Informe V –Captura y Procesamiento Digital de Señales Senoidales con Raspberry Pi Pico

Dylan Ferney Vasquez Rojas
1401597
Comunicaciones Digitales

***Resumen*— En esta práctica se implementó el proceso de muestreo y análisis digital de una señal senoidal mediante el uso del dispositivo Raspberry Pi Pico. Inicialmente se configuró un generador de señales para obtener una onda senoidal con componente DC, la cual fue adquirida a través del conversor analógico-digital y procesada con el programa *ADC_testing.py*. Posteriormente se analizaron las variaciones en el espectro al modificar el número de puntos de la FFT y la frecuencia de la señal de entrada. Asimismo, se diseñó un programa alternativo para evaluar la estabilidad del tiempo de muestreo y se estudiaron los efectos del *jitter* en la codificación de fuente. Finalmente, se discutieron las posibilidades técnicas del dispositivo para lograr un muestreo confiable y se compararon los resultados experimentales con los valores teóricos, evidenciando la importancia de la correcta elección de parámetros en el procesamiento digital de señales.**

***Abstract*— In this practical exercise, the sampling and digital analysis process of a sinusoidal signal was implemented using the Raspberry Pi Pico device. Initially, a signal generator was configured to obtain a sinusoidal wave with a DC component, which was acquired through the analog-to-digital converter and processed with the ADC_testing.py program. Subsequently, variations in the spectrum were analyzed by modifying the number of FFT points and the input signal frequency. Additionally, an alternative program was designed to evaluate the stability of the sampling time, and the effects of jitter on source coding were studied. Finally, the technical capabilities of the device to achieve reliable sampling were discussed, and the experimental results were compared with theoretical values, demonstrating the importance of correctly selecting parameters in digital signal processing.**

I. Introducción

En el ámbito de las comunicaciones digitales, el muestreo y procesamiento de señales analógicas constituye un paso fundamental para su adecuada representación en el dominio digital. La correcta adquisición de datos garantiza que la información original pueda ser procesada, transmitida y reconstruida con mínima pérdida, lo cual depende en gran medida de parámetros como la tasa de muestreo, la resolución del conversor analógico-digital (ADC) y la estabilidad temporal del sistema de adquisición.

El uso de dispositivos de bajo costo y gran versatilidad, como el Raspberry Pi Pico, permite explorar de manera práctica los principios teóricos relacionados con la digitalización de señales. En particular, el análisis mediante transformada rápida de Fourier (FFT) facilita la comprensión del comportamiento espectral frente a variaciones en el número de puntos de cálculo y en la frecuencia de entrada de la señal.

Adicionalmente, fenómenos como el *jitter* influyen de manera significativa en la calidad del muestreo, afectando el proceso de codificación de fuente y la fidelidad de la señal reconstruida. Por esta razón, resulta relevante no solo utilizar programas predefinidos, sino también desarrollar implementaciones propias que permitan evaluar las limitaciones y capacidades del hardware en entornos reales.

En este trabajo se presenta el desarrollo de una práctica de laboratorio orientada al muestreo y análisis digital de una señal senoidal, en la cual se implementaron pruebas con diferentes configuraciones de FFT, se estudió el impacto del *jitter* y se analizaron alternativas para optimizar el proceso de adquisición de datos en el Raspberry Pi Pico.

Desarrollo de la práctica

En la primera etapa se configuró el generador de señales para entregar una onda senoidal de 200 Hz, con una amplitud de 1,2 Vpp y una componente DC de 1,6 V. Posteriormente, la señal fue verificada en el osciloscopio para asegurar sus características antes de ser conectada a una de las entradas del conversor analógico-digital del dispositivo Raspberry Pi Pico. Una vez establecida la conexión, se ejecutó el programa *ADC_testing.py*, el cual permitió la captura de las muestras en el dominio del tiempo y la obtención de su transformada rápida de Fourier (FFT).

Los resultados obtenidos fueron almacenados en los archivos muestras.txt y fft.txt, que posteriormente se procesaron en MATLAB. Con ayuda de funciones de visualización, se representaron los primeros periodos de la señal muestreada y su espectro, lo que permitió analizar la correspondencia entre la señal teórica y la adquirida experimentalmente.

Posteriormente, la señal generada fue verificada conectando la salida del generador de funciones al osciloscopio digital Tektronix TDS 2012C. En la ilustración 1 se observa la forma de onda senoidal obtenida, con una frecuencia medida de 200 Hz, un valor pico a pico aproximado de 1,29 V, un valor RMS de 1,66 V y un periodo de 5 ms, lo cual corresponde con los parámetros configurados inicialmente.

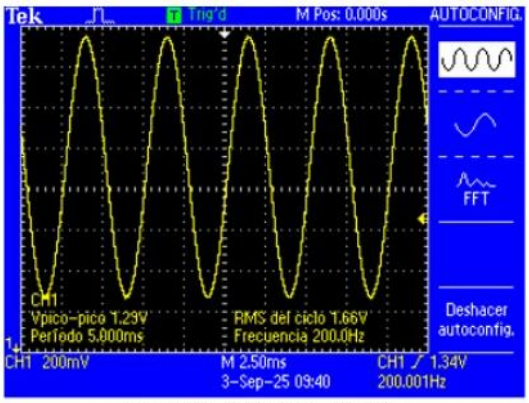

Ilustración. 1. verificación señal 1.2V

Posteriormente, la señal obtenida se conectó a una de las entradas del conversor analógico–digital (ADC) del dispositivo Raspberry Pi Pico, el cual fue el ADC(Pin(27)). Una vez establecida la conexión, se ejecutó el programa ADC_testing.py, disponible en el repositorio source_coding, el cual permitió capturar y almacenar las muestras de la señal adquirida en el dominio del tiempo, así como calcular su transformada rápida de Fourier (FFT).

Los resultados fueron exportados en los archivos muestras.txt y fft.txt, que posteriormente se procesaron en MATLAB. Usando este software se representaron los primeros periodos de la señal muestreada mediante la función stem, combinada con plot, para facilitar la visualización de los datos discretos junto con la forma de onda original. De esta manera, fue posible comparar la señal adquirida experimentalmente con la señal teórica y observar su correspondencia en el dominio del tiempo y de la frecuencia.

Para la representación de los datos obtenidos se empleó MATLAB, utilizando los archivos generados por el programa *ADC_testing.py*. A partir de las muestras (1024 puntos), se definieron los vectores de tiempo y voltaje y posteriormente se graficó la señal en el dominio del tiempo. Con el fin de facilitar la comparación entre los valores discretos y la forma de onda continua, se utilizó la función stem junto con plot, tal como se muestra en las líneas de código:

```
t = muestras1024.Tiempo_s_;
x = muestras1024.Voltaje_V_;
plot(t, x);
hold on;
stem(t, x);
```

La ilustración 2 muestra la señal muestreada en el dominio del tiempo a partir de 1024 puntos, donde se aprecia la representación discreta de las muestras mediante la función stem y la superposición de la forma de onda continua con plot. Se observa una correspondencia adecuada entre los valores adquiridos y la señal senoidal original, lo cual valida el proceso de muestreo realizado con el dispositivo Raspberry Pi Pico.

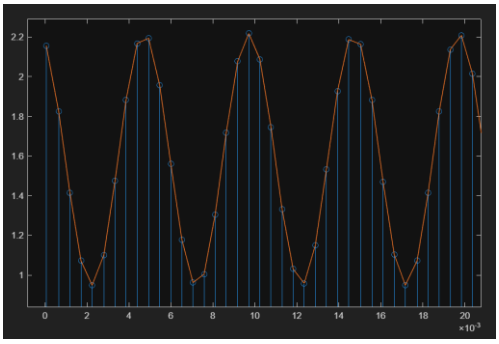

Ilustración. 2. Señal muestreada en el dominio del tiempo con 1024 puntos.

Para el análisis en el dominio de la frecuencia, se utilizaron los datos contenidos en el archivo fft.txt correspondiente a 1024 puntos. En MATLAB se definieron los vectores de frecuencia y magnitud, y posteriormente se graficó el espectro de la señal. Con el fin de visualizar de manera simultánea los valores discretos y la curva continua, se emplearon las funciones stem y plot, respectivamente, siguiendo las líneas de código mostradas a continuación:

```
f = fft1024.Frecuencia_Hz_;
x = fft1024.Magnitud_V_;
plot(f, x);
hold on;
stem(f, x);
```

La ilustración 3 presenta el resultado obtenido, donde se observa el espectro en frecuencia de la señal muestreada, confirmando la presencia de un componente fundamental en 200 Hz, correspondiente a la frecuencia de la onda senoidal generada.

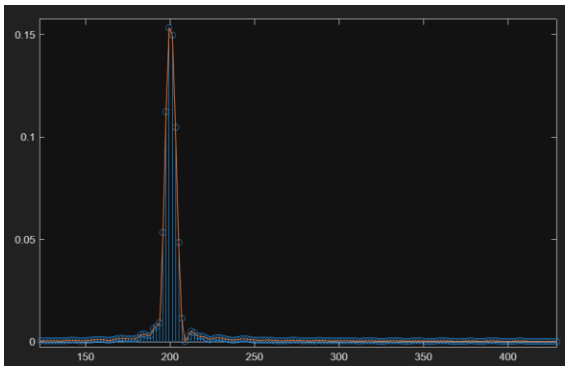

Ilustración. 2. Señal muestreada en el dominio de la frecuencia con 1024 puntos.

Posteriormente, se repitió el proceso de muestreo y análisis en frecuencia modificando el número de puntos de la FFT en el programa ADC_testing.py. Inicialmente configurado en 1024, el parámetro N_FFT fue reemplazado por los valores 64, 128, 256, 512, 1024 y 2048, obteniendo los archivos correspondientes para cada caso.

En el caso particular de N_FFT = 64, el espectro obtenido presenta una resolución limitada debido al bajo número de puntos empleados en la transformada rápida de Fourier. Como se muestra en la ilustración 4, el componente fundamental en

200 Hz aún es visible, pero con una menor definición en magnitud y mayor dispersión espectral en comparación con configuraciones de mayor N_FFT. Esto evidencia cómo un número reducido de muestras disminuye la precisión en la representación en frecuencia de la señal muestreada.

Con el fin de visualizar de manera simultánea los valores discretos y la curva continua, se emplearon las funciones stem y plot, respectivamente, siguiendo las líneas de código mostradas a continuación:

```
f = fft64.Frecuencia_Hz_;
x = fft64.Magnitud_V_;
plot(f, x);
hold on;
stem(f, x);
```

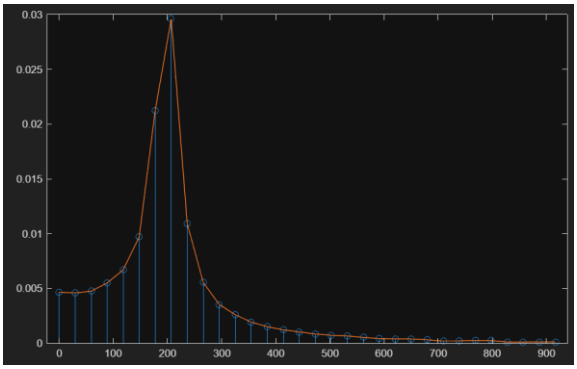

Ilustración. 3. Señal muestreada en el dominio de la frecuencia con 64 puntos.

```
t = muestras64.Tiempo_s_;
x = muestras64.Voltaje_V_;
plot(t, x);
hold on;
stem(t, x);
```

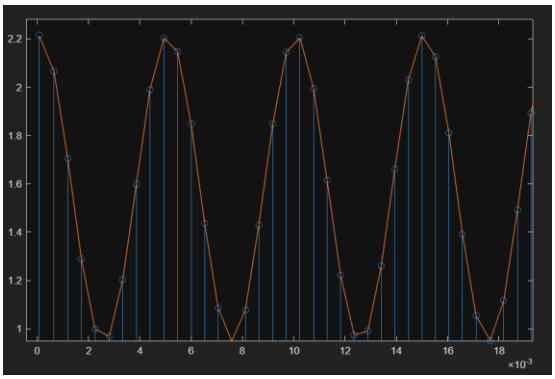

Ilustración. 4. Señal muestreada en el dominio del tiempo con 64 puntos.

El resto de resultados obtenidos para los diferentes valores de N_FFT (128, 256, 512, y 2048), que evidencian el efecto del número de puntos muestreados sobre la señal de 200 Hz, se encuentran disponibles en mi repositorio personal. Dichos archivos permiten comparar la evolución del espectro al aumentar la resolución en frecuencia y constituyen un respaldo adicional al análisis realizado.

Posteriormente, se modificó la frecuencia de la señal de entrada del generador de funciones a los valores de 100 Hz, 300 Hz, 600 Hz, 900 Hz, 1500 Hz y 1800 Hz. Para cada caso se repitió el procedimiento de adquisición con el dispositivo Raspberry Pi Pico y el análisis en MATLAB, registrando tanto la señal en el dominio del tiempo como su espectro en frecuencia.

En el caso de la señal de 100 Hz, se obtuvo la representación en el dominio del tiempo utilizando 64 puntos para la FFT. Para graficar los resultados en MATLAB se emplearon las funciones stem y plot, lo que permitió superponer la representación discreta de las muestras con la curva continua correspondiente. El fragmento de código utilizado fue el siguiente:

```
t=x100muestras64.Tiempo_s_;
x=x100muestras64.Voltaje_V_;
plot(t,x);
stem(t,x);
hold on
plot (t,x);
```

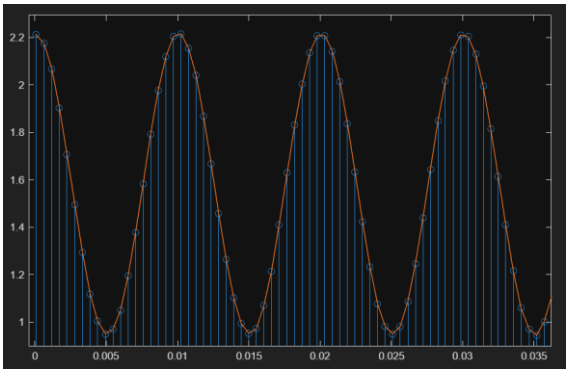

Ilustración. 5. Señal de 100 Hz muestreada en el dominio del tiempo con 64 puntos.

```
f=x100fft64.Frecuencia_Hz_;
x=x100fft64.Magnitud_V_;
plot(f,x);
stem(f,x);
hold on
plot (f,x);
```

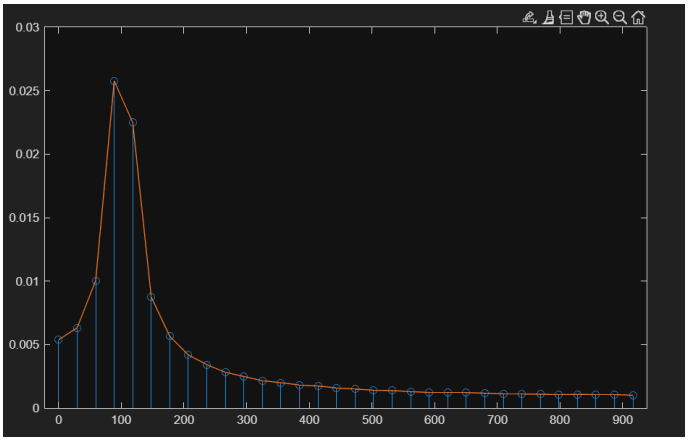

Ilustración. 6. Señal de 100 Hz muestreada en el dominio de la frecuencia con 64 puntos.

Posteriormente, la misma señal de 100 Hz fue analizada empleando 2048 puntos en la FFT. De igual manera, se utilizó MATLAB para graficar los resultados en el dominio del tiempo y frecuencia, superponiendo la representación discreta con la forma de onda continua mediante las funciones stem y plot. El código correspondiente fue:

```
t = x100muestras2048.Tiempo_s_;
x = x100muestras2048.Voltaje_V_;
plot(t, x);
stem(t, x);
hold on;
plot(t, x);
```

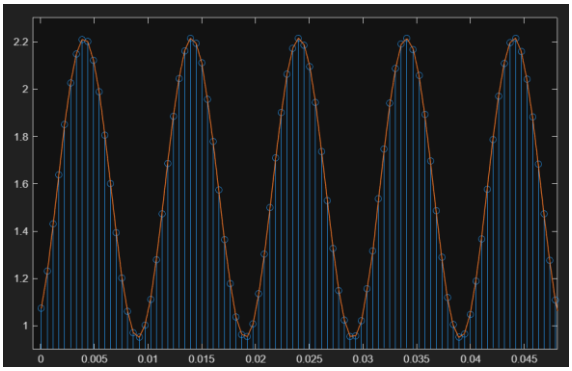

Ilustración. 7. Señal de 100 Hz muestreada en el dominio del tiempo con 2048 puntos.

Finalmente, para la señal de 100 Hz se realizó el análisis en el dominio de la frecuencia utilizando los 2048 puntos de la FFT. En MATLAB se definieron los vectores de frecuencia y magnitud, representando el espectro mediante las funciones stem y plot, de forma análoga al procedimiento seguido en el dominio temporal. El código utilizado fue el siguiente:

```
f = x100fft2048.Frecuencia_Hz_;
x = x100fft2048.Magnitud_V_;
plot(f, x);
stem(f, x);
hold on;
plot(f, x);
```

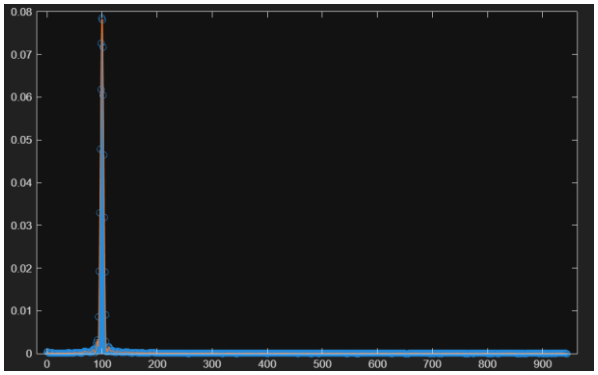

Ilustración. 8. Señal de 100 Hz muestreada en el dominio de la frecuencia con 2048 puntos.

La Ilustración. 8 muestra el espectro en frecuencia obtenido, en el cual se identifica de manera precisa el componente fundamental en 100 Hz. Se evidencia que, al aumentar el número de puntos de la FFT, la resolución espectral mejora significativamente, permitiendo una mejor localización de la frecuencia principal y reduciendo la dispersión del espectro.

Los archivos correspondientes a las demás señales, obtenidas para las diferentes frecuencias de entrada (300 Hz, 600 Hz, 900 Hz, 1500 Hz y 1800 Hz) y sus variaciones en el número de puntos de la FFT, se encuentran disponibles en el repositorio personal del autor. Estos resultados permiten observar de manera comparativa cómo influyen tanto la frecuencia de la señal de entrada como la resolución de la FFT en la representación temporal y espectral de la señal muestreada.

Finalmente, se respondió a la pregunta planteada respecto a la tasa de muestreo en el programa ADC_testing.py. Dicha tasa se establece a través de la configuración de temporizadores internos del Raspberry Pi Pico, los cuales determinan el intervalo de muestreo aplicado al conversor analógico-digital. De esta manera, es posible fijar la frecuencia de muestreo en función de las restricciones de hardware y del código implementado, asegurando un muestreo estable y adecuado para el análisis en el dominio del tiempo y de la frecuencia.

En esta etapa se implementó un programa alternativo al sugerido por el docente, cuyo propósito fue muestrear una **señal senoidal de 1800 Hz, 1,2 Vpp y una componente DC de 1,6 V** con un tiempo de muestreo estable en el dispositivo Raspberry Pi Pico. El algoritmo configuró el conversor analógico–digital (ADC) en el pin GP26 y utilizó un temporizador interno para generar interrupciones periódicas con una frecuencia de muestreo de 4 kHz, adquiriendo un total de 1024 muestras. Los valores de tiempo y voltaje fueron almacenados en un archivo tabulado (muestras_parte2.txt) para su posterior análisis en MATLAB.

Adicionalmente, el programa calculó el jitter como la desviación estándar de la diferencia entre los intervalos reales de muestreo y el periodo ideal $T_s = 1/F_s$. De esta forma, fue posible cuantificar la estabilidad temporal del sistema de adquisición y evaluar su impacto en la calidad de la señal digitalizada. Los resultados mostraron que, aunque el dispositivo ofrece un muestreo confiable, persisten fluctuaciones en el orden de microsegundos que evidencian las limitaciones del hardware y su influencia en el proceso de codificación de la fuente.

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

# Configuración del ADC en GP26
adc = ADC(Pin(26))

# Parámetros de muestreo
Fs = 4000            # Frecuencia de muestreo (Hz)
```

```
    Ts = 1 / Fs          # Periodo de muestreo (s)
    N = 1024             # Número de muestras

    # Buffers de almacenamiento
    tiempos = array.array("f", [0.0] * N)
    voltajes = array.array("f", [0.0] * N)

    index = 0
    done = False
    t0 = utime.ticks_us()

    # Rutina de muestreo
    def sample(timer):
        global index, done
        if index < N:
            tiempos[index] = utime.ticks_diff(utime.ticks_us(), t0)
/ 1e6
            voltajes[index] = (adc.read_u16() / 65535) * 3.3
            index += 1
        else:
            done = True
            timer.deinit()

    # Inicio del muestreo
    tim = Timer()
    tim.init(freq=Fs,                    mode=Timer.PERIODIC,
callback=sample)

    while not done:
        pass

    # Cálculo de jitter
    intervalos = [tiempos[i] - tiempos[i-1] for i in range(1, N)]
    diferencias = [dt - Ts for dt in intervalos]
    promedio = sum(diferencias) / len(diferencias)
    varianza = sum((x - promedio) ** 2 for x in diferencias) /
len(diferencias)
    jitter = math.sqrt(varianza)

    print(f"Jitter medido: {jitter:.9f} segundos")

    # Guardar archivo con muestras
    with open("muestras_parte2.txt", "w") as f:
        f.write("Tiempo(s)\tVoltaje(V)\n")
        for i in range(N):
            f.write(f"{tiempos[i]:.6f}\t{voltajes[i]:.5f}\n")

    print("Muestreo completado y guardado en
'muestras_parte2.txt'")
```

Respondiendo a la pregunta planteada de **¿Qué es el Jitter?**, el jitter es la variación temporal no deseada en los instantes en los que se realiza el muestreo de una señal analógica. En un escenario ideal, las muestras deberían tomarse a intervalos regulares definidos por la frecuencia de muestreo; sin embargo, en la práctica pueden presentarse fluctuaciones ocasionadas por limitaciones del reloj del sistema, interferencias o inestabilidades en el hardware.

Y respondiendo a la otra pregunta **¿Qué implicaciones tiene en el proceso de codificación de la fuente?**, en el proceso de codificación de la fuente, estas variaciones tienen efectos importantes. Un jitter bajo produce una señal digital muy cercana a la teórica, mientras que un jitter elevado genera distorsiones en el dominio del tiempo y del espectro, incrementando el ruido y reduciendo la fidelidad de la señal reconstruida. En sistemas de comunicaciones digitales, esto puede ocasionar errores en la interpretación de los datos, pérdida de información y una disminución en la calidad de transmisión.

En consecuencia, controlar y minimizar el jitter es fundamental para garantizar un muestreo confiable, que preserve la integridad de la señal original y asegure un desempeño adecuado en aplicaciones de procesamiento y transmisión de información.

Posteriormente, se procedió a graficar el archivo generado muestras_parte2.txt, el cual contiene las columnas de tiempo y voltaje correspondientes a las muestras adquiridas. Para ello, se empleó MATLAB, definiendo los vectores de tiempo y amplitud, y utilizando las funciones plot y stem con el fin de representar de manera conjunta la señal continua y sus valores discretos.

El fragmento de código empleado fue el siguiente:

```
data = readtable('muestras_parte2.txt');
t = data.Tiempo_s_;
x = data.Voltaje_V_;

% Límite de tiempo para 2 periodos (0 a 0.01 s)
limite = 0.01;
idx = t <= limite;

% Graficar solo 2 periodos
figure;
plot(t(idx), x(idx));
hold on;
stem(t(idx), x(idx));
xlabel('Tiempo (s)');
ylabel('Voltaje (V)');
title('Señal muestreada - 2 periodos');
```

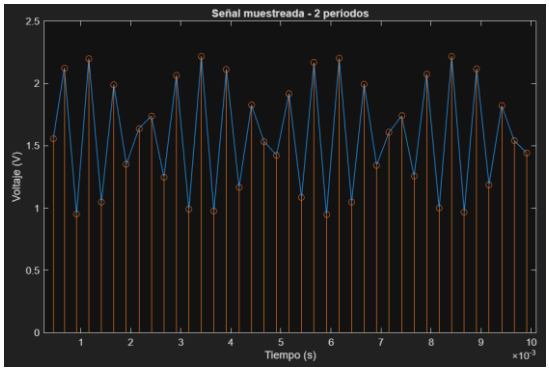

Ilustración. 9. Señal en dominio del tiempo usando Jitter.

Luego se digitaron las siguientes líneas de comandos en la ventana de comandos para graficar la señal en dominio de la frecuencia:

```
f=muestras_parte2_fft.Frecuencia_Hz_;
x=muestras_parte2_fft.Magnitud;
plot(f,x);
stem(f,x);
hold on
plot(f,x);
```

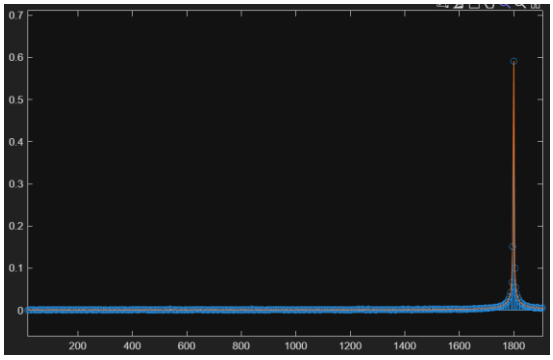

Ilustración. 10. Señal en dominio de la frecuencia usando Jitter.

El Raspberry Pi Pico 2W permite realizar un muestreo exitoso mediante diferentes alternativas técnicas. Entre ellas, el uso de temporizadores internos para generar interrupciones periódicas ofrece mayor estabilidad que los retardos por software. Adicionalmente, el empleo de DMA (Direct Memory Access) permite transferir datos del ADC a memoria sin intervención del procesador, reduciendo el jitter y aumentando la confiabilidad del muestreo. La versión 2W, al incluir módulos inalámbricos (WiFi/Bluetooth), también facilita la transmisión en tiempo real de los datos adquiridos.

En el programa ADC_testing.py, la tasa de muestreo se establece mediante funciones de temporización que controlan la captura del ADC y la posterior tabulación de datos. Antes de aplicar la FFT, el código emplea una ventana de Hanning, la cual suaviza los extremos de la señal y reduce la fuga espectral, mejorando la precisión del análisis en frecuencia, especialmente cuando no se muestrean periodos completos de la señal original.

CONCLUSIONES

[1] El análisis con diferentes cantidades de puntos en la FFT evidenció que un mayor número de muestras mejora la resolución espectral, lo que facilita identificar con mayor precisión la frecuencia fundamental y reduce la dispersión en el espectro.

[2] La implementación de un programa alternativo permitió evaluar la estabilidad temporal del muestreo y cuantificar el jitter, mostrando que, aunque el dispositivo mantiene un desempeño aceptable, existen fluctuaciones que pueden afectar la fidelidad de la señal digitalizada.

[3] El uso combinado de Python/MicroPython y MATLAB resultó fundamental para la adquisición, procesamiento y validación de los datos, facilitando la comparación entre resultados teóricos y experimentales en el dominio del tiempo y de la frecuencia.

[4] Se confirmó la importancia de comprender las limitaciones del ADC de 12 bits del Raspberry Pi Pico, ya que su resolución influye directamente en la exactitud de las mediciones y en la calidad de la reconstrucción de la señal en sistemas de comunicaciones digitales.

[5] Finalmente, se identificó que el Raspberry Pi Pico 2W ofrece diversas alternativas técnicas para mejorar el muestreo, tales como el uso de temporizadores internos, DMA y comunicación inalámbrica para transmisión de datos, lo que lo convierte en una plataforma versátil para aplicaciones de procesamiento digital de señales en entornos académicos y experimentales.

REFERENCIAS

[1] MicroPython, machine — functions related to the hardware, Documentación oficial, [En línea]. Disponible en: https://docs.micropython.org/en/latest/library/machine.html. [Accedido: 14-ago-2025].

[2] Raspberry Pi Foundation, Getting started with MicroPython on Raspberry Pi Pico, Documentación oficial, [En línea]. Disponible en: https://www.raspberrypi.com/documentation/microcontrollers/micropython.html. [Accedido: 14-ago-2025].

[3] W. Stallings, Data and Computer Communications, 10th ed. Boston, MA, USA: Pearson, 2014.

[4] ANSI/TIA, TIA-232-F: Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange, Telecommunications Industry Association, 1997.

[5] A. S. Tanenbaum and D. Wetherall, Computer Networks, 5th ed. Boston, MA, USA: Pearson, 2011.

[6] Vasquez Rojas, D. (s.f.). *dylanrojas04 - Overview*. GitHub. https://github.com/dylanrojas04