Conversor A/D

Brayan Steven Mendivelso Pérez
est.brayan.mendive@unimilitar.edu.co
Docente: José de Jesús Rúgeles

***Resumen*— El laboratorio de conversión A/D con la Raspberry Pi Pico 2W tuvo como propósito comprender el funcionamiento de los conversores analógico–digitales (ADC) y su importancia en los sistemas de comunicación digital. En la primera parte, se exploró el uso básico del ADC conectando una señal variable mediante un potenciómetro y visualizando los resultados a través de la función read_u16(), comprendiendo la relación entre la resolución de 12 bits del ADC y el formato de 16 bits entregado por el microcontrolador. Posteriormente, en la segunda parte se trabajó con señales muestreadas, utilizando programas en MicroPython y Matlab para registrar, almacenar y analizar las muestras obtenidas, lo que permitió verificar la tasa de muestreo y la reconstrucción de señales. Finalmente, en la tercera parte se realizó un análisis estadístico aplicando voltajes constantes, calculando la media y la desviación estándar de los valores digitalizados, además de graficar histogramas para evaluar la distribución de datos. En conjunto, el laboratorio permitió afianzar conceptos de cuantización, muestreo y procesamiento digital de señales.**

***Abstract*— The A/D conversion laboratory with the Raspberry Pi Pico 2W allowed us to understand the operation of the analog-to-digital converter and its application in digital communication systems. First, a potentiometer was used to generate a variable voltage, and the read_u16() function was analyzed to understand the relationship between the 12-bit ADC resolution and the 16-bit value reported by the microcontroller. Then, signal sampling was performed using MicroPython and Matlab programs, verifying the sampling rate and comparing the original signals with their reconstruction from the collected data. Finally, a statistical analysis was carried out by applying different constant voltages, calculating the mean, standard deviation, and plotting histograms to evaluate the distribution of samples. This laboratory reinforced fundamental concepts such as quantization, sampling, and data analysis in digital signal processing.**

I. Introducción

La conversión analógica–digital (A/D) constituye uno de los procesos fundamentales dentro de la ingeniería electrónica y las telecomunicaciones, ya que permite transformar señales continuas, como el voltaje o la corriente, en representaciones digitales que pueden ser procesadas, almacenadas y transmitidas por sistemas digitales. Gracias a este procedimiento es posible integrar el mundo físico con el digital, lo que hace indispensable el estudio del funcionamiento de los conversores A/D en la formación de un ingeniero.

En este laboratorio se empleó la tarjeta de desarrollo Raspberry Pi Pico 2W, la cual dispone de un conversor A/D de 12 bits, con el fin de analizar de manera práctica los conceptos de cuantización, muestreo y procesamiento digital de señales. La práctica se dividió en tres fases: en la primera se exploró el uso básico del ADC, conectando un voltaje variable y analizando la función read_u16() para comprender cómo se representa un valor de 12 bits en un formato de 16 bits. En la segunda fase se estudió el muestreo de señales, verificando la relación entre frecuencia de muestreo, número de muestras por período y la reconstrucción de la señal mediante herramientas como Python y Matlab. Finalmente, en la tercera fase se desarrolló un análisis estadístico, en el que se aplicaron voltajes constantes y se calcularon parámetros como la media, la desviación estándar y la distribución de valores obtenidos.

La importancia de este laboratorio radica en que permitió conectar la teoría de la conversión A/D con su aplicación práctica en un microcontrolador, fortaleciendo las competencias en el manejo de hardware y software para el procesamiento digital de señales. Así, los estudiantes no solo verificaron el funcionamiento del ADC, sino que también comprendieron su papel en sistemas de comunicación modernos, donde la correcta digitalización de las señales garantiza la calidad y fidelidad en la transmisión y el análisis de la información.

II. Uso del conversor A/D.

Para el desarrollo de este laboratorio se empleó el código suministrado por el docente, al cual se le realizó una modificación relacionada con el voltaje de referencia. Para este ajuste, se midió el voltaje de salida en el pin 36 de la Raspberry Pi Pico 2W, obteniendo un valor de 3,317 voltios, tal como se muestra en la ilustración 1.

Ilustración 1 Voltaje de referencia.

Se realizará esta modificación en el código con el fin de obtener una mayor exactitud en el cálculo tanto del voltaje de salida como del valor digital code12 .

```python
import machine
import utime

ADC_PIN = 26      # GP26 -> ADC0 (cambia a 27 o 28 si usas ADC1/ADC2)
VREF    = 3.316   # mide tu 3V3 real y ajústalo aquí para mejor exactitud
PERIOD  = 0.02    # segundos entre lecturas (0.05–0.5 )

adc = machine.ADC(ADC_PIN)

while True:
    raw16  = adc.read_u16()      # 0..65535 (12 bits alineado a la izq.)
    code12 = raw16 >> 4          # 0..4095 (12 bits reales)
    volts  = (code12 * VREF) / 4095.0
    print(f"{volts:.4f}")        # <-- SOLO un número por línea
    utime.sleep(PERIOD)

```

Ilustración 2 Código 1.

Se verificó que la salida A/D, ubicada en el pin 31, entregara un valor de 3,316 voltios. Una vez realizada esta comprobación, se procedió a conectar una fuente de alimentación de corriente continua (DC) ajustada a 2 voltios, tal como se muestra en la ilustración 3.

Ilustración 3 Fuente DC.

En la parte inferior izquierda del software Thonny se observa un valor de 2.2 voltios. Este resultado varía debido al proceso de conversión A/D realizado por el microcontrolador.

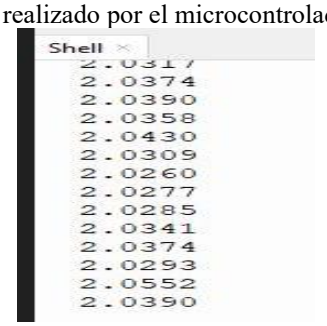

Ilustración 4 Voltaje A/D.

A continuación, se analiza el funcionamiento de la función **read_u16()**. Es importante recordar que en la Raspberry Pi Pico 2W el conversor realiza la conversión a 16 bits, aunque en realidad las muestras corresponden a una resolución de 12 bits. Para obtener el valor correcto, se utiliza el comando:

$$code12 = raw16 \gg 4$$

Este desplazamiento a la derecha elimina los 4 bits menos significativos, que son los de menor peso. De esta manera se recupera el valor real de 12 bits del ADC. Posteriormente, se procederá a calcular el valor de **code12** empleando la siguiente fórmula:

$$V = \frac{Code12 \times V_{REF}}{4095}$$

Ecuación 1.

Dado que contamos con V=2 y usando la ecuación (1)

$$V = \frac{Code12 \times V_{REF}}{4095}$$

despejamos code12 y remplazamos los valores

$$Code12 = \frac{2\ V}{3{,}317\ V} \times 4095 = 2469{,}10$$

Por lo tanto, el código de 12 bits esperado es code12≈2469 (decimal).

Si se requiere el valor de 16 bits entregado por read_u16(), entonces:

$$raw16 = code12 \ll 4 = 2469 \times 16 = 39504$$

A continuación, se activó el Plotter en el entorno de Thonny y se ajustó el parámetro PERIOD con valores de 0,05 y 0,5, con el fin de observar las diferencias en la visualización. El uso del Plotter permite representar gráficamente los valores obtenidos por el conversor A/D en función del tiempo, facilitando así el análisis de la señal. Tal como se muestra en la ilustración 4, los valores obtenidos no son fijos, sino que presentan variaciones constantes debido a la naturaleza del muestreo y la conversión.

Ilustración 5 periodo 0,05.

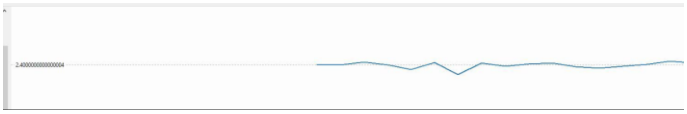

Ilustración 6 periodo 0,5.

Como se observa en las ilustraciones 5 y 6, a medida que el periodo se hace más pequeño, los cambios de voltaje se representan de manera más rápida en comparación con lo mostrado en la ilustración 4. Finalmente, se modificó el código para visualizar no solo el voltaje, sino también el número binario correspondiente a raw16 y el valor de code12 en formato binario. De esta manera fue posible diferenciar con mayor claridad el proceso de conversión A/D y comprender cómo el valor analógico se traduce en distintas representaciones digitales.

```
import machine
import utime

ADC_PIN = 26      # GP26 -> ADC0 (cambia a 27 o 28 si usas ADC1/ADC2)
VREF    = 3.316   # mide tu 3V3 real y ajústalo aquí para mejor exacti
PERIOD  = 0.9    # segundos entre lecturas (0.05-0.5 )
adc = machine.ADC(ADC_PIN)
while True:
    raw16  = adc.read_u16()      # 0..65535 (12 bits alineado a la izq
    code12 = raw16 >> 4          # 0..4095 (12 bits reales)
    volts  = (code12 * VREF) / 4095.0
    print(f"Voltaje:{volts:.4f}")        # <-- SOLO un número por líne
    print("---------")
    print(raw16)
    numero_binario1 = bin(raw16)[2:]  # Eliminar el prefijo '0b'
    print(f"El número binario 16 bits es: {numero_binario1}")
    print("---------")
    print(code12)
    numero_binario = bin(code12)[2:]  # Eliminar el prefijo '0b'
    print(f"El número binario 12 bits es: {numero_binario}")
    print("---------")
    utime.sleep(PERIOD)
```

Ilustración 7 Código modificado.

A continuación, se utilizarán los tres ejemplos presentados al final de la guía, correspondientes a los valores 2048, 1024 y 4095. Adicionalmente, se seleccionaron tres valores propios para el análisis: 512, 1500 y 1900. Estos casos permitirán comprender con mayor detalle cómo se realiza la conversión A/D, la relación entre el valor de 12 bits (code12) y el valor desplazado a 16 bits (raw16), así como la interpretación binaria de cada uno de ellos.

Vamos a usar la ecuación 1 para hallar los voltajes respecticos de estos Code12.

$$\frac{2048 \times 3{,}317}{4095} = 1{,}65\ V$$

$$\frac{1024 \times 3{,}317}{4095} = 0{,}82\ V$$

$$\frac{4095 \times 3{,}317}{4095} = 3{,}317\ V$$

$$\frac{512 \times 3{,}317}{4095} = 0{,}41\ V$$

$$\frac{1500 \times 3{,}317}{4095} = 1{,}21\ V$$

$$\frac{1900 \times 3{,}317}{4095} = 1{,}53\ V$$

De esta manera, para cada valor de Code12 (2048, 1024, 4095, 512, 1500 y 1900) se calculó el voltaje teórico asociado. Posteriormente, estos voltajes se configurarán en la fuente de poder de laboratorio y se compararán con las mediciones obtenidas a través del conversor A/D de la Raspberry Pi Pico 2W. Esta comparación permitirá verificar la exactitud del proceso de conversión y evaluar posibles diferencias entre los valores teóricos y los experimentales.

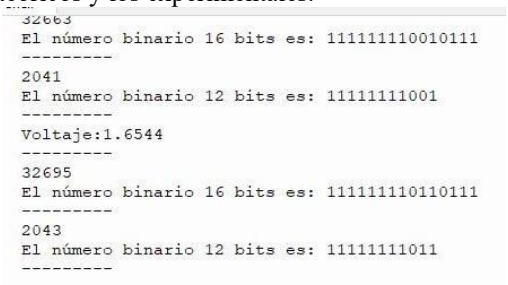

Ilustración 8 Code12 2048.

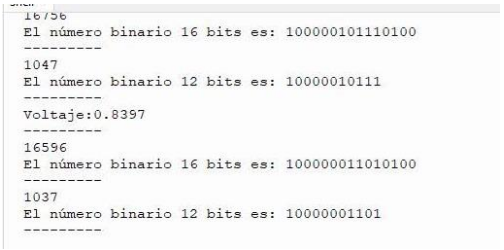

Ilustración 9 Code12 1024.

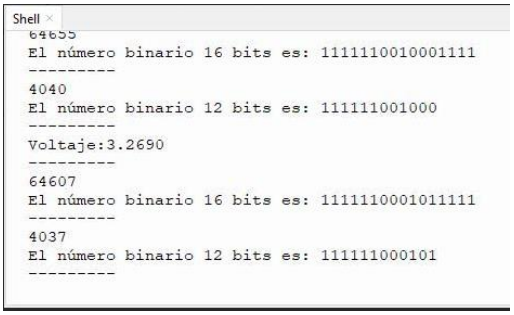

Ilustración 10 Code12 4095.

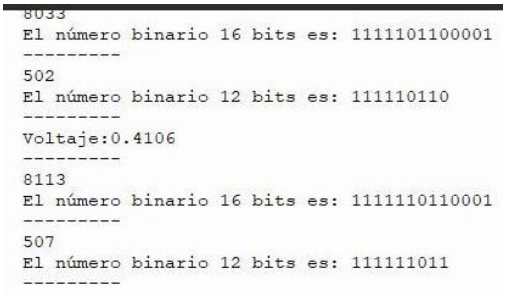

Ilustración 11 Code12 512.

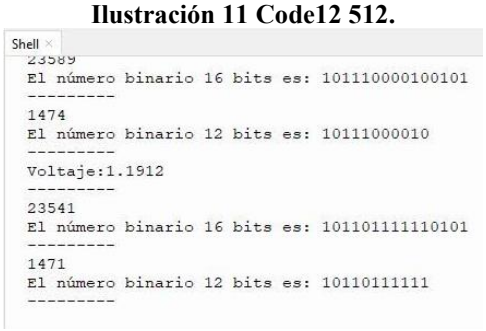

Ilustración 12 Code12 1500.

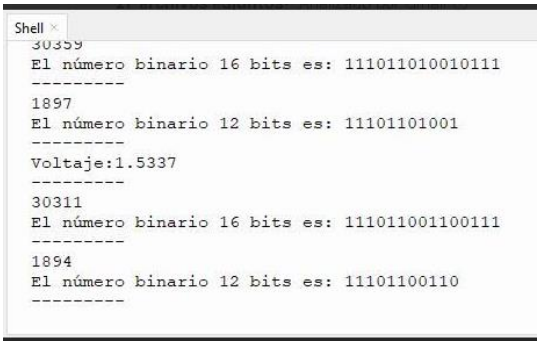

Ilustración 13 Code12 1900.

Como se aprecia en las ilustraciones 8 a 13, los valores obtenidos a partir de los diferentes ejemplos de conversión A/D se encuentran muy próximos a los voltajes teóricos calculados. Esto confirma que el método empleado para pasar de un número de 16 bits a su equivalente de 12 bits es correcto, aunque inevitablemente aparecen pequeñas diferencias. Dichas discrepancias se deben a que el voltaje aplicado nunca es totalmente constante, sino que siempre presenta ligeras fluctuaciones ocasionadas por el ruido eléctrico, las tolerancias de los componentes y la propia inestabilidad de la fuente de alimentación. En consecuencia, las mediciones nunca serán idénticas al valor teórico, sino que oscilarán alrededor de él.

En este sentido, se pudo observar que la función read_u16() entrega un valor de 16 bits (raw16), pero el conversor A/D de la Raspberry Pi Pico 2W realmente opera a 12 bits. Para recuperar este valor real se aplica un corrimiento de cuatro posiciones hacia la derecha (code12 = raw16 >> 4). Idealmente, los cuatro bits menos significativos deberían ser siempre cero; sin embargo, en la práctica esto no ocurre, ya que el proceso de conversión digital no es exacto y esos bits reflejan pequeñas variaciones debidas al ruido y al tiempo de muestreo. Aun así, una vez realizado el corrimiento, el valor de 12 bits obtenido se mantiene estable y permite calcular un voltaje muy cercano al real aplicado en la entrada.

Este análisis evidencia que la conversión A/D es confiable para representar señales analógicas dentro del rango de referencia, pero también demuestra que la digitalización nunca será perfecta, pues siempre existirán pequeñas desviaciones entre el valor teórico y el experimental. Dichas diferencias no representan un error grave, sino el comportamiento natural de un sistema de muestreo y cuantización, lo que resalta la importancia de comprender la resolución del conversor y la influencia de factores externos en las mediciones.

III. MUESTREO DE SEÑALES.

Para el desarrollo de esta parte del laboratorio se utilizaron dos archivos de código suministrados por el docente: uno ejecutado en Matlab y otro en Thonny. El primer archivo corresponde al programa implementado en Matlab, el cual se presenta a continuación:

```
f = 625;        % frecuencia de la señal (Hz)
T = 1/f;        % periodo (s)
A = 1.6;        % amplitud (V)
NT = 2;         % nº de periodos a capturar
N = 40;         % nº de muestras por periodo
ts = T/N;       % periodo de muestreo (s)

% Vector de tiempo para las muestras (N*NT puntos exactos)
t = 0:ts:(NT*T - ts);
V = A * sin(2*pi*f*t);  % señal muestreada

% Señal continua para comparación
t_cont = 0:ts/20:NT*T;
V_cont = A * sin(2*pi*f*t_cont);

% ===== Gráfica =====
figure('Position',[100 100 900 400]);  % tamaño ventana
plot(t_cont, V_cont, 'b-', 'LineWidth',2); hold on;
stem(t, V, 'r','filled','LineWidth',1.5);

xlabel('Tiempo (s)','FontSize',20,'FontWeight','bold');
ylabel('Amplitud (V)','FontSize',20,'FontWeight','bold');
title(sprintf('Muestreo de un seno f=%d Hz, N=%d muestras/periodo', f, N), ...
      'FontSize',20,'FontWeight','bold');

legend({'Señal continua','Muestras'},'FontSize',14,'Location','best');
grid on;
set(gca,'FontSize',16,'LineWidth',1.2);
```

Ilustración 14 Código adc.m

El código en Matlab tiene como objetivo mostrar el proceso de muestreo de una señal senoidal y compararlo con la señal continua original. Para ello, se define una señal de frecuencia 625 Hz y amplitud de 1,6 V, la cual se representa a lo largo de dos periodos. Cada periodo es discretizado en 40 muestras, obteniendo así una versión muestreada de la señal con un intervalo de muestreo de T/N.

Además, se genera un vector de tiempo mucho más denso para graficar la señal continua de referencia. En la figura resultante se observa la señal continua en color azul y, sobre esta, las muestras en color rojo mediante el comando stem. De esta manera, el código permite visualizar cómo el muestreo conserva la forma general de la onda original y resalta la importancia de la tasa de muestreo en la representación digital de una señal analógica. Este código nos genera la siguiente grafica.

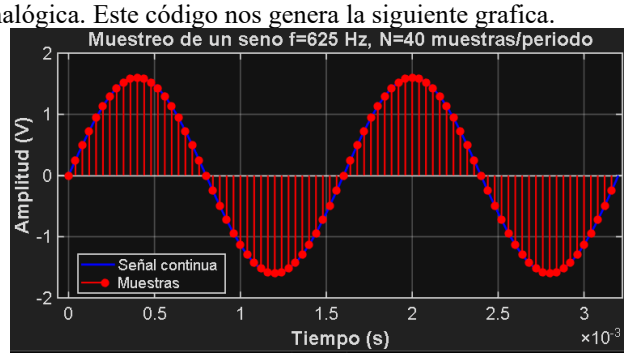

Ilustración 15 Grafico adc.m

Ahora vamos a ver una parte del segundo programa este tiene la función de muestrear una señal analógica conectada al pin GP26 (ADC0) de la Raspberry Pi Pico y almacenar los resultados en un archivo CSV para su posterior análisis. Para ello, configura parámetros como la frecuencia de la señal a medir (50 Hz), el número de muestras por periodo y la cantidad de periodos a capturar, lo que define la frecuencia de muestreo. Durante la ejecución, el microcontrolador obtiene valores del ADC con la instrucción read_u16(), los convierte a 12 bits, calcula el voltaje correspondiente y genera un registro en formato decimal, binario, hexadecimal y en voltios. Cada muestra se almacena junto con su tiempo relativo desde el inicio de la captura, lo que permite evaluar la tasa de muestreo real y compararla con la teórica. Además, el código controla un LED para indicar el estado de la captura y muestra en consola un reporte final con métricas como la frecuencia de muestreo alcanzada y posibles "overruns" (muestras tomadas tarde). En conclusión, este programa permite capturar y documentar digitalmente una señal analógica en tiempo real, facilitando su análisis en Matlab o Python.

```
import machine
import utime

ADC_PIN       = 26         # GP26 -> ADC0
VREF          = 3.300      # Mida en 3V3 (pin 36) y reemplace para m
SIGNAL_FREQ   = 50.0       # Hz
SAMPLES_PER_PERIOD = 20    # N muestras por periodo
PERIODS       = 2          # >= 2
OUT_CSV       = "adc_capture_2.csv"
LED_PIN       = "LED"      # LED integrado

FS_HZ   = SIGNAL_FREQ * SAMPLES_PER_PERIOD
TS_US   = int(1_000_000 / FS_HZ)          # periodo de muestreo (us
NSAMPS  = int(PERIODS * SAMPLES_PER_PERIOD)

adc = machine.ADC(ADC_PIN)
led = machine.Pin(LED_PIN, machine.Pin.OUT)

print("-----------------------------------------")
print("Captura ADC - N muestras por periodo (CSV limpio)")
print(f"f_señal = {SIGNAL_FREQ} Hz")
print(f"N/periodo = {SAMPLES_PER_PERIOD}")
print(f"Periodos a capturar = {PERIODS}")
print(f"Frecuencia de muestreo fs (teórica) = {FS_HZ:.1f} Hz")
print(f"Periodo de muestreo Ts (trunc) = {TS_US} us")
print(f"N total de muestras = {NSAMPS}")
print(f"Voltaje de referencia VREF = {VREF:.4f} V")
print("-----------------------------------------")
input("Presione ENTER para iniciar la captura...")

```

Ilustración 16 ADC_Sampling.

Ya sabiendo para que sirven ambos códigos vamos a prender el generador de señales y configurarlo una señal de 3Vpp, un offset de 1,6 y una frecuencia de 625 Hz, a esta señal se le agregar un offset ya que sabemos que el Pi Pico 2W solo maneja voltaje de 0 a 3,3 voltios , vamos a verificar en el osciloscopio .

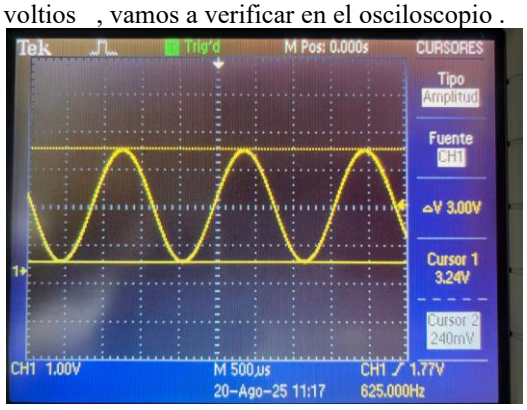

Ilustración 17 Verificación de osciloscopio.

Ahora vamos a concertar el generador de señales a las tierra y a el pin 36 del Pi Pico 2W, y le daremos entre para generar un archivo llamado "adc_capture_3.csv", vamos a abrir el archivo y mirar que obtenemos.

	n,t_us,dec,bin,hex,volts
2	0,564,3902,111100111110,0xF3E,3.144469
3	1,1676,3917,111101001101,0xF4D,3.156557
4	2,2434,3824,111011110000,0xEF0,3.081612
5	3,3159,3624,111000101000,0xE28,2.920440
6	4,4007,3313,110011110001,0xCF1,2.669817
7	5,5008,2827,101100001011,0xB0B,2.278168
8	6,6014,2275,100011100011,0x8E3,1.833333
9	7,7012,1732,011011000100,0x6C4,1.395751
10	8,8006,1207,010010110111,0x4B7,0.972674

Ilustración 18 adc_capture_3.

Como observamos en la ilustración 18, este nos generó un código , la primera nos muestra las 40 muestras, después no

muestra el tiempo de cada muestra y de ultima el voltaje, ahora vamos a configurar el código de Matlab

```
f = 20;         % frecuencia de la señal (Hz)
T = 1/f;        % periodo (s)
A = 1.6;        % amplitud (V)
NT = 2;         % nº de periodos a capturar
N = 40;         % nº de muestras por periodo
ts = T/N;       % periodo de muestreo (s)
dc = 1.5;       % componente DC (offset)
% Vector de tiempo para las muestras (N*NT puntos exactos)
t = 0:ts:(NT*T - ts);
V = dc + A * sin(2*pi*f*t);  % señal muestreada con DC
% Señal continua para comparación
t_cont = 0:ts/20:NT*T;
V_cont = dc + A * sin(2*pi*f*t_cont);
% ===== Gráfica con subplot =====
figure('Position',[100 100 900 700]);  % tamaño ventana más alto para dos plots
% Primer plot: señal continua y muestras
subplot(2,1,1) % 2 filas, 1 columna, 1er gráfico
plot(t_cont, V_cont, 'b-', 'LineWidth',2); hold on;
stem(t, V, 'r', 'filled', 'LineWidth',1.5);
xlabel('Tiempo (s)', 'FontSize',16, 'FontWeight','bold');
ylabel('Amplitud (V)', 'FontSize',16, 'FontWeight','bold');
title(sprintf('Muestreo de un seno con DC=%.1f V, f=%d Hz, N=%d muestras/periodo',
      'FontSize',18, 'FontWeight','bold');
legend({'Señal continua', 'Muestras'}, 'FontSize',14, 'Location','best');
grid on;
set(gca, 'FontSize',14, 'LineWidth',1.2);
% Segundo plot: muestras conectadas con línea
subplot(2,1,2) % 2 filas, 1 columna, 2do gráfico
plot(t1, v1, 'm-o', 'LineWidth',1.8, 'MarkerSize',6, 'MarkerFaceColor','m');
xlabel('Tiempo (s)', 'FontSize',16, 'FontWeight','bold');
ylabel('Amplitud (V)', 'FontSize',16, 'FontWeight','bold');
title('Muestras conectadas con línea', 'FontSize',18, 'FontWeight','bold');
grid on;
set(gca, 'FontSize',14, 'LineWidth',1.2);
```

Ilustración 19 Código adc.m modificado.

A este código se le agrego el offset, y un subplot() el primero grafica la señal teórica , el segundo subplot() es la reconstrucción a partir de archivo .csv que se mencionó antes, hicimos 3 pruebas, con 50 Hz y 40 muestras, 20 Hz y 40 muestras, 20 Hz y 200 muestras.

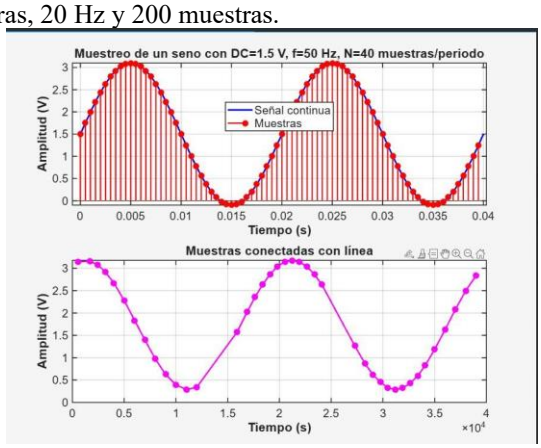

Ilustración 20 50 Hz, 40 muestras.

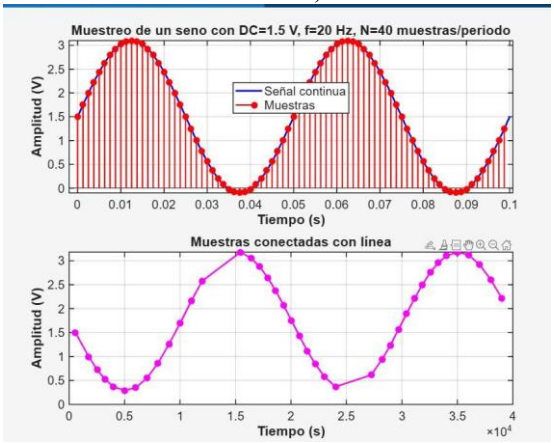

Ilustración 21 20 Hz, 40 muestras.

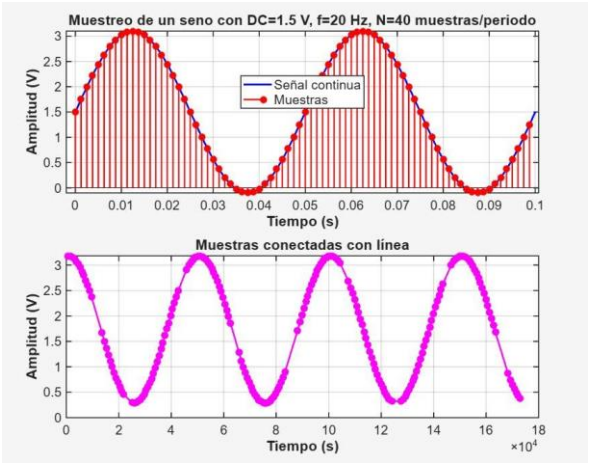

Ilustración 22 20 Hz , 200 muestras.

Para el caso de la señal de 50 Hz muestreada con 40 muestras por periodo, se observa que el intervalo de muestreo corresponde a

$$T_s = \frac{1}{50\ Hz \times 40} = 0{,}0005\ s$$

es decir, cada muestra se toma cada 500 µs. este es un valor teórico, ahora vamos a medir algunos puntos para sacar el tiempo experimental

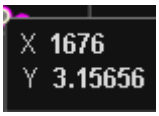

Ilustración 23 tiempo 2.

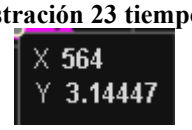

Ilustración 24 tiempo 1.

Estos dos tiempo los vamos a restar para hallar el tiempo de muestreo .

$$T_{SE} = 1676 - 564 = 1112\ \mu s$$

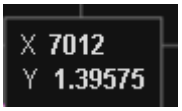

Ilustración 25 tiempo 4.

Ilustración 26 tiempo 3.

$$T_{SE} = 7012 - 6014 = 998\ \mu s$$

Este es maso menos el tiempo promedio se encuentras entre 1000 µs, ahora vamos a medir dos tiempo que está fuera de este rango que son los siguientes marcados en rojo.

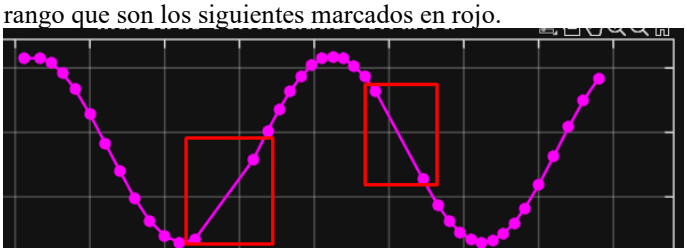

Ilustración 27 tiempo irregulares.

Vamos a medir estos tiempo y mirar si tiempo de muestreo.

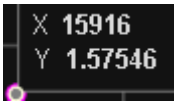

Ilustración 28 Tiempo 6.

Ilustración 29 Tiempo 5.

$$T_{SE} = 15916 - 12006 = 3610\ \mu s$$

Como vemos este tiene una irregular casi del cuatro veces el tiempo promedio. Ahora vamos a medir la otra irregularidad.

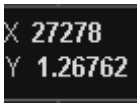

Ilustración 30 Tiempo 8.

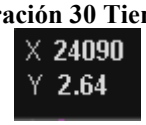

Ilustración 31 Tiempo 7.

$$T_{SE} = 15916 - 12006 = 3188\ \mu s$$

Durante la captura de la señal se midieron intervalos de tiempo entre muestras. El valor promedio esperado, de acuerdo con los parámetros de configuración, era de aproximadamente 1000 µs (1 ms). Sin embargo, en los resultados prácticos aparecieron valores de 3610 µs y 3188 µs en algunos registros. Estas diferencias se deben a factores propios del proceso de muestreo con el microcontrolador, como retardos en la ejecución del código, interrupciones internas del sistema y limitaciones en la sincronización de la instrucción utime.ticks_us(). Además, es importante resaltar que la Raspberry Pi Pico 2W no cuenta con un temporizador de alta precisión, por lo que siempre existirán pequeñas desviaciones en el tiempo de muestreo. Incluso al emplear equipos de mayor exactitud, es imposible eliminar totalmente el error, ya que todo sistema digital está sujeto a tolerancias y fluctuaciones en la captura de datos. A pesar de ello, al promediar los tiempos de adquisición, el sistema logra mantener la frecuencia de muestreo en un rango aceptable, lo que permite representar la señal con fidelidad suficiente para el análisis, ahora vamos analizar la formar de la señal.

La deformación que se observa en las señales muestreadas está directamente relacionada con la frecuencia de la señal respecto a la frecuencia de muestreo establecida en el sistema. En la práctica, aunque el criterio de Nyquist indica que la frecuencia de muestreo debe ser al menos el doble de la frecuencia de la señal para evitar aliasing, este valor mínimo no garantiza una representación fiel de la forma de onda. Para lograr una visualización adecuada se requiere un número mucho mayor de muestras por periodo, generalmente entre 10 y 40, dependiendo de la precisión deseada. Cuando la frecuencia de la señal aumenta y la frecuencia de muestreo no se incrementa de manera proporcional, el número de puntos disponibles para representar cada ciclo disminuye y la señal empieza a perder continuidad, mostrándose con formas dentadas, deformadas o incluso similares a ruido.

A este efecto se suman otros factores que acentúan la distorsión a frecuencias elevadas. Entre ellos se encuentran los errores de cuantización propios del conversor A/D, el jitter o variación en el tiempo exacto de muestreo, y el tiempo de apertura del ADC, que introduce imprecisiones en señales que cambian rápidamente. Además, la entrada analógica del microcontrolador tiene un ancho de banda finito y no siempre logra seguir variaciones de voltaje demasiado rápidas, lo que puede generar atenuaciones o formas no continuas en las muestras. Incluso el propio sistema de temporización de la Raspberry Pi Pico 2W presenta limitaciones, ya que no dispone de un reloj de alta precisión, de manera que siempre existirán pequeñas desviaciones en los instantes de captura.

En conclusión, la formación o deformación de la señal digitalizada depende principalmente de la relación entre la frecuencia de la señal y la frecuencia de muestreo, pero también está influenciada por errores inherentes al ADC y las limitaciones del hardware. Mientras más alta sea la frecuencia de la señal analógica sin que se ajusten proporcionalmente los parámetros de muestreo, más evidente será la pérdida de continuidad en la forma digitalizada, hasta el punto de que la representación puede asemejarse a ruido y dejar de reflejar correctamente la señal original.

Ahora vamos a analizar el grafico de la frecuencia de 20 Hz con 40 muestras.

Para el caso de la señal de 50 Hz muestreada con 40 muestras por periodo, se observa que el intervalo de muestreo corresponde a

$$T_s = \frac{1}{20\ Hz \times 40} = 0{,}000125\ s$$

es decir, cada muestra se toma cada 125 ms. este es un valor teórico, ahora vamos a medir algunos puntos para sacar el tiempo experimental.

Ilustración 32 20 Hz tiempo 1.

Ilustración 33 20 Hz tiempo 2.

Estos dos tiempo los vamos a restar para hallar el tiempo de muestreo .

$$T_{SE} = 1773 - 566 = 1207\ \mu s$$

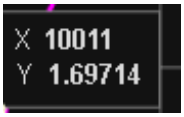

Ilustración 34 20 Hz tiempo 4.

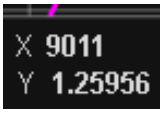

Ilustración 35 20 Hz tiempo 3.

$$T_{SE} = 10011 - 9011 = 1000\ \mu s \text{ o } 1ms$$

El tiempo promedio obtenido se encuentra aproximadamente en 1 ms. No obstante, se identificaron dos valores atípicos fuera de este rango, los cuales han sido marcados en rojo para su análisis. Es importante destacar que, incluso cuando la señal se muestrea a una frecuencia de 20 Hz, el tiempo promedio se mantiene constante, permaneciendo cercano a los 1 ms.

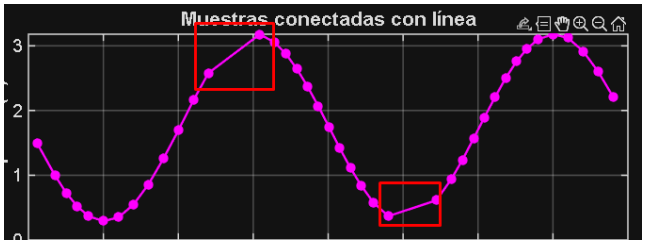

Ilustración 36 tiempo irregulares.

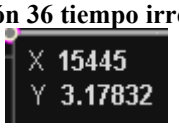

Ilustración 37 20 Hz tiempo 6.

Ilustración 38 20 Hz tiempo 5.

$$T_{SE} = 15445 - 12009 = 3536\ \mu s$$

Como vemos este tiene una irregular es de 3,5 veces el tiempo promedio. Ahora vamos a medir la otra irregularidad.

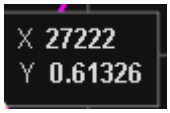

Ilustración 39 20 Hz tiempo 8.

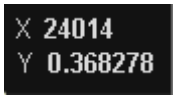

Ilustración 40 20 Hz tiempo 7.

$$T_{SE} = 27222 - 24014 = 3208\ \mu s$$

Como se mencionó en la muestra anterior, la aparición de tiempos irregulares en el registro puede deberse a distintos factores del proceso de muestreo, como retardos en la ejecución del código, interrupciones internas o limitaciones propias del microcontrolador. En este caso, al trabajar con una señal de 20 Hz, se observa el mismo comportamiento: algunos intervalos superan el tiempo esperado de 1 ms en la Raspberry Pi Pico 2W. Esto confirma que dichas irregularidades no dependen directamente de la frecuencia de la señal, sino de las condiciones inherentes al sistema de adquisición y a las tolerancias de su temporización. Por último, vamos a hacer la prueba con 20 Hz y 200 muestras, en este caso

$$T_s = \frac{1}{50\ Hz \times 40} = 0{,}00025\ s$$

es decir, cada muestra se toma cada 125 ms. este es un valor teórico, ahora vamos a medir algunos puntos para sacar el tiempo experimental.

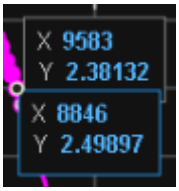

Ilustración 41 20 Hz tiempos.

$$T_{SE} = 9583 - 8846 = 737\ \mu s$$

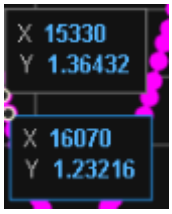

Ilustración 42 20 Hz tiempos 2.

$$T_{SE} = 16070 - 15330 = 740\ \mu s$$

Como observamos en la ilustración 41 y 42 , demás pruebas que hicimos el tiempo promedio son entre 700 µs y 750 µs , ahora vamos a sacar lo tiempo irregulares en esta ocasión son 7 tiempo irregulares.

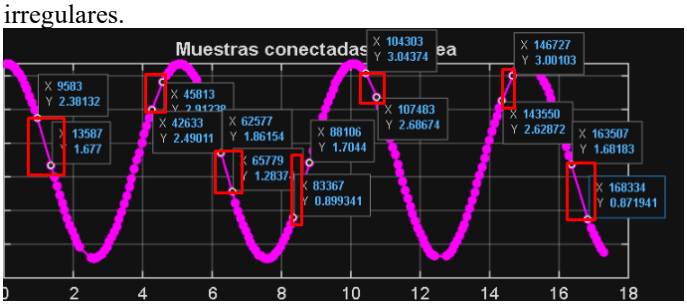

Ilustración 43 20 Hz tiempo irregulares.

$$T_{SE1} = 13587 - 9583 = 4004\ \mu s$$
$$T_{SE2} = 45813 - 42633 = 3180\ \mu s$$
$$T_{SE3} = 65779 - 62577 = 3202\ \mu s$$
$$T_{SE4} = 88106 - 83367 = 4739\ \mu s$$
$$T_{SE5} = 107483 - 104303 = 3180\ \mu s$$
$$T_{SE6} = 146727 - 143550 = 3177\ \mu s$$
$$T_{SE7} = 168334 - 163507 = 4827\ \mu s$$

Al realizar la captura de la señal de 20 Hz tomando 200 muestras, se observó que el tiempo promedio de adquisición se encuentra entre 700 µs y 750 µs, lo cual representa una reducción frente al valor de 1000 µs obtenido en pruebas anteriores con menos muestras. Este resultado confirma que el tiempo de muestreo depende directamente de la cantidad de datos adquiridos: a mayor número de muestras por ciclo, el sistema ajusta su desempeño y reduce el intervalo promedio entre cada captura.

No obstante, dentro del registro aparecieron varios valores irregulares que superan de manera significativa el rango esperado, alcanzando tiempos de hasta 4004 µs, 4739 µs y 4827 µs. Estos intervalos no corresponden al comportamiento experimental esperado y reflejan limitaciones propias de la Raspberry Pi Pico 2W, tales como retardos en la ejecución del código, interrupciones internas o la falta de un temporizador de alta precisión.

En conclusión, aunque el sistema logra mantener un tiempo de muestreo promedio más corto y acorde con el incremento de muestras, persisten irregularidades que introducen variaciones en algunos intervalos. Dichos valores atípicos evidencian que, aun con una mayor densidad de puntos por ciclo, el proceso de adquisición sigue estando condicionado por las características del hardware y por los factores de sincronización en el microcontrolador.

IV. Análisis estadístico.

En la última parte del laboratorio se utilizó una fuente de poder de corriente continua (DC). Es fundamental tener en cuenta que el voltaje suministrado no debe exceder los 3,3 V, ya que esto podría ocasionar daños en la Raspberry Pi Pico 2W. Para la

práctica se seleccionaron cinco niveles de tensión: 0,8 V, 1,4 V, 1,6 V, 2,0 V y 2,6 V.

En esta fase, el docente proporcionó un código que genera dos archivos: el primero contiene los valores de tiempo y voltaje, y el segundo incluye los valores del ADC y la frecuencia. Dichos datos se emplean para calcular la media y la desviación estándar. Además, el mismo código muestra los valores teóricos de media y desviación estándar, lo que permite comparar los resultados experimentales con los esperados.

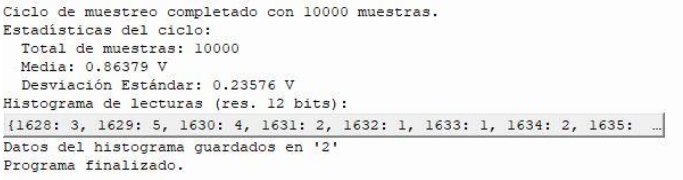

Ilustración 44 0,8 Voltios.

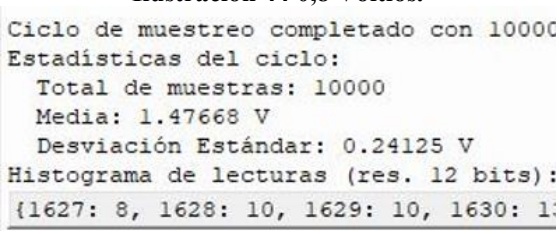

Ilustración 45 1,4 Voltios.

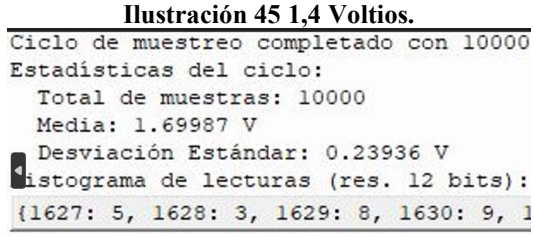

Ilustración 46 1,6 Voltios.

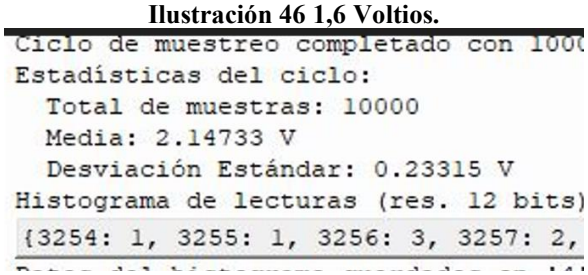

Ilustración 47 2 Voltios.

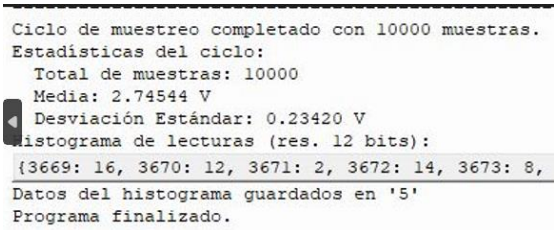

Ilustración 48 2,6 Voltios.

Con estos valores vamos a llenar la tabla teórica con las ilustración 44,45,46,47 y 48.

Test	V in(DC)	Media	Desviacíon estándar	Nombre de los archivos
1	0.8	0,86379	0,23576	0,8
2	1.4	1,47668	0,24125	1,4
3	1.6	1,69987	0,23936	1,6
4	2	2,14733	0,23315	2
5	2.6	2,74544	0,234200	2,6

Ilustración 49 Valores teóricos de desviación estándar y media.

Ahora debemos hacer un programa en Matlab para calcular la media y la desviación estándar de los programas

```
archivo = '0,8.csv';   % tu archivo
opts = detectImportOptions(archivo, 'Delimiter', '\t');
datos = readmatrix(archivo, opts);
% La segunda columna es la de voltajes
voltaje = datos(:,2);

% Calcular media y desviación estándar
media = mean(voltaje);
desviacion = std(voltaje);

% Mostrar resultados
fprintf('Media: %.4f\n', media);
fprintf('Desviacion estandar: %.4f\n', desviacion);
```

Ilustración 50 Código para Media y Desviación estándar.

Lo único que debemos hacer es cambiar en la línea 1 cambiar el nombre de los archivos, ahora vamos a mirar la media y la desviación estándar.

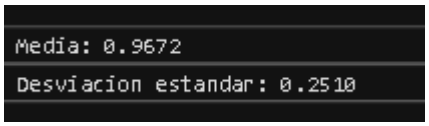

Ilustración 51 0,8 Voltios media y desviación estándar.

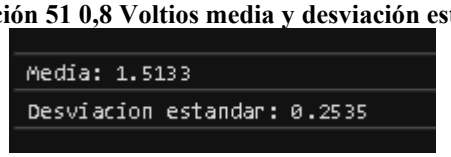

Ilustración 52 1,4 Voltios media y desviación estándar.

Ilustración 53 1,6 Voltios media y desviación estándar.

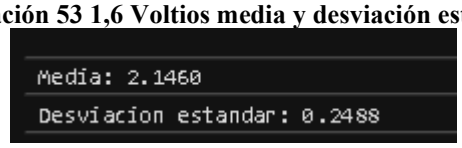

Ilustración 54 2 Voltios media y desviación estándar.

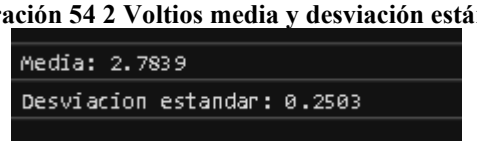

Ilustración 55 2,6 Voltios media y desviación estándar.

Ya tenemos los valores experimentales y teóricos, vamos a sacar los errores experimentales para esto usaremos la siguiente ecuación.

$$\%Error = \frac{|Valor\ teorico - Valor\ experimental|}{Valor\ teorico}$$

Ecuación 2.

Ahora vamos a remplazar los valores y sacar sus errores experimentales tan de la media como de la desviación estándar, primero vamos a empezar con el voltaje de 0.8.

$$\frac{|0.86379 - 0.9672|}{0.86379} = 11.9\%$$

Como vemos el error experimental de la media es del 11.9%.

$$\frac{|0.23576 - 0.2510|}{0.23576} = 6.4\%$$

El error experimental obtenido en la desviación estándar fue de 6,4 %. Posteriormente, se calcularán los errores correspondientes a todas las mediciones y, a partir de ellos, se realizará un análisis global de los resultados. Este procedimiento

se repetirá para cada uno de los valores de voltaje evaluados. Con el fin de evitar un exceso de figuras en el documento, se presentarán únicamente dos tablas de resumen: una con los valores de la media y otra con los de la desviación estándar.

Test	Media	Media Exp	Error experimental
1	0,86379	0,9672	11,97165978
2	1,47668	1,5133	2,479887315
3	1,69987	1,7835	4,919787984
4	2,14733	2,1460	0,061937383
5	2,74544	2,7839	1,400868349

Ilustración 56 error experimentales de la media.

Test	Desviacíon estándar	Desviacíon estándar exp	Error experimental
1	0,23576	0,2510	6,46420088
2	0,24125	0,2535	5,07772021
3	0,23936	0,254	6,11631016
4	0,23315	0,2488	6,7124169
5	0,234200	0,2503	6,87446627

Ilustración 57 error experimentales de desviación estándar.

En la tabla de medias se observa que, en general, los valores experimentales obtenidos para cada voltaje aplicado (0,8 V, 1,4 V, 1,6 V, 2,0 V y 2,6 V) se encuentran próximos a los teóricos esperados. No obstante, se aprecia que el mayor error relativo se presenta en el voltaje más bajo (0,8 V), mientras que para valores intermedios y altos la diferencia con respecto al teórico es menor. Esto puede explicarse porque, a tensiones reducidas, la cuantización del ADC tiene un impacto más significativo: unos pocos niveles digitales representan un mayor porcentaje del valor total, lo que incrementa la discrepancia.

Por otra parte, en la tabla de desviaciones estándar se evidencia que la dispersión de los datos alrededor de la media se mantiene baja en todos los casos, con un error experimental calculado en 6,4 %. Esto indica que, aunque existen variaciones, el sistema de muestreo es relativamente estable. Dichas variaciones se deben principalmente a factores inherentes al hardware, como ruido en las mediciones, limitaciones del conversor A/D y la ausencia de un temporizador de alta precisión en la Raspberry Pi Pico 2W.

En conjunto, los resultados muestran que los errores disminuyen a medida que se incrementa el voltaje aplicado, confirmando que el sistema se comporta de manera más precisa en rangos de tensión intermedios y altos, mientras que en valores bajos las limitaciones del ADC se hacen más evidentes.

Por último, se desarrolló un código en MATLAB con el objetivo de graficar los resultados obtenidos. Para ello, se generó un histograma a partir de los datos contenidos en el segundo archivo proporcionado por el código suplementado por el docente. El código utilizado es el siguiente:

```
datos = readmatrix('h0,8.csv');

valores = datos(:,1);

figure;
histogram(valores, 20);

title('Histograma de Voltajes Medidos');
xlabel('Voltaje [V]');
ylabel('Frecuencia');
grid on;
```

Ilustración 58 Código histograma.

A este código el único cambio que le debemos hacer es cambiar en la primera línea el nombre del archivo al cual necesitábamos generarle el histograma, ahora vamos a ver los histogramas de los 5 voltajes diferentes.

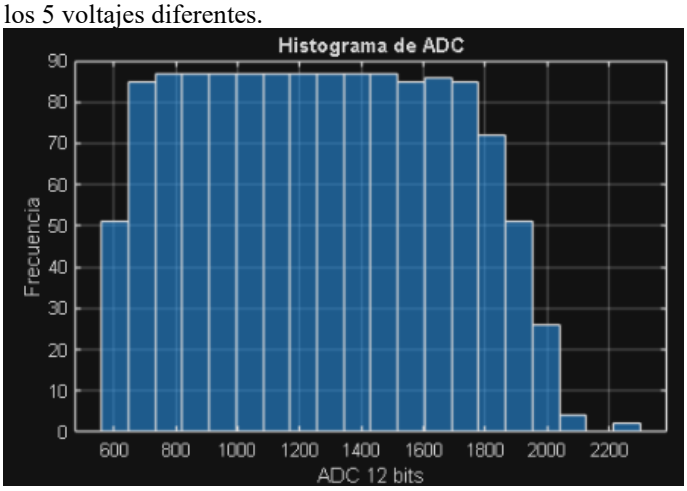

Ilustración 59 Histograma de 0,8.

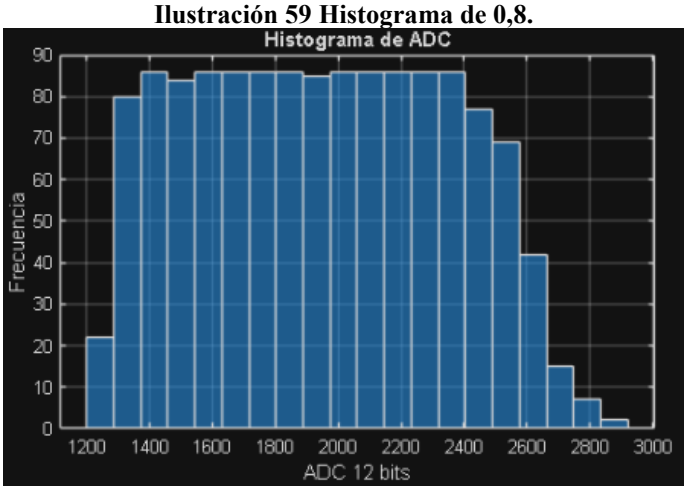

Ilustración 60 Histograma de 1,4.

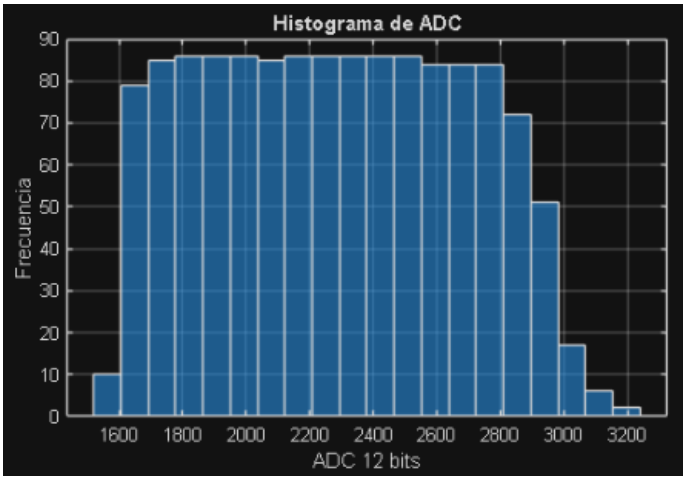

Ilustración 61 Histograma de 1,6.

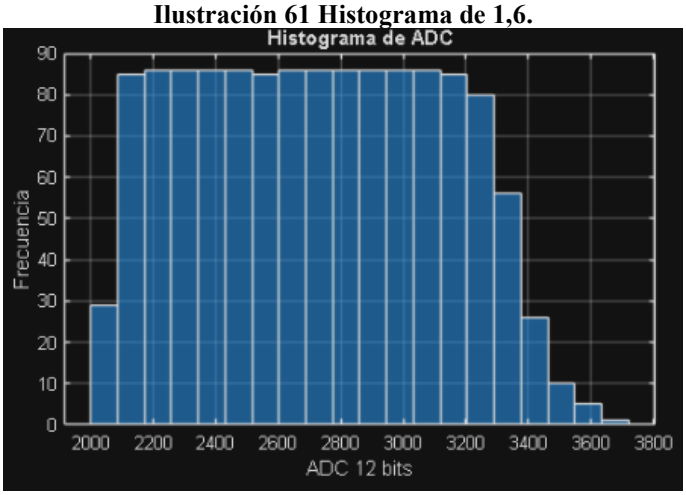

Ilustración 62 Histograma de 2.

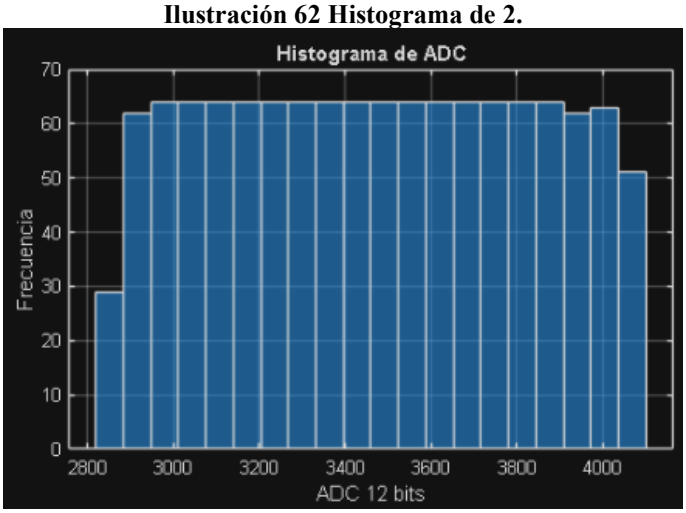

Ilustración 63 Histograma de 2.6.

En los histogramas obtenidos a partir de los valores registrados por el conversor analógico–digital (ADC) de la Raspberry Pi Pico 2W se observa una relación directa entre el voltaje aplicado y la distribución de los datos digitalizados.

Para un voltaje de 0,8 V, el histograma se concentra aproximadamente entre 600 y 2200 unidades del ADC. La dispersión es relativamente amplia debido a que, a bajos voltajes, la cuantización introduce un mayor porcentaje de error y la señal resulta más sensible al ruido. En el caso de 1,4 V, el rango se desplaza hacia valores más altos, ubicándose entre 1200 y 2900 unidades. Aunque la distribución se mantiene densa en la parte central, aún se evidencia cierta dispersión hacia los extremos.

Con 1,6 V, la señal presenta un rango entre 1600 y 3200 unidades, mostrando una concentración de valores más uniforme que en los casos anteriores, lo cual refleja mayor estabilidad en la conversión. Para 2,0 V, el histograma se distribuye entre 2000 y 3700 unidades, con un comportamiento más regular y menor dispersión relativa respecto al voltaje aplicado, evidenciando una buena linealidad del ADC en este rango. Finalmente, con 2,6 V, el histograma se extiende entre 2800 y 4100 unidades, presentando una concentración bien definida en el centro. Aunque aún existe cierta dispersión, la respuesta del ADC mantiene la proporcionalidad esperada con respecto al voltaje aplicado.

En términos generales, se observa que, al incrementar el voltaje de entrada, el histograma se desplaza progresivamente hacia valores más altos del ADC, lo cual concuerda con su resolución de 12 bits (rango de 0 a 4095). Además, la dispersión relativa tiende a disminuir conforme el voltaje aplicado aumenta, lo que indica que a bajos niveles de voltaje el ruido y las pequeñas fluctuaciones tienen un mayor impacto en la digitalización.

V. Análisis.

El desarrollo del laboratorio permitió examinar de manera integral el funcionamiento del conversor analógico–digital (ADC) de la Raspberry Pi Pico 2W, así como su desempeño en distintos escenarios experimentales. A partir de los resultados obtenidos en las tres fases de la práctica, se logró identificar tanto el comportamiento esperado de un sistema de muestreo y cuantización, como las limitaciones prácticas impuestas por el hardware y el entorno de ejecución.

En la primera parte, enfocada en la verificación básica del ADC, se pudo comprobar la relación entre la resolución de 12 bits del conversor y los valores de 16 bits reportados por la función read_u16(). La aplicación del corrimiento de cuatro posiciones hacia la derecha permitió obtener el valor code12, el cual reflejó de manera precisa el voltaje aplicado en el pin de entrada. Al comparar los valores teóricos con los experimentales se encontraron pequeñas discrepancias, atribuibles principalmente al ruido eléctrico, a la inestabilidad de la fuente de alimentación y a las tolerancias inherentes de los componentes. Este comportamiento evidenció que la conversión digital nunca es completamente exacta, sino que siempre está acompañada de ligeras fluctuaciones, lo cual corresponde al funcionamiento natural de cualquier sistema de adquisición de datos.

En la segunda parte del laboratorio, orientada al estudio del muestreo de señales, se realizaron pruebas con señales senoidales de diferentes frecuencias y distintos números de muestras por periodo. Para el caso de 50 Hz y 40 muestras, se obtuvo un tiempo de muestreo promedio cercano a 1 ms, aunque se detectaron intervalos irregulares que multiplicaban hasta por cuatro este valor esperado. Un fenómeno similar se observó al trabajar con 20 Hz y 40 muestras, lo cual confirma que estas irregularidades no dependen directamente de la frecuencia de la señal, sino de factores propios de la Raspberry Pi Pico 2W, como retardos en la ejecución del código, interrupciones internas o la ausencia de un temporizador de alta precisión.

Cuando se incrementó el número de muestras a 200 en la señal de 20 Hz, se evidenció una mejora en la resolución temporal, ya

que el tiempo promedio de muestreo se redujo a un rango entre 700 µs y 750 µs. Esto permitió representar la señal con mayor fidelidad y continuidad, aunque persistieron varios valores atípicos que superaron los 3 y 4 ms. Estos resultados demostraron que, aunque aumentar la cantidad de muestras por periodo mejora la calidad de la digitalización, no elimina por completo las limitaciones derivadas del hardware y de la gestión del tiempo en el microcontrolador. Además, se corroboró que la representación digital de una señal no depende únicamente del criterio de Nyquist, sino que también requiere un número suficiente de muestras (entre 10 y 40 por ciclo) para garantizar una reconstrucción adecuada, evitando así deformaciones que pueden asemejarse a ruido.

La tercera parte del laboratorio se enfocó en un análisis estadístico de la conversión A/D. Para ello, se aplicaron cinco niveles de tensión continua: 0,8 V, 1,4 V, 1,6 V, 2,0 V y 2,6 V. Los resultados mostraron que la media experimental se aproximó de manera satisfactoria a los valores teóricos, aunque con un error más elevado en el voltaje más bajo (11,9 % para 0,8 V). Este comportamiento se explica porque, a tensiones reducidas, el efecto de la cuantización es proporcionalmente mayor: unos pocos niveles del ADC representan un porcentaje significativo del valor total, amplificando así las discrepancias. A medida que el voltaje aplicado aumentó, los errores relativos disminuyeron, lo que confirma que el ADC presenta un desempeño más preciso en rangos de tensión intermedios y altos.

En cuanto a la desviación estándar, se obtuvo un error experimental promedio del 6,4 %, lo cual indica que el sistema es relativamente estable en sus mediciones. No obstante, la dispersión registrada en cada voltaje evidenció la influencia de factores como el ruido, el jitter en el muestreo y la propia resolución del conversor. Los histogramas generados a partir de los datos permitieron visualizar de manera gráfica la correspondencia entre el voltaje aplicado y la distribución de los valores digitales. En todos los casos se observó un desplazamiento progresivo hacia valores más altos del ADC conforme aumentaba el voltaje, confirmando la linealidad del conversor dentro de su rango operativo. Asimismo, se evidenció que la dispersión relativa es mayor a bajos voltajes y tiende a reducirse conforme el nivel de tensión se incrementa, lo que valida la hipótesis sobre la influencia de la cuantización y el ruido en los valores bajos de entrada.

En conjunto, el análisis experimental permitió corroborar los principios teóricos de la conversión A/D y del muestreo de señales, mostrando a la vez las limitaciones prácticas de los sistemas digitales de adquisición. El uso combinado de herramientas como MicroPython y Matlab facilitó la captura, procesamiento y visualización de los datos, integrando así los aspectos teóricos con su aplicación práctica en un entorno real.

VI. Conclusiones.

El laboratorio permitió comprender de manera integral el proceso de conversión analógica–digital, destacando cómo la Raspberry Pi Pico 2W traduce señales analógicas en valores discretos de 12 bits. Se confirmó que, aunque la función read_u16() entrega datos en 16 bits, el corrimiento de cuatro posiciones hacia la derecha es necesario para obtener el valor real (*code12*), el cual guarda una relación directa con el voltaje de entrada.

Las pruebas demostraron que el proceso de conversión nunca es completamente exacto, ya que siempre existen pequeñas discrepancias entre los valores teóricos y los experimentales. Estas diferencias se deben a factores como ruido eléctrico, tolerancias de los componentes, limitaciones del ADC y fluctuaciones en la fuente de alimentación. Dichas variaciones no representan fallos graves, sino que forman parte del comportamiento natural de los sistemas de cuantización.

En el estudio del muestreo de señales se verificó la influencia directa de la frecuencia de muestreo y del número de muestras por periodo sobre la fidelidad de la representación digital. Cuando se muestrearon señales de 50 Hz y 20 Hz con 40 muestras, el sistema mantuvo un tiempo promedio cercano a 1 ms; sin embargo, aparecieron intervalos irregulares que multiplicaban este valor hasta por cuatro veces. Al aumentar la cantidad de muestras a 200, se redujo el tiempo promedio a 700–750 µs y mejoró la continuidad de la señal reconstruida, aunque no desaparecieron por completo los valores atípicos. Esto demuestra que la calidad de la digitalización depende tanto del cumplimiento del criterio de Nyquist como de la densidad de puntos por ciclo.

El análisis estadístico realizado con voltajes constantes mostró que la media experimental se aproxima a los valores teóricos, con mayor error relativo en el voltaje más bajo (0,8 V), lo que confirma que la cuantización tiene un impacto más significativo en niveles reducidos de tensión. A medida que el voltaje aumentó, los errores relativos disminuyeron, evidenciando un mejor desempeño del ADC en rangos medios y altos.

La desviación estándar presentó un error experimental promedio del 6,4 %, lo que indica que, aunque existen variaciones en las mediciones, el sistema mantiene una estabilidad aceptable. Estas variaciones se atribuyen a factores como jitter en el muestreo, ruido en la tarjeta y la ausencia de un temporizador de alta precisión en la Raspberry Pi Pico 2W.

Los histogramas generados para los cinco niveles de voltaje aplicados confirmaron la proporcionalidad entre el voltaje de entrada y la distribución de valores digitales. Asimismo, mostraron que la dispersión relativa disminuye a medida que el voltaje aumenta, lo cual refuerza la conclusión de que la cuantización y el ruido afectan con mayor intensidad a valores bajos de tensión.

En términos generales, el laboratorio permitió integrar conceptos teóricos de cuantización, muestreo, análisis estadístico y procesamiento digital con la práctica experimental en un microcontrolador real. Esto no solo reforzó la comprensión de la conversión A/D, sino que también resaltó la importancia de considerar las limitaciones prácticas del hardware al momento de trabajar con sistemas de adquisición de señales en entornos de telecomunicaciones y electrónica aplicada.

Referencias

[1] Raspberry Pi Ltd., “Raspberry Pi Pico W Datasheet”, 2022. [Online]. Available: https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf

[2] Raspberry Pi Ltd., “Getting Started with Raspberry Pi Pico”, 2021. [Online]. Available: https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf

[3] B. Razavi, Design of Analog CMOS Integrated Circuits, McGraw-Hill, 2001. (Referencia clásica sobre jitter, temporización y errores de muestreo).

[4] A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing, 3rd ed., Pearson, 2010. (Fundamento de muestreo, criterio de Nyquist y distorsiones).

[5] Texas Instruments, “Understanding Data Converters”, 1995. (Explica errores de cuantización, jitter y limitaciones del ADC).
[6] A. Maxim, “ADC Sampling and Aliasing”, Maxim Integrated, Application Note 634. [Online]. Available: https://www.analog.com/media/en/technical-documentation/application-notes/636.pdf
[7] ARM Ltd., “ARM Cortex-M0+ Technical Reference Manual”, 2011. [Online]. Available: https://developer.arm.com/documentation/ddi0484/c