador hizo de nuestro muy diferente la visión que teniamos de este sistema operativo. Ya es posible darse una idea de como es que podemos controlar los robots y de la misma manera la importancia de tener muy claro un sistema robusto, así siendo capaz de tener una buena comunicación y manejo de datos. Los resultados obtenidos fueron los de esperarse, en este caso no existió ninguna complicación, creemos que fue debido a que los conceptos de ros2 ya estaban claros gracias a las actividades realizadas anteriormente y de la misma manera hemos trabajado muchas veces con microcontroladores, por lo tanto solamente era hacer la unión de ambos conceptos y así obteniendo un gran resultado, al final fuimos capaces de controlar los dos leds de las dos maneras planteadas en el reto. Algo que podríamos mejorar es la presentación del circuito debido a que estuvo un poco desordenada debido al tiempo. En conclución, gracias a las actividades previas y el conocimiento de conceptos básicos esta actividad se ha concluido de manera exitosa.

References

[1] Micro-ROS. (s.f.). micro-ROS. link.

[2] Matan. (2023, Septiembre 21). ¿Cómo funcionan los ADC en un circuito? Electricity - Magnetism. link.

[3] Llamas, L. (2023, Agosto 25). Cómo usar las salidas analógicas PWM en un ESP32. Luis Llamas. link.

[4] rqt plot - ROS Wiki. (s. f.). link.

[5] rqt graph - ROS Wiki. (s. f.). link.

[6] ManchesterRoboticsLtd. (s. f.). GitHub - ManchesterRoboticsLtd/TE3001B-Robotics-Foundation-2024. GitHub. link.