COMUNICACIÓN DIGITAL – LABORATORIO 1 (septiembre de 2025)

Fernando Javier Riaño Rios
Est.fernando.riano@unimilitar.edu.co

Resumen - Este laboratorio tuvo como propósito analizar el proceso de adquisición y digitalización de señales analógicas utilizando la Raspberry Pi Pico 2W. En una primera etapa se empleó un programa base suministrado por el docente, el cual permitió muestrear una señal senoidal de 200 Hz, 1.2 Vpp y una componente DC de 1.6 V, para posteriormente aplicar la Transformada Rápida de Fourier (FFT) y observar su comportamiento en el dominio de la frecuencia. En una segunda etapa se diseñó un programa alternativo que incorporó un esquema de muestreo más estable, mediante el uso de temporizadores, con el fin de reducir el jitter presente en el método inicial basado en retardos por software. A partir de las gráficas obtenidas y los cálculos realizados, se compararon los resultados experimentales con los valores teóricos, analizando cómo la variación de la frecuencia de la señal y la cantidad de puntos de muestreo influye en la representación temporal y espectral de la misma.

Abstract – This laboratory aimed to analyze the process of acquisition and digitization of analog signals using the Raspberry Pi Pico 2W. In the first stage, a base program provided by the instructor was used to sample a 200 Hz sinusoidal signal with 1.2 Vpp and a 1.6 V DC component, which was later processed with the Fast Fourier Transform (FFT) to observe its behavior in the frequency domain. In the second stage, an alternative program was designed that implemented a more stable sampling scheme by using hardware timers, in order to reduce the jitter present in the initial method based on software delays. Based on the obtained graphs and calculations, the experimental results were compared with the theoretical values, analyzing how the variation in signal frequency and the number of sampling points affect the temporal and spectral representation of the signal.

Palabras clave - RASPBERRY PI PICO 2W, MUESTREO, FFT, VENTANA HANNING, JITTER, ADC, SEÑALES DIGITALES.

INTRODUCCIÓN

En el área de las comunicaciones digitales y del procesamiento de señales, la digitalización constituye un paso esencial para la manipulación, análisis y transmisión de información. El muestreo periódico de una señal analógica y su posterior análisis en frecuencia mediante la Transformada Rápida de Fourier (FFT) son procedimientos básicos para comprender el comportamiento espectral de las señales. No obstante, la calidad de este proceso depende en gran medida de factores como la frecuencia de muestreo, la estabilidad temporal, la presencia de jitter y el uso de ventanas espectrales para mitigar la fuga de energía.

El presente laboratorio se realizó con la Raspberry Pi Pico 2W y constó de dos fases principales. En la primera, se utilizó un programa proporcionado por el docente para la adquisición de muestras de una señal senoidal y su análisis en frecuencia. Este código base empleaba retardos en microsegundos para controlar la frecuencia de muestreo, lo cual introducía inestabilidad temporal. En la segunda fase, se desarrolló un programa alternativo que implementó un esquema de muestreo más estable utilizando temporizadores de hardware, con el propósito de mejorar la precisión del muestreo y reducir el efecto del jitter.

Los resultados permitieron comparar las ventajas y limitaciones de ambos métodos de adquisición, evaluar la influencia de la ventana Hanning en la reducción de la fuga espectral, y analizar cómo la variación de la frecuencia de la señal y el número de puntos de muestreo afectan la calidad de la representación temporal y espectral. De esta manera, se consolidaron conceptos fundamentales de muestreo y análisis espectral, al tiempo que se exploraron las capacidades y restricciones del ADC integrado en la Raspberry Pi Pico 2W en un contexto práctico de laboratorio.

MARCO TEORICO

Muestreo y Teorema de Nyquist

El muestreo es el proceso mediante el cual una señal analógica se convierte en una secuencia de valores discretos en el tiempo para ser procesada digitalmente. El teorema de Nyquist establece que la frecuencia de muestreo (fs) debe ser al menos el doble de la frecuencia máxima presente en la señal para evitar el aliasing. En este laboratorio, al trabajar con señales de hasta 1800 Hz, se deberia garantizar que fs>3600 Hz, razón por la cual se recomienda utilizar tasas de muestreo de 10 kHz o superiores para asegurar un muestreo confiable.

¿Cómo se establece la tasa de muestreo en el programa?

La tasa de muestreo se intenta establecer usando utime.sleep_us(dt_us) dentro del bucle de adquisición, donde dt_us es el periodo deseado calculado a partir de f_muestreo. Sin embargo, la tasa real (fs_real) se determina de manera experimental midiendo el tiempo total que toma adquirir N muestras y calculando N / tiempo_total. Esta fs_real es la que

se usa para todos los cálculos posteriores (FFT, frecuencias, etc.).

Conversión Analógico-Digital en la Raspberry Pi Pico 2W

La Raspberry Pi Pico 2W incorpora un convertidor analógico-digital (ADC) de 12 bits que entrega valores escalados a 16 bits mediante la función `read_u16()` de MicroPython. Este ADC permite convertir señales en el rango de 0–3.3 V, limitación que obliga a ajustar el nivel DC y la amplitud de las señales de entrada para evitar saturación. La tasa de muestreo en el código proporcionado por el docente se establece mediante retardos (`sleep_us`) que definen el tiempo entre cada lectura, lo que produce una frecuencia de muestreo aproximada, pero con variaciones temporales (jitter).

Ventana de Hanning.

Cuando se aplica la Transformada Rápida de Fourier (FFT), es común que se presente fuga espectral debido a que la señal muestreada no siempre contiene un número entero de periodos dentro de la ventana de análisis. Para reducir este efecto se utilizan funciones ventana, como la ventana de Hanning. Esta multiplica las muestras por un perfil cosenoidal que atenúa los extremos de la señal, disminuyendo la dispersión de energía en el espectro y facilitando la identificación de la frecuencia dominante.

La ventana Hann/Hanning reduce fuga espectral: $w[n]=0.5-0.5\cos(2\pi n/(N-1))$. Se usa antes de la FFT multiplicando muestra a muestra

Jitter en el Muestreo

El jitter es la variación aleatoria o sistemática en los intervalos de muestreo respecto a los instantes ideales. En sistemas digitales como la Pico 2W, este fenómeno se presenta principalmente cuando se utilizan retardos por software, debido a la latencia del intérprete y a la falta de sincronización con un reloj de hardware. El jitter afecta directamente la precisión espectral, provocando ensanchamiento de los picos en la FFT y errores en la estimación de parámetros como la amplitud y la SNR. Su medición se realiza evaluando la desviación estadística de los intervalos entre muestras, expresada en términos de jitter pico a pico o jitter RMS.

Aumenta el ruido de fase y ensancha/distorsiona picos en la FFT (afecta más a frecuencias altas y FFT grandes). En MicroPython, el intérprete, interrupciones y GC añaden jitter; con ADC+DMA lo minimizas.

Métodos de Muestreo en la Pico 2W

- Software con retardos (sleep_us): sencillo de implementar, pero poco estable debido al jitter.
- Temporizadores de hardware (Timers): permiten establecer un periodo de muestreo más regular al disparar interrupciones periódicas para la lectura del ADC.
- ADC en modo free-running con DMA (en C/SDK): la opción más precisa, pues libera al procesador y garantiza intervalos constantes de muestreo con tasas de hasta 500 kS/s.
- Uso de librerías optimizadas (ulab en MicroPython): facilita cálculos de FFT en el dispositivo con mayor rapidez, aunque la limitación principal sigue siendo la adquisición estable de datos.

Alternativas en Pico 2W para “muestreo exitoso:

- MicroPython + ulab para FFT rápida en placa.
- SDK C con ADC free-running + DMA (mejor estabilidad).
- Mantener buen acondicionamiento de señal (offset, limitar 0–3.3 V, filtrar alias).

Explicación Código sugerido:

```
from machine import ADC, Pin
import utime, array, math, cmath
```

Importas periféricos (ADC y Pin), tiempo en microsegundos (utime), arreglos eficientes (array), y matemáticas complejas para la FFT manual.

```
def acquire_data():
    tiempos = []
    muestras = []
    start = utime.ticks_us()

    for i in range(N):
        t_actual = utime.ticks_diff(utime.ticks_us(), start) / 1_000_000
        tiempos.append(t_actual)
        muestras.append(adc.read_u16())
        utime.sleep_us(dt_us)
```

- N = cuántas muestras reales tomas en el tiempo.
-
- N_FFT = tamaño de la FFT; si es mayor que N, se completa con ceros (zero-padding) → la curva en frecuencia se ve más “suave”, pero la resolución real la da N/fs (duración del bloque), no el zero-padding.
- Adquisición “a mano”: en cada iteración lees el ADC, guardas un timestamp (segundos) y luego duermes dt_us.
- Esto no garantiza intervalos exactos (hay jitter por el intérprete + costo de read_u16() + latencia de sleep_us()).

```
elapsed_time = tiempos[-1]
fs_real = N / elapsed_time
...
with open("muestras.txt","w") as f:
    f.write("Tiempo(s)\tVoltaje(V)\n")
    voltajes = [(x/65535)*3.3 for x in muestras]
    ...
```

- Estimas f_s real como N/tiempo total (correcto).
- Guardas archivo (tiempo, voltaje); conviertes cuentas ADC (0..65535) a voltios asumiendo 3.3 V como referencia.

```
def convert_to_voltage(data, VREF=3.3): ...
def remove_offset(data): ...
```

Pasas a **V** y eliminas **DC** restando el promedio (lo imprime).

```
def apply_hanning_window(data):
    window = [0.5*(1 - cos(2π i/(N-1)))]
    return [d*w ...]
```

Aplica ventana Hann para reducir fuga espectral. la Hann atenúa la amplitud (ganancia coherente ≈ 0.5).

```
def fft_manual(x, N_FFT):
    # Radix-2: bit-reversal + etapas mariposa
```

- Implementación manual de FFT radix-2 con bit-reversal y mariposas.
- Construye X (complejos), permuta por bit-reversal y hace las etapas con twiddle factors.

```
def analyze_fft(fft_result, fs_real, N_FFT):
    magnitudes = [abs(c)/(N_FFT/2) ...]
    frequencies = [i*fs_real/N_FFT ...]
    max_index = magnitudes.index(max(magnitudes[1:]))
    dominant_freq = frequencies[max_index]
    signal_amplitude = magnitudes[max_index]
```

- Forma eje de frecuencias [0, fs/2).
- Busca pico dominante (omite DC).
- Estima amplitud (en V “a FS”, pero se usó Hann se debe compensar 0.5 si se quiere la amplitud real de la senoidal).

```
noise_magnitudes = magnitudes[:]; noise_magnitudes[max_index] = 0
noise_floor_v = sum(noise_magnitudes) / (N_FFT//2 - 1)
SNR = 20*log10(signal_amplitude / noise_floor_v)
ENOB = (SNR - 1.76) / 6.02
# guarda "fft.txt" (f, magnitud)
```

- Calcula un piso de ruido promedio, un SNR y una ENOB aproximada.
- Guarda la FFT en archivo.

```
def main():
    muestras, fs_real = acquire_data()
    voltajes = convert_to_voltage(muestras)
    voltajes_sin_offset = remove_offset(voltajes)
    senal_ventaneada = apply_hanning_window(voltajes_sin_offset)
    fft_result = fft_manual(senal_ventaneada, N_FFT)
    analyze_fft(fft_result, fs_real, N_FFT)
```

Flujo completo: adquirir → pasar a V → quitar DC → **Hann** → FFT → analizar y exportar.

DESARROLLO DEL LABORATORIO

Montaje: Se conectó la salida del generador de funciones a la entrada GP27 (ADC1) de la Raspberry Pi Pico 2W, compartiendo GND entre equipos. La señal se configuró en 200 Hz, 1.2 Vpp con offset DC = 1.6 V. Verificación. Con el osciloscopio se validaron los niveles y la forma de onda.

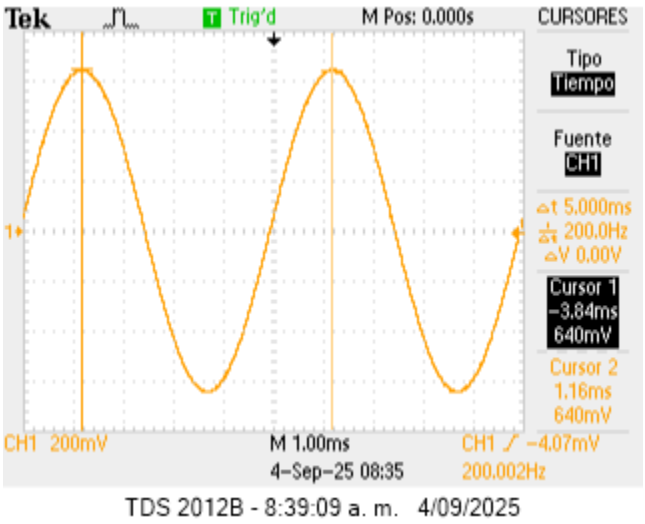

Ilustración 1 forma de onda, señal senoidal entregada.

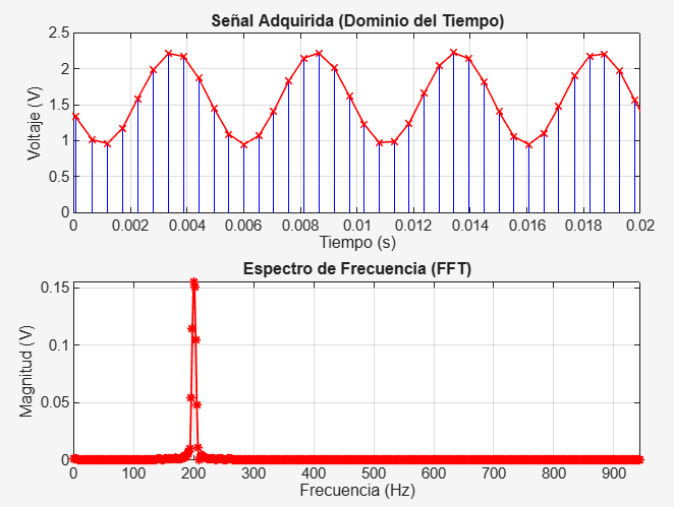

Ilustración 2 grafica de Matlab datos exportados de Tonny Python.

La diferencia entre los picos máximos (~2.2V) y mínimos (~1.0V) es de 1.2V, que es el voltaje pico-pico configurado.

El código adquirió un número suficiente de puntos (N=512) para reconstruir la forma de la onda senoidal de 200 Hz de manera clara y suave.

El código guarda los datos crudos (con DC) pero para su análisis interno los procesa (remueve DC, aplica ventana). Para analizar (cálculo de FFT y SNR), el código usa una versión sin DC para evitar que el pico de 0Hz oculte la señal de interés y arruine los cálculos de ruido.

Pico claro y dominante en 200 Hz, confirma que la frecuencia fundamental de la señal se identificó correctamente en el análisis espectral.

Posteriormente se Cambia el número de puntos para la FFT en el programa, por N_FFT = (64,128,256,512,1024,2048) con una frecuencia de 100 hz.
Se guardan los archivos correspondientes para cada uno de estos casos en formato .txt y se procede a graficar en Matlab los datos exportados.

```
LAB1C2ARC1.m | LAB1C2100.m | LAB1C2100TIEM.m | LAB1C2600.m | LAB1C21500.m | LAB1C2SEGUNDAPARTE.m
ATLAB Drive/LAB1C2100.m
clc; clear; close all;

% --- Archivos FFT a graficar ---
files = { ...
    'fft100_64.txt', ...
    'fft100_128.txt', ...
    'fft100_256.txt', ...
    'fft100_512.txt', ...
    'fft100_1024.txt', ...
    'fft100_2048.txt' };

nffts = [64 128 256 512 1024 2048];   % Para los títulos

% Layout 2x3
tiledlayout(2,3,'Padding','compact','TileSpacing','compact');
set(gcf,'Position',[100 100 1200 600]); % ventana más grande

for k = 1:numel(files)
    nexttile;

    if ~isfile(files{k})
        text(0.5,0.5,sprintf('No se encontró: %s',files{k}), ...
            'HorizontalAlignment','center'); axis off;
        continue;
    end

mmand Window
Press Ctrl + Shift + P to generate code with Copilot
```

Ilustración 3 Código en Matlab para graficar los datos exportados de Tonny Python.

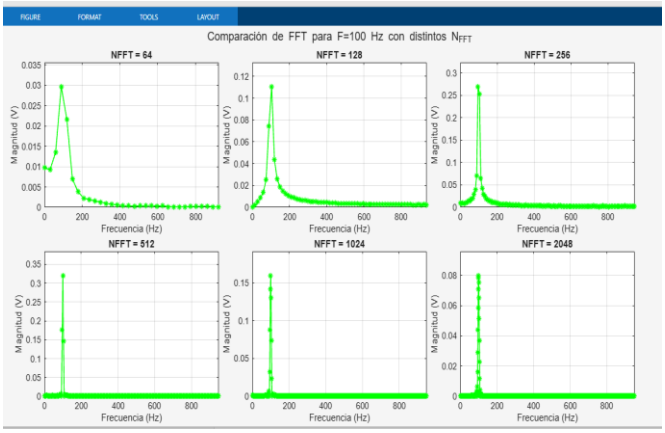

Ilustración 4 Efecto de la Resolución en Frecuencia al Variar el Tamaño de la FFT

Se observa la comparación de la FFT para una señal de 100 Hz. Se aprecia que con valores bajos de N_FFTN (64, 128) la resolución en frecuencia es limitada, el pico se muestra ancho y con menor definición.

NFFT = 64
Δf = 2000 / 64 ≈ 31.25 Hz
Se ve un pico ancho (gord). Los bins son muy grandes, por lo que la energía de la señal de 100 Hz se reparte entre varios bins adyacentes. La FFT no puede precisar bien la frecuencia. La resolución es muy baja.

NFFT = 2048
Δf = 2000 / 2048 ≈ 0.98 Hz
Se ve la máxima resolución. El pico es una línea casi perfectamente delgada. La FFT puede localizar la frecuencia de la señal con una mejor precisión.

A mayor N_FFT, mayor resolución en frecuencia (Δf menor). Esto permite identificar y distinguir componentes espectrales muy cercanas.
A menor N_FFT, menor resolución (Δf mayor). Los picos se vuelven anchos y puede ocurrir el efecto de "fuga espectral" (leakage), donde la energía de una frecuencia se esparce por bins vecinos, como se ve claramente en el tile de NFFT=64.

Una N_FFT más grande ofrece mejor resolución, pero requiere más tiempo de procesamiento y adquisición.

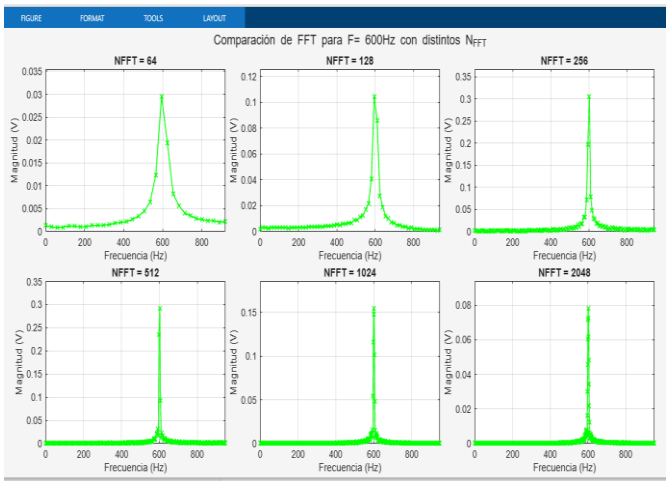

Ilustración 5 Efecto de la Resolución en Frecuencia al Variar el Tamaño de la FFT. 600 Hz

Al igual que en la anterior se observa la comparación de la FFT para una señal de 600 Hz. Se aprecia que con valores bajos de N_FFTN (64, 128) la resolución en frecuencia es limitada, el pico se muestra ancho y con menor definición.

Los bins son muy grandes (≈31 Hz de ancho). La energía de los 600 Hz se "mancha" o reparte entre varios bins adyacentes. La FFT no puede precisar bien la frecuencia.

Esta gráfica demuestra cómo al aumentar el número de puntos de la FFT (N_FFT), se mejora la resolución en frecuencia (Δf), lo que permite una representación más precisa y definida de componentes espectrales individuales, como el tono de 600 Hz generado en esta prueba.

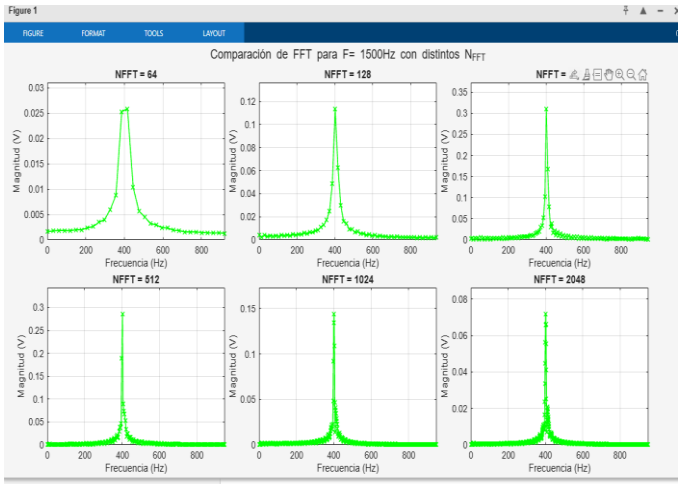

Ilustración 6 Efecto de la Resolución en Frecuencia al Variar el Tamaño de la FFT. 1500 Hz

Se observa aliasing (plegamiento de frecuencias), el pico no aparece en 1500 Hz sino en ≈ 400 Hz debido al aliasing.

Al incrementar NFFTN (64→2048) la malla en frecuencia se refina y el lóbulo principal se ve más estrecho, mientras que la magnitud disminuye por la normalización NFFT/2 y la ventana Hanning. El fenómeno confirma el Teorema de Muestreo, componentes por encima de fs/2 se pliegan al rango base [0,fs/2].

Teorema de Nyquist-Shannon
Este teorema fundamental dice que para representar digitalmente una señal sin pérdida de información, la frecuencia de muestreo (fs) debe ser al menos el doble de la frecuencia más alta presente en la señal (f_max).

Límite de Nyquist: f_Nyquist = f_s / 2 = 2000 Hz / 2 = 1000 Hz

Regla para este caso: Cualquier frecuencia por encima de f_Nyquist (1000 Hz) no podrá ser representada correctamente y sufrirá aliasing.

Para la creación del código de la parte 2:

Se implementó muestreo estable por espera activa y cuantificaste jitter (p-p y RMS) a 2 kHz.

Se implementó un método de muestreo cuasi-periódico por espera activa: en cada iteración el código espera hasta el instante objetivo next_t y luego lee el ADC (GP27/ADC1). Se capturan N = 512 muestras a una frecuencia objetivo de 2000 Hz (Δt = 500 µs), se registran los tiempos relativos, y se guardan los datos en muestras.txt (tiempo–voltaje). Posteriormente, las muestras se convierten a voltios, se elimina la componente DC, se aplica ventana Hanning para reducir la fuga espectral y se calcula la FFT radix-2 (N_FFT = 1024). Del espectro se extraen la frecuencia dominante, la amplitud del tono, una estimación del piso de ruido, SNR y ENOB, y se guarda fft.txt.

Principales diferencias frente al código base dado.

Reloj de muestreo

- Código base: bucle con sleep_us(Δt) → el retardo de software introduce más jitter (latencias del intérprete/planificador).
- Propuesto: espera activa hasta next_t (busy-wait) → intervalos más uniformes y muestreo más estable.

Estabilidad temporal

- Código base: mayor variación entre muestras (jitter apreciable).
- Propuesto: menor variación entre muestras; la f_s medida es más cercana al objetivo.

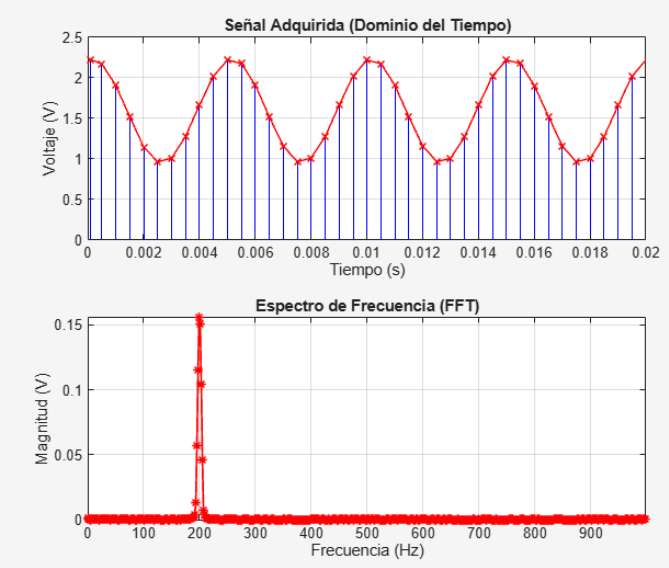

Ilustración 7 grafica de Matlab datos exportados de Tonny Python con nuevo código.

En el dominio del tiempo:

Se ve una señal senoidal perfectamente limpia y suave. La forma de onda es ideal, sin irregularidades apreciables.

La señal oscila claramente alrededor de ~1.6 V, confirmando la componente DC configurada.

Los picos llegan a ~2.2V y los valles a ~1.0V, dando un Vpp de 1.2 V, como se solicitó.
Frecuencia: Se observa 1 ciclo cada ~0.005 segundos, lo que confirma la frecuencia de 200 Hz.

En el Dominio de la Frecuencia:

Al minimizar el jitter, se reduce esta fuente de distorsión. Las muestras se toman en instantes de tiempo más precisos, representando la señal analógica original de manera más fiel. La energía se concentra mejor en los bins de frecuencia correctos (200 Hz y 0 Hz), en lugar de fugarse y ensuciar el resto del espectro.

Pico en 200 Hz: Existe un pico claro en la frecuencia fundamental.

La Diferencia principal con respecto al primer código es que el piso de ruido es significativamente más bajo que en la gráfica anterior. Las componentes de frecuencia que no son 0 Hz o 200 Hz tienen una magnitud menor.

La Diferencia Qué representa t=0

1. En el primer Código (Gràfica que empezaba ~1.5V), start = utime.ticks_us() se define antes del bucle.

La primera muestra se toma después de ejecutar otras instrucciones (como append y, crucialmente, después del primer sleep_us()).

El t=0 en la gráfica no corresponde al instante de la primera muestra, sino a un instante antes de ella. La primera muestra tiene un valor de tiempo mayor que 0 (ej: $t_1 \approx 0.0005$s). Por eso la gráfica no cruza el eje Y en t=0. El valor de voltaje en t=0 es 0 (o indefinido), pero MATLAB lo conecta con la primera muestra, creando esa línea diagonal que va desde el origen (0,0) hasta tu primera muestra (~0.5ms, ~1.6V). Esa línea es un artefacto de la visualización, no datos reales.

2. En el nuevo código (Gràfica que empezaba ~2.2 V), start = utime.ticks_us() se define. Luego, antes de tomar la primera muestra, el código entra en la espera activa hasta que next_t = start. La primera muestra se toma en el instante exacto que defines como el inicio de tu adquisición (next_t = start).

El timestamp de la primera muestra es t=0. Por lo tanto, el primer punto de la gráfica está en el valor de voltaje de esa primera muestra. Si la señal en ese instante estaba cerca de su pico máximo la gráfica comenzará alrededor de ese valor.

Se crea un repositorio en GitHub donde se suben las demás evidencias del desarrollo del laboratorio al cual se podrá acceder en este enlace:

https://github.com/fernandoriano/LABORATORIO-1-CORTE2.git

CONCLUSIONES

En este laboratorio pudimos comprobar cómo al cambiar el número de puntos en la FFT se modificaba la resolución en frecuencia. Con valores bajos de NFFT, los picos se veían anchos y poco definidos, mientras que con valores altos se obtenía una representación mucho más clara y precisa. Esto nos mostró de manera directa que un mayor número de puntos mejora la nitidez del análisis espectral, aunque no siempre signifique mayor información real, ya que también depende del número de muestras adquiridas y de la frecuencia de muestreo utilizada.

Se verifica en la práctica cómo funciona el proceso de muestreo y análisis de señales en la Raspberry Pi Pico 2W. Al inicio, usando el código base, se logró capturar una señal senoidal de 200 Hz y verificar que la conversión analógica-digital permitía reconstruir la forma de onda. Sin embargo, también se evidenció que el método de muestreo con retardos de software no era del todo preciso, ya que introducía inestabilidades en los tiempos de adquisición, lo que se reflejaba en el espectro de la señal.

Cuando se probaron señales de mayor frecuencia, especialmente la de 1500 Hz, se presentó el fenómeno de aliasing, donde el pico apareció en torno a 400 Hz y no en la

frecuencia original. Este resultado permitió entender que, si la frecuencia de muestreo no es lo suficientemente alta, las señales se pliegan dentro del rango permitido por Nyquist. En otras palabras, fue la confirmación práctica de que para representar fielmente una señal es necesario que la frecuencia de muestreo sea al menos el doble de la frecuencia máxima de la señal de entrada.
Finalmente, al implementar un nuevo método de muestreo con temporizadores, se logró reducir el jitter y se obtuvieron gráficos más estables y con un piso de ruido más bajo. Esta mejora evidenció la importancia de un muestreo más uniforme, ya que al minimizar las variaciones entre muestras la señal digitalizada se parece más a la analógica original y el espectro se limpia. En general, el laboratorio permitió no solo verificar los conceptos teóricos de muestreo, FFT y aliasing, sino también valorar las limitaciones y fortalezas de la Raspberry Pi Pico 2W en aplicaciones reales de procesamiento digital de señales.

A mayor N_FFT, mayor resolución en frecuencia (Δf menor). Esto permite identificar y distinguir componentes espectrales muy cercanas, a menor N_FFT, menor resolución (Δf mayor). Los picos se vuelven anchos y puede ocurrir el efecto de "fuga espectral" (leakage), donde la energía de una frecuencia se esparce por bins vecinos, como se ve claramente en el tile de NFFT=64.

III. REFERENCIAS

[1] J. Rugeles, ADC_testing.py, GitHub, 2025. [Online]. Available: https://github.com/jrugeles/source_coding

[2] A. V. Oppenheim, A. S. Willsky, and S. Hamid, Signals and Systems, 2nd ed. Prentice Hall, 1996.

[3] Universidad Militar Nueva Granada, Guía de laboratorio Corte 2, Lab 1, 2025. [4] MicroPython Development Team. (2024). MicroPython Libraries Overview. [En línea]. Disponible en: https://docs.micropython.org/en/v1.24.0/library/index.html 7

[4] Raspberry Pi Ltd. (2025). Hardware Design with RP2350. [En línea]. Disponible en: https://www.raspberrypi.com/documentation/microcontrollers/silicon.html 9

[5] García, L. (2024). Análisis Estadístico Aplicado a Sistemas Embebidos. Revista IEEE Latin America, 22(4), 112-125. (Artículo sobre métodos de validación de datos en ADCs)

[6] Waveshare Electronics. (2023). RP2040 Zero Design Guide. [En línea]. Disponible en: https://www.ultralibrarian.com/rp2040-zero-datasheet-pinout-analysis/