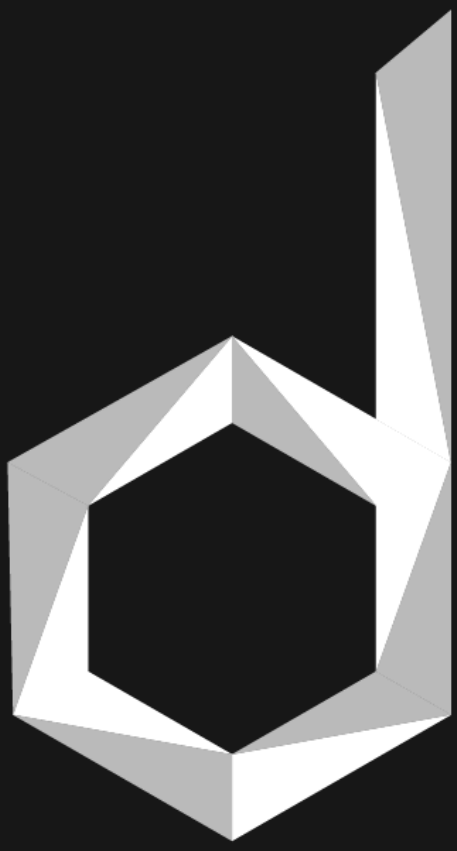
INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

INSTRUMENTACIÓN VIRTUAL

PYTHON 3.9.7, C# & LABVIEW

Instrumentación Virtual con Arduino - Recibir Datos

Contenido

Teoría – Instrumentación Virtual:

La instrumentación virtual es el uso de una computadora como instrumento de medición, en vez de utilizar herramientas físicas como osciloscopios, principalmente para bajar costos de operación en algún proceso. Esto se logra haciendo uso de instrumentos que no son tangibles para desarrollar aplicaciones con interfaz de usuario que realicen adquisición de datos a través de lenguajes de programación, protocolos de comunicación, etc. Los lenguajes de programación de código libre (gratuitos) que se pueden usar para ello son clasificados y descritos a continuación:

Lenguaje de programación	Generación	Características
Ensamblador (Lenguaje Máquina)	Primera Generación (1940 - 1950)	Lenguajes que la computadora comprende de manera directa, utiliza secuencias de números, operaciones lógicas binarias con bits e instrucciones de bajo nivel.
Fortran, Cobol, Basic.	Segunda Generación (1950 - 1965)	Surgen lenguajes de alto nivel, pero estaban destinados principalmente al cálculo numérico.
Algol, Pascal, C.	Tercera Generación (1966 - 1981)	Lenguajes de alto nivel con gramática, sintaxis, compiladores, etc.
C++, Visual Basic, JAVA, Python.	Cuarta Generación (1982 - 1994)	Son ya entornos de desarrollo con herramientas pre integradas y sintaxis más avanzada, muy utilizado en bases de datos.
Haskell	Quinta Generación (1995 - actualidad)	Utiliza lógica en vez de algoritmos como tal, da paso a la inteligencia artificial.

Visual Studio Code (Logo Azul) y Visual Studio (Logo Morado)

Además de poder utilizar programación de uso libre y usar algún editor de código gratuito, cuya función es simplemente asistir en el desarrollo de los programas, identificando y resaltando la correcta sintaxis en los lenguajes de programación utilizados y permitiendo que se completen automáticamente líneas de código en algunos casos como en el caso de Visual Studio Code (logo azul), se puede usar editores de paga como lo es Visual Studio (logo morado), no se debe confundir Visual Studio Code con Visual Studio, ya que son editores distintos, Visual Studio cuenta con herramientas que permiten crear dentro de la misma plataforma los siguientes proyectos:

- Interfaces gráficas (GUI).
- Conexión con bases de datos y/o conexión con servidores.

- Codificar entornos virtuales con modelos gráficos 3D o codificar videojuegos (Blender).
- Crear aplicaciones móviles en sistemas operativos Android o iOS, etc.

Plan Visual Studio	Costo mensual (USD)	Características
Individual	Gratuito	Editor de código gratuito.
Business	$45	Aprendizaje y soporte técnico, software de desarrollo y pruebas, etc.
Enterprise	$250	Plan básico y de pruebas.

LabVIEW (National Instruments)

Además de poder utilizar programación de uso libre (gratis pero más complejo) o editores de código como Visual Studio, existe la opción de usar plataformas que hacen uso de licencia (cobros anuales, pero con mayor facilidad de uso), a continuación, se muestra una tabla con los costos de la herramienta más famosa de instrumentación virtual en el mercado llamada LabVIEW, la cual pertenece a la empresa National Instruments:

Versión LabVIEW	Costo mensual (USD)	Características
LabVIEW Base	$10,980.00	Incluye procesamiento de señales, matemática básica, controladores para NI.
LabVIEW Completo	$86,515.00	Incluye matemática avanzada, controladores de NI y terceros, asistencia para FPGA en tiempo real
LabVIEW Profesional	$144,180.00	Incluye además de las versiones anteriores, complementos de ingeniería, capacidad de implementación de aplicaciones.

Cualquiera de las 3 opciones descritas anteriormente puede ser mejor que una instrumentación tradicional, ya que puede ser fácilmente escalable, modular, digitalizable y de mayor alcance.

Instrucciones – **Instrumentación Virtual Arduino**:

Escriba un programa que realice una operación de instrumentación virtual con Arduino, donde la instrumentación virtual se refiere al uso de software y hardware para simular o emular instrumentos físicos en un entorno virtual. En lugar de depender completamente de instrumentos físicos reales, la instrumentación virtual utiliza técnicas de programación y simulación para crear una representación virtual de los instrumentos y así poderlos analizar o transformar matemática y digitalmente.

En este caso lo que se busca hacer es leer los datos de tensión de un puerto analógico del Arduino (A0) para luego poderlos almacenar en una variable de Python y graficarlos actualizando sus datos en tiempo real.

Para ello primero se debe haber ejecutado un programa en el IDE de Arduino que indique cuál es el puerto de conexión serial que se debe utilizar, además de permitir la lectura de un puerto analógico del Arduino de donde se obtendrán los niveles de tensión en tiempo real a través del convertidor analógico digital (ADC) de 10 bits incluido en el Arduino.

Pseudocódigo.

1. Ejecutar un código que indique que el pin A0 es de lectura analógica y el pin 13 es de salida digital, para que de esa forma se recaben datos de tensión que vayan de 0 a 5V en el pin A0 y al mismo tiempo se haga parpadear un led en el pin 13, además de indicar cual es el puerto de conexión serial entre la placa de desarrollo y la computadora.
2. Iniciar la comunicación serial con el puerto de conexión previamente utilizado en el IDE del Arduino.
3. Indicar el número de datos a recabar del Arduino.
4. Leer los datos del pin analógico A0 del Arduino y realizar su conversión de número digital a valor de tensión, esto se realiza tomando en cuenta el rango de valores de tensión (que va de 0 a 5V) y el rango de valores digitales que se conforma de 10 bits, cuando este número binario se convierte a decimal se considera que va de valer 0 a valer $2^{10} - 1 = 1023$, la conversión entonces se realiza a través de la siguiente operación:

$$Tensión = Tensión_{Binaria} * \frac{Tensión_{Max}}{Resolución_{ADC}} = Tensión_{Binaria} * \frac{5\ [V]}{2^{10} - 1} = Tensión_{Binaria} * \frac{5\ [V]}{1023}$$

5. Almacenar los datos de tensión en una lista, tupla o diccionario.
6. Graficar el vector de datos recabados de tiempo vs. tensión.
7. Actualizar dinámicamente la gráfica para que se actualicen sus valores y se muestren en tiempo real.
    a. Esto se realiza utilizando la librería **drawnow** de Python.
8. Hacer que solo se muestren dos decimales de los valores de tensión recabados.
9. Imprimir en consola el vector que haya almacenado todos los valores de tensión recabados.

Código IDE Arduino:

Es importante mencionar que los archivos de Arduino a fuerza deben encontrarse dentro de una carpeta que tenga el mismo nombre que el nombre del archivo con extensión .ino que almacena el programa escrito en lenguaje Arduino, este nombre tanto de la carpeta no puede contener espacios.

```
//PROGRAMA PARA RECOPILAR DATOS Y MANDARLOS A PYTHON PARA QUE LOS GRAFIQUE EN TIEMPO REAL
int led = 13;           //Puerto de salida donde hay un led integrado en el Arduino
int voltage = A0;       //Declaración del puerto de entrada analógico que se quiere leer
int n = 0;              //variable que lleva la cuenta de los ciclos de parpadeo del LED.

//CONFIGURACIÓN DE LOS PINES Y COMUNICACIÓN SERIAL
void setup() {
  /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
```

```
  de la comunicación serial*/
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
      - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
      - segundo parámetro: Usa la insctrucción OUTPUT para indicar que el pin es una salida o
        INPUT para indicar que el pin es una entrada.
  El número del pin que recibe este método como primer parámetro se puede declarar directamente
  como un número o se puede declarar al inicio del programa como una variable*/
  pinMode(led, OUTPUT); //El pin 13 es una salida digital.
  /*Serial.begin(baudRate): Este método inicializa la comunicación serial entre la placa Arduino
  y la computadora, además de que configura su velocidad de transmisión dada en unidad de baudios
  (bit trasmitido por segundo) que recibe como su único parámetro:
      - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es
        compatible con la mayoría de los dispositivos y programas.
      - Sin embargo, si se necesita una transferencia de datos más rápida y el hardware/software
        lo admiten, se puede optar por velocidades más altas como 115200 o 57600 baudios.
  Es importante asegurarse de que la velocidad de transmisión especificada coincida con la
  velocidad de comunicación del otro dispositivo al que se conecta el Arduino. Si la velocidad de
  transmisión no coincide, los datos pueden no transmitirse o recibirse correctamente*/
  Serial.begin(9600);   //El pin 13 es una salida digital.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*El código principal se coloca dentro de la instrucción loop() para que se ejecute
  interminablemente en el microcontrolador ATMEGA328P de Arduino*/
  /*La operación % significa módulo y lo que hace esta es dividir un número y ver el resultado de
    su residuo, en el caso de la operación 3%2 == 0, lo que está haciendo es dividir 3/2 y ver si
    su residuo es cero, que en este caso no lo sería, ya que residuo = 1.
      - n%2 == 0: La operación verifica si el valor de n es divisible por 2 sin dejar residuo.
        En otras palabras, verifica si n es un número par*/
  if(n%2 == 0){         //Si n es par se prende el led
    /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
    específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
    constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada*/
    digitalWrite(led, HIGH);  //Con esta línea de código se prende el led
  }else{                //Si n es impar NO se prende el led
    digitalWrite(led, LOW);   //Con esta línea de código se apaga el led
  }
  n = n+1;              //Después de ver si n es par, se le suma 1 antes de su siguiente ejecución.
  //Posteriormente se reinicia la variable después de que haya llegado a n = 100.
  if(n == 100){
    n = 0;
  }

  /*analogRead(): El método se utiliza para leer valores analógicos de un pin específico, permitiendo
  leer la tensión analógica presente en un pin y convertirla en un valor digital.
      - Pines Analógicos A0, A1,..., A5: No es necesario configurarlos explícitamente, ya que el método
        se encarga de establecerlos como entradas analógicas automáticamente.
      - Pines Digitales 0, 1,..., 13: Estos pines antes de utilizarlos se deben establecer por medio del
        método pinMode() como entradas.
  El ADC del Arduino es de 10 bits, esto significa que cuando reciba su valor máximo de 5V, en la consola
  imprimirá (2^10)-1 = 1023, ya que es el valor máximo que puede convertir de analógico a digital porque
  recibe tensiones de 0 a 5V y por lo tanto cuenta con valores digitales de 0 a 1023 en formato decimal*/
  int data = analogRead(voltage);//Lectura analógica del puerto A0 para imprimirlo en la consola de Arduino

  /*Serial.println(): Método que imprime en las Herramientas Monitor Serie y Serial Plotter el valor dado
  en su parámetro.*/
  Serial.println(data);

  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos*/
  delay(1000);          //Esto retrasa 500 milisegundos el código antes de volver a ejecutarse.
}
```

Código Python – Visual Studio Code (Logo Azul):

```
# -*- coding: utf-8 -*-


#En Python se introducen comentarios de una sola linea con el simbolo #.

#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación

#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo

#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
```

```
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#INSTRUMENTACIÓN VIRTUAL CON ARDUINO:
#Para que se pueda realizar la instrumentación virtual con una tarjeta de desarrollo Arduino, primero se debe
#correr en en Arduino IDE el programa 18.-Instrumentacion_Virtual_con_Arduino.ino, en donde se declara el
#puerto de salida donde se hará parpadear un led y el puerto de entrada analógico con el cual se grafican los
#niveles de tensión que recibe
#   - Puerto de salida digital donde parpadea un led = 13.
#   - Puerto de entrada analógico que recibe niveles de tensión = A0.
#Si la tarjeta de desarrollo Arduino no se encuentra conectada al ordenador, el programa arrojará un error.
#El código completo de Arduino se incluye comentado en la parte de abajo de este programa.

import serial #serial: Librería que establece una comunicación serial con  microcontroladores, módems, etc.
import matplotlib.pyplot as plt #matplotlib: Librería de graficación matemática.
import drawnow #drawnow: Librería para generar gráficos que se actualizan continuamente en tiempo real.
import time #time: Librería del manejo de tiempos, como retardos, contadores, etc.

#VARIABLES:
t = []                #Lista que almacena los datos del eje horizontal (x = tiempo [s]) de la gráfica.
data = []             #Lista que almacena los datos del eje vertical (y = tensión [V]) de la gráfica.
temp_t = 0            #Variable que cuenta cada 0.5 segundos el tiempo de ejecución del programa.
samplingTime = 0.5    #Intervalo del conteo del objeto Time indicado en segundos.

serial_port = "COM7"  #Variable tipo string con el mismo puerto al que está conectado el Arduino en su IDE.

#Instancia de la librería serial por medio del constructor de la clase Serial para establecer una comunicación
#serial por medio de puertos seriales o USB con dispositivos externos como microcontroladores, módems, teclados,
#impresoras, etc. Los parámetros que puede recibir el constructor de la clase Serial son:
# - port: Especifica el nombre en formato string del puerto serial al que se desea conectar.
#           - Por ejemplo: "COM1" para sistemas operativos Windows o "/dev/ttyUSB1" para sistemas operativos
#             Unix/Linux o iOS.
# - baudrate: Define la velocidad de transmisión en baudios (bit trasmitido por segundo) para la comunicación
#   serial.
#           - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es compatible
#             con la mayoría de los dispositivos y programas.
#           - Sin embargo, si se necesita una transferencia de datos más rápida y el hardware/software lo
#             admiten, se puede optar por velocidades más altas como 115200 o 57600 baudios.
# - bytesize: Especifica el tamaño de los bytes en la comunicación serial. Puede adoptar uno de los siguientes
#   valores:
#           - serial.FIVEBITS: Tamaño de 5 bits en los paquetes de la transmisión serial.
#           - serial.SIXBITS: Tamaño de 6 bits en los paquetes de la transmisión serial.
```

```
#           - serial.SEVENBITS: Tamaño de 7 bits en los paquetes de la transmisión serial.
#           - serial.EIGHTBITS: Tamaño de 8 bits en los paquetes de la transmisión serial.
# - parity: Indica el tipo de paridad utilizado en la comunicación serial. La paridad es un mecanismo utilizado
#   en las comunicaciones seriales para verificar la integridad de los datos transmitidos, se basa en la adición
#   de un bit adicional (bit de paridad) en el bit más significativo (hasta la izquierda) de cada paquete de
#   datos transmitido. Al seleccionar la paridad, nos debemos asegurar de que tanto el dispositivo emisor como
#   el receptor estén configurados con la misma paridad para efecutar una comunicación adecuada:
#           - serial.PARITY_NONE: No se utiliza ningún bit de paridad. Esto implica que no se verifica la
#             integridad de los datos mediante la paridad.
#           - serial.PARITY_EVEN: Se utiliza la paridad par. Para ello se cuentan el número de bits en el byte,
#             incluido el bit de paridad:
#               - Si el número total de bits es impar, se establece el bit de paridad en 1 para que el número
#                 total de bits sea par.
#               - Si el número total de bits es par, se deja el bit de paridad en 0.
#                   - Por ejemplo, supongamos que se desea transmitir el byte 11010110. El bit de paridad en la
#                     transmisión de la comunicación se calcularía contando el número total de bits, que es 8,
#                     el número total de bits es par, por lo que el bit de paridad se establece en 0, Por lo
#                     tanto, el byte transmitido sería 011010110, donde el bit más significativo es el bit de
#                     paridad. Luego en el extremo receptor de la comunicación, se realizará un cálculo similar
#                     para verificar la integridad de los datos. Si el número total de bits, incluido el bit de
#                     paridad, no coincide con la paridad esperada (en este caso, par), se puede detectar un
#                     error en la transmisión de datos.
#           - serial.PARITY_ODD: Se utiliza paridad impar. El bit de paridad se establece de manera que el
#             número total de bits en el byte transmitido (incluido el bit de paridad) sea impar.
#           - serial.PARITY_MARK: Se utiliza paridad de marca. El bit de paridad se establece en 1 (marcado)
#             para todos los bytes transmitidos.
#           - serial.PARITY_SPACE: Se utiliza paridad de espacio. El bit de paridad se establece en 0 (espacio)
#             para todos los bytes transmitidos.
# - stopbits: Define el número de bits de parada en la comunicación serial. El número de bits de parada se
#   utiliza para indicar el final de cada byte transmitido en la comunicación serial. La elección del número de
#   bits de parada depende de la configuración del dispositivo externo con el que se está comunicando. El
#   parámetro uno de los siguientes valores:
#           - serial.STOPBITS_ONE: Indica que se utiliza un bit de parada.
#           - serial.STOPBITS_ONE_POINT_FIVE: Indica que se utiliza un bit y medio de parada. Este valor puede
#             ser utilizado en algunas configuraciones especiales.
#           - serial.STOPBITS_TWO: Indica que se utilizan dos bits de parada.
# - timeout: Especifica el tiempo de espera en segundos para las operaciones de lectura. Si no se recibe ningún
#   dato dentro de este tiempo, la operación de lectura se interrumpe.
# - xonxoff: Con True o False indica si se utiliza el control de flujo XON/XOFF para la comunicación serial.
# - rtscts: Con True o False indica si se utiliza el control de flujo RTS/CTS para la comunicación serial.
# - dsrdtr: Con True o False indica si se utiliza el control de flujo DSR/DTR para la comunicación serial.
# - write_timeout: Especifica el tiempo de espera en segundos para las operaciones de escritura. Si no se puede
#   escribir ningún dato dentro de este tiempo, la operación de escritura se interrumpe.
# - inter_byte_timeout: Define el tiempo de espera en segundos entre la recepción de bytes consecutivos durante
#   las operaciones de lectura.
```

```
arduino_board = serial.Serial(port = serial_port, baudrate = 9600)  #Inicio de comunicación serial.

#GRÁFICA ÚNICA TIPO MATPLOTLIB:
#make_figure(): Función para graficar los datos recabados del Arduino, se declara dentro de una función propia
#la graficación de los datos ya que para que estos puedan ser actualizados en tiempo real, se utiliza el método
#drawnow(), perteneciente a la librería drawnow, que debe recibir esta función como su parámetro.
def make_figure():
    plt.ylim(-0.1, 6) #matplotlib.ylim(): Método que indica el rango del eje vertical en la gráfica.
    #matplotlib.grid(): Método que recibe un valor booleano para indicar si aparece una rejilla o no en la
    #gráfica, por default está en valor False.
    plt.grid(True)           #La rejilla es visible en la gráfica.
    plt.xlabel("Tiempo (s)") #matplotlib.xlabel(): Método que indica el texto que aparece en el eje horizontal.
    plt.ylabel("Tensión (V))") #matplotlib.ylabel(): Método que indica el texto que aparece en el eje vertical.
    #matplotlib.plot(): Método usado para crear la gráfica, indicando como primer parámetro su eje horizontal,
    #luego su eje vertical y finalmente el estilo de la gráfica.
    # - Colores:             C1: color naranja, r: color rojo, b: color azul, g: verde, c: cyan, m: morado,
    #   y: amarillo, k: negro, w: blanco.
    # - Tipo de marcadores:  o: círculos, +: símbolos de más, .; puntos, v: Triángulo hacia abajo, h: Hexágono,
    #   s: cuadrados, etc.
    # - Tipo de Líneas:      -: sólida, --: punteada (líneas), :: punteada (puntos), -.: línea y punto,
    #   'or': Nada.
    plt.plot(t, data, 'rh--') #'rh--' significa r: color rojo, h: simbolo hexágonos, --: línea punteada
#matplotlib.show(): Método para mostrar la gráfica creada.
plt.show()

#EJECUCIÓN DE LA GRÁFICA:
N = 50        #Variable que indica el límite de datos recopilados, el límite de Excel es de 32,000 datos.
print("El programa recopilará ", N, " datos.")
for i in range(N):
    #MANEJO DE EXCEPCIONES: Es una parte de código que se conforma de dos partes, try y except:
    # - Primero se ejecuta el código que haya dentro del try y si es que llegara a ocurrir una excepción durante
    #   su ejecución, el programa brinca al código del except
    # - En la parte de código donde se encuentra la palabra reservada except, se ejecuta cierta acción cuando
    #   ocurra el error.
    #Se utiliza esta arquitectura de código cuando se quiera efectuar una acción donde se espera que pueda
    #ocurrir un error durante su ejecución.
    try:
        #VECTOR TENSIÓN OBTENIDO DEL PIN A0:
        #serial.Serial.readline(): Método para leer una línea completa de datos desde el puerto serial. Esta
        #función lee caracteres desde el puerto serial hasta que encuentra un carácter de nueva línea ('\n') o
        #una secuencia de retorno de carro-separador de línea ('\r\n'), que es es una combinación de caracteres
        #utilizada usualmente para indicar el final de una línea de texto en muchos sistemas informáticos y de
        #telecomunicaciones, donde el carácter de retorno de carro ('\r') mueve el cursor hacia el inicio de la
        #línea, como se hacía en las antiguas máquinas de escribir.
        data_arduino = arduino_board.readline()
```

```
        print("Dato del Arduino antes de ser decodificado: ", data_arduino)
        #serial.Serial.decode(): Método que convierte los datos recibidos desde un dispositivo externo conectado
        #a través de una comunicación serial a una cadena de caracteres legible, se utiliza para decodificar una
        #secuencia de bytes en una cadena de caracteres utilizando un determinado esquema de codificación:
        # - encoding (opcional): Especifica el esquema de codificación a utilizar para decodificar los bytes en
        #   una cadena de caracteres, si no se proporciona este parámetro, se utiliza la codificación
        #   predeterminada del sistema y los parámetros que puede recibir son los siguientes:
        #           - 'utf-8': Esquema de codificación UTF-8 que es capaz de representar caracteres especiales,
        #             letras acentuadas y otros caracteres no ASCII.
        #           - 'utf-16 o utf-32': UTF-8 es ampliamente utilizado y recomendado para aplicaciones web y
        #             transferencia de datos, mientras que UTF-16 y UTF-32 son comunes en sistemas que requieren
        #             soporte completo para todos los caracteres Unicode.
        #           - 'ascii': Especifica el esquema de codificación ASCII de 7 bits. Este esquema es compatible
        #             con los caracteres ASCII estándar y no puede representar caracteres fuera del rango ASCII.
        #           - 'latin-1' o 'iso-8859-1': Especifica el esquema de codificación Latin-1. Este esquema es
        #             capaz de representar los primeros 256 caracteres Unicode.
        #           - 'cp437': El esquema de codificación CP437 fue ampliamente utilizado en los sistemas
        #             informáticos antiguos, especialmente en los sistemas DOS y representa caracteres en un
        #             rango de 8 bits, representando hasta 256 caracteres diferentes.
        #str(): Método que convierte un tipo de dato cualquiera en string.
        data_str = str(data_arduino.decode('cp437'))
        #El string crudo obtenido del método readline() y decodificado con el método decode() viene por default
        #con un salto de línea de más, por eso es que se debe utilizar el método replace para removerlo.
        #replace(): Método que reemplaza un caracter que se encuentra en un string por otro declarado por
        #nosotros, esto se ejecutará todas las veces que dicho caracter aparezca en el string.
        data_str = data_str.replace("\n", "")
        print("Valor decimal obtenido del ADC de 10 bits del Arduino:       ", data_str)

        #float(): Método que convierte un tipo de dato cualquiera en númerico decimal.
        temp_data = float(data_str)

        #Se realiza esta operación porque como el ADC del arduino lee de 0 a 5V y como tiene una resolución de
        #10 bits permitiendo que en el ADC los valores de tensión se interpreten como valores numéricos enteros
        #que valen de 0 a (2^10)-1 = 1023, se hace una regla de 3 para que se imprima el valor de la tensión en
        #consola en vez del valor decimal binario.
        highValueBoard = 5              #Valor de tensión Máxima = 5V
        boardResolution = 1023          #Resolución de 10 bits del ADC perteneciente al Arduino: (2^10)-1 = 1023
        #Tensión = Tensión_decimal*(ValorMáximoTensión/ResoluciónADC) = Tensión_decimal*(5/1023)
        temp_data = temp_data*(highValueBoard/boardResolution)
        print(i, "-. Valor real de tensión:                              ", temp_data, "[V]")
        #append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
        data.append(temp_data)          #Creación del vector tensión (voltaje).

        #VECTOR TIEMPO:
        #Se usa una variable intermedia que va contando el tiempo transcurrido desde que se empezó a recopilar
```

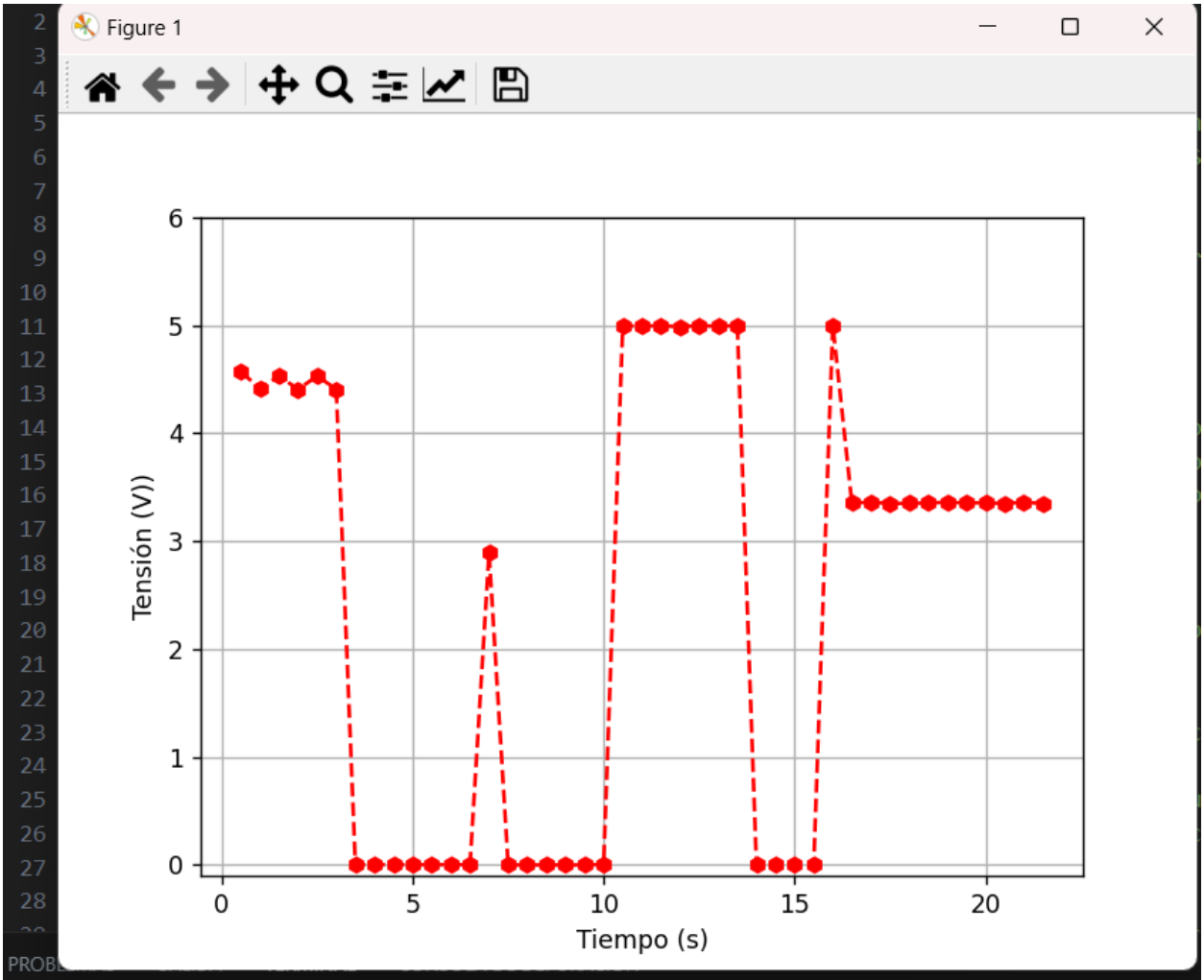
```
        #los valores de tensión del puerto analógico A0 del Arduino hasta que acaba. El intervalo de tiempo con
        #el que cuenta el temporizador y el tiempo que se detiene delay que se declarará después del except debe
        #ser el mismo.
        temp_t = temp_t + samplingTime  #Variable intermedia que cuenta el tiempo de ejecución del programa.
        #append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
        t.append(temp_t)                #Creación del vector tiempo.

        #drawnow.drawnow(): Método que permite actualizar y redibujar una figura o gráfico en tiempo real. Se
        #utiliza comúnmente junto con la librería matplotlib para crear visualizaciones interactivas que se
        #actualizan dinámicamente y se utiliza en conjunto con una función de graficación personalizada.
        #Esta función de trazado define cómo se representan los datos en el gráfico.
        drawnow.drawnow(make_figure)    #Actualización dinámica de la gráfica.
    except:
        #print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
        print("Chafeó tu medida bro!")

    #time.sleep(): Método que se utiliza para suspender la ejecución de un programa durante un intervalo de
    #tiempo específico dado en segundos.
    time.sleep(samplingTime)            #Delay de 0.5 segundos

#serial.Serial.decode(): Método que cierra la comunicación serial. Es muy importante mencionar que si no se
#ejecuta este método, el puerto serial se va a quedar bloqueado y no se podrá usar.
arduino_board.close()                   #Terminación de la comunicación serial.
#matplotlib.close(): Método que cierra la gráfica previamente abierta con el método matplotlib.show().
plt.close()
print("Se recopilaron correctamente ", N, " datos.\n")
#round(): Método que permite redondear un número a una cantidad específica de decimales, en su primer
#parámetro se indica número que se quiere recorrer y en el segundo el número de decimales que se quiere
#conservar, si este método se usa dentro de una lista vacía, utilizando un bucle for de una sola línea, se
#podrá redondear cada elemento de la lista al número de decimales deseado.
#Bucle for en una sola línea: [instrucción      for   variable_local   in   range(inicio, final)]
#Bucle for en una sola línea: [instrucción      for   variable_local   in   lista_tupla_diccionario]
data_round = [round(x, 2) for x in data]
print(data_round)



#CÓDIGO ARDUINO: El código Arduino se encuentra a continuación comentado en líneas múltiples utilizando la
#nomenclatura de triple comilla: """Comentario Múltiple""".
"""//PROGRAMA PARA RECOPILAR DATOS Y MANDARLOS A PYTHON PARA QUE LOS GRAFIQUE EN TIEMPO REAL
int led = 13;           //Puerto de salida donde hay un led integrado en el Arduino
int voltage = A0;       //Declaración del puerto de entrada analógico que se quiere leer
int n = 0;              //variable que lleva la cuenta de los ciclos de parpadeo del LED.

//CONFIGURACIÓN DE LOS PINES Y COMUNICACIÓN SERIAL
```

```
void setup() {
  /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
  de la comunicación serial*/
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
      - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
      - segundo parámetro: Usa la insctrucción OUTPUT para indicar que el pin es una salida o
        INPUT para indicar que el pin es una entrada.
  El número del pin que recibe este método como primer parámetro se puede declarar directamente
  como un número o se puede declarar al inicio del programa como una variable.*/
  pinMode(led, OUTPUT); //El pin 13 es una salida digital.
  /*Serial.begin(baudRate): Este método inicializa la comunicación serial entre la placa Arduino
  y la computadora, además de que configura su velocidad de transmisión dada en unidad de baudios
  (bit trasmitido por segundo) que recibe como su único parámetro:
      - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es
        compatible con la mayoría de los dispositivos y programas.
      - Sin embargo, si se necesita una transferencia de datos más rápida y el hardware/software
        lo admiten, se puede optar por velocidades más altas como 115200 o 57600 baudios.
  Es importante asegurarse de que la velocidad de transmisión especificada coincida con la
  velocidad de comunicación del otro dispositivo al que se conecta el Arduino. Si la velocidad de
  transmisión no coincide, los datos pueden no transmitirse o recibirse correctamente.*/
  Serial.begin(9600);   //El pin 13 es una salida digital.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*El código principal se coloca dentro de la instrucción loop() para que se ejecute
  interminablemente en el microcontrolador ATMEGA328P de Arduino.*/
  /*La operación % significa módulo y lo que hace esta es dividir un número y ver el resultado de
    su residuo, en el caso de la operación 3%2 == 0, lo que está haciendo es dividir 3/2 y ver si
    su residuo es cero, que en este caso no lo sería, ya que residuo = 1.
      - n%2 == 0: La operación verifica si el valor de n es divisible por 2 sin dejar residuo.
        En otras palabras, verifica si n es un número par.*/
  if(n%2 == 0){         //Si n es par se prende el led
    /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
    específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
    constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
    digitalWrite(led, HIGH);  //Con esta línea de código se prende el led
  }else{                //Si n es impar NO se prende el led
    digitalWrite(led, LOW);   //Con esta línea de código se apaga el led
  }
  n = n+1;              //Después de ver si n es par, se le suma 1 antes de su siguiente ejecución.

  //Posteriormente se reinicia la variable después de que haya llegado a n = 100.
  if(n == 100){
    n = 0;
```

```
  }

  /*analogRead(): El método se utiliza para leer valores analógicos de un pin específico, permitiendo
  leer la tensión analógica presente en un pin y convertirla en un valor digital.
      - Pines Analógicos A0, A1,..., A5: No es necesario configurarlos explícitamente, ya que el método
        se encarga de establecerlos como entradas analógicas automáticamente.
      - Pines Digitales 0, 1,..., 13: Estos pines antes de utilizarlos se deben establecer por medio del
        método pinMode() como entradas.
  El ADC del Arduino es de 10 bits, esto significa que cuando reciba su valor máximo de 5V, en la consola
  imprimirá (2^10)-1 = 1023, ya que es el valor máximo que puede convertir de analógico a digital porque
  recibe tensiones de 0 a 5V y por lo tanto cuenta con valores digitales de 0 a 1023 en formato decimal.*/
  int data = analogRead(voltage);//Lectura analógica del puerto A0 para imprimirlo en la consola de Arduino

  /*Serial.println(): Método que imprime en las Herramientas Monitor Serie y Serial Plotter el valor dado
  en su parámetro.*/
  Serial.println(data);

  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(1000);          //Esto retrasa 500 milisegundos el código antes de volver a ejecutarse.
}"""
```

Resultado del Código Python

```
PROBLEMAS   SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN

Valor decimal obtenido del ADC de 10 bits del Arduino:        688
49 -. Valor real de tensión:                                  3.3626588465298144 [V]
Se recopilaron correctamente  50  datos.

[4.57, 4.42, 4.53, 4.41, 4.53, 4.41, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.89, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.0, 4.99, 5.0, 4.99, 5.0, 4.99, 5.0, 0.0,
0.0, 0.0, 0.0, 5.0, 3.35, 3.36, 3.35, 3.35, 3.35, 3.36, 3.35, 3.36, 3.35, 3.36, 3.35, 3.36, 3.35, 3.35, 3.35, 3.35, 3.35, 3.36]
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>
```



Código C# (.Net Framework) – Visual Studio (Logo Morado):

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

//Referencias agregadas
using System.IO.Ports; //Puerto Serial
using System.Threading; //Temporización asíncrona
```

```
using System.IO; //Manejo de archivos


namespace Serial
{
    public partial class Frm_Arduino : Form
    {
        //Variables de instancia
        SerialPort serialPort; //Objeto para puerto serial
        int m = 0;//Numero de muestras
        bool stopAcquisition = false; //Bandera de estado de muestreo
        string[] dataStr = new string[0]; //Datos hacia archivo

        public Frm_Arduino()
        {
            InitializeComponent();
        }

        private void Frm_Arduino_Load(object sender, EventArgs e)
        {
            string[] ports = SerialPort.GetPortNames();//Leyendo los nombres de los puertos
            foreach(string port in ports)
            {
                CB_Port.Text = port;//Asiga los nombres de los puerto serial al combobox
            }
            Btn_Save.Visible = false;
            Btn_Start.Visible = true;
            Btn_Stop.Visible = false;
        }
        //Iniciando el puerto serial
        private bool InitSer()
        {
            bool result = false;
            try
            {
                serialPort = new SerialPort(CB_Port.Text);//Establece conexión con el arduino
                serialPort.BaudRate = 9600; //Tasa de transferencia
                serialPort.WriteTimeout = 100;//Retraso de 100ms
                result = true;
            }
            catch
            {
                MessageBox.Show("Fallo la conexion a la placa");
                result = false;
            }
            return result;
        }

        private async System.Threading.Tasks.Task GetDelay(int ms)
        {
            await System.Threading.Tasks.Task.Delay(ms);
        }//Delay

        private async void Btn_Start_Click(object sender, EventArgs e)//Click sobre el boton start
        {
            Btn_Start.Visible = false;//Ocultando botones
            Btn_Save.Visible = false;//
            bool statusPort = false;//Estado del puerto
            int period = 500;//Delay
            double time = 0;//
            dataStr = new string[0];//Inicializando la variable
            int samples = Convert.ToInt16(NUD_Samples.Value);//Obteniendo el numero de muestras
            m = 0;//Inicializando numero de muestras
            stopAcquisition = false;//Bandera de adquisión
            Plt_Data.Series.Clear();//Inicializando grafica
            Plt_Data.Series.Add("xy");
            Plt_Data.Series["xy"].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
            Plt_Data.Series["xy"].BorderWidth = 3;
            Plt_Data.Series["xy"].BorderDashStyle = System.Windows.Forms.DataVisualization.Charting.ChartDashStyle.Dash;
            Plt_Data.Series["xy"].MarkerStyle = System.Windows.Forms.DataVisualization.Charting.MarkerStyle.Circle;
            Plt_Data.Series["xy"].MarkerSize = 9;
            Plt_Data.Series["xy"].Color = Color.OrangeRed;
            statusPort = InitSer();
            await GetDelay(2000);
            if(statusPort==true)
            {
                serialPort.Open();
                serialPort.DiscardInBuffer();
                await GetDelay(1000);
```

```
            string data = "";
            double value = 0;
            int fail = 0;
            time = 0;
            for (int i=0;i<samples;i++)
            {
                if(i==0)
                {
                    Btn_Stop.Visible = true;
                }
                try
                {
                    data = serialPort.ReadLine();
                    value = Convert.ToDouble(data)*(5.0/1023);
                    Plt_Data.Series["xy"].Points.AddXY(time, value);
                    Array.Resize(ref dataStr, dataStr.Length + 1);
                    //Dato flotante 4 digitos
                    dataStr[dataStr.Length - 1] = Convert.ToString(time) + "," + value.ToString("F4");
                    m++;
                }
                catch
                {
                    fail++;
                }
                if (stopAcquisition==true)
                {
                    break;
                }
                time = time + (period / 1000.0);
                await GetDelay(period);
            }
            serialPort.Close();
        }
        await GetDelay(1000);
        Btn_Stop.Visible = false;
        Btn_Start.Visible = true;
        Btn_Save.Visible = true;
    }

    private async void Btn_Stop_Click(object sender, EventArgs e)
    {
        stopAcquisition = true;
        Btn_Stop.Visible = false;
        Btn_Start.Visible = false;
        Btn_Save.Visible = false;
        await GetDelay(1000);
        Btn_Start.Visible = true;
        Btn_Save.Visible = true;
    }

    private void Btn_Save_Click(object sender, EventArgs e)
    {
        Btn_Start.Visible = false;
        Btn_Save.Visible = false;
        Stream myStream;
        SaveFileDialog saveFileDialog = new SaveFileDialog();
        saveFileDialog.RestoreDirectory = true;
        if(saveFileDialog.ShowDialog()==DialogResult.OK)
        {
            if((myStream=saveFileDialog.OpenFile()) !=null)
            {
                using (StreamWriter writer = new StreamWriter(myStream))
                {
                    for(int i=0;i<dataStr.Length;i++)
                    {
                        writer.WriteLine(dataStr[i]);
                    }
                }
                myStream.Close();
            }
        }
        Btn_Start.Visible = true;
        Btn_Save.Visible = true;
    }
  }
}
```

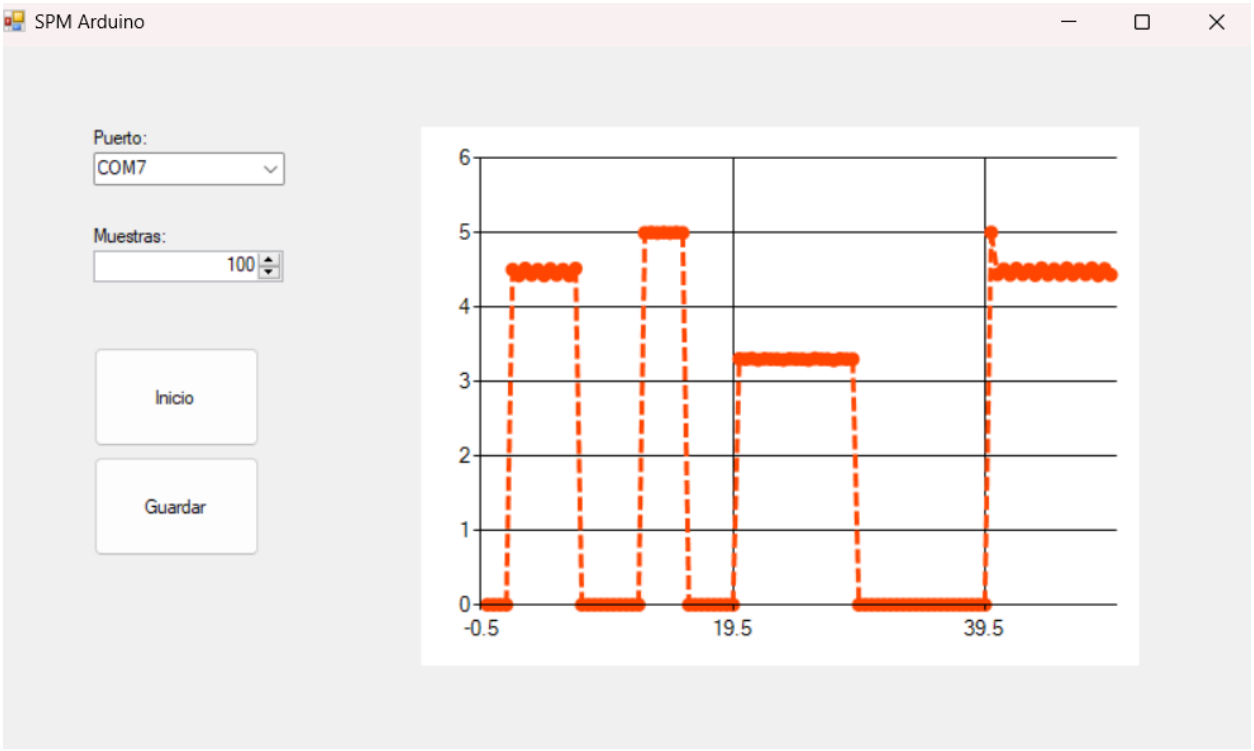
Código C# de la interfaz gráfica (GUI) [Design .cs]:

```
namespace _12._Instrumentación_Virtual_con_Arduino
{
    partial class Frm_Arduino
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.Windows.Forms.DataVisualization.Charting.ChartArea chartArea2 = new
System.Windows.Forms.DataVisualization.Charting.ChartArea();
            this.Lbl_Port = new System.Windows.Forms.Label();
            this.Lbl_Samples = new System.Windows.Forms.Label();
            this.NUD_Samples = new System.Windows.Forms.NumericUpDown();
            this.CB_Port = new System.Windows.Forms.ComboBox();
            this.Btn_Start = new System.Windows.Forms.Button();
            this.Btn_Stop = new System.Windows.Forms.Button();
            this.Btn_Save = new System.Windows.Forms.Button();
            this.Plt_Data = new System.Windows.Forms.DataVisualization.Charting.Chart();
            ((System.ComponentModel.ISupportInitialize)(this.NUD_Samples)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.Plt_Data)).BeginInit();
            this.SuspendLayout();
            //
            // Lbl_Port
            //
            this.Lbl_Port.AutoSize = true;
            this.Lbl_Port.Location = new System.Drawing.Point(63, 51);
            this.Lbl_Port.Name = "Lbl_Port";
            this.Lbl_Port.Size = new System.Drawing.Size(41, 13);
            this.Lbl_Port.TabIndex = 0;
            this.Lbl_Port.Text = "Puerto:";
            //
            // Lbl_Samples
            //
            this.Lbl_Samples.AutoSize = true;
            this.Lbl_Samples.Location = new System.Drawing.Point(63, 113);
            this.Lbl_Samples.Name = "Lbl_Samples";
            this.Lbl_Samples.Size = new System.Drawing.Size(53, 13);
            this.Lbl_Samples.TabIndex = 1;
            this.Lbl_Samples.Text = "Muestras:";
            //
            // NUD_Samples
            //
            this.NUD_Samples.Location = new System.Drawing.Point(66, 129);
            this.NUD_Samples.Minimum = new decimal(new int[] {
            1,
            0,
            0,
            0});
            this.NUD_Samples.Name = "NUD_Samples";
            this.NUD_Samples.Size = new System.Drawing.Size(120, 20);
            this.NUD_Samples.TabIndex = 2;
            this.NUD_Samples.TextAlign = System.Windows.Forms.HorizontalAlignment.Right;
            this.NUD_Samples.Value = new decimal(new int[] {
            1,
```

```
            0,
            0,
            0});
            //
            // CB_Port
            //
            this.CB_Port.FormattingEnabled = true;
            this.CB_Port.Location = new System.Drawing.Point(66, 67);
            this.CB_Port.Name = "CB_Port";
            this.CB_Port.Size = new System.Drawing.Size(121, 21);
            this.CB_Port.TabIndex = 3;
            //
            // Btn_Start
            //
            this.Btn_Start.Location = new System.Drawing.Point(66, 190);
            this.Btn_Start.Name = "Btn_Start";
            this.Btn_Start.Size = new System.Drawing.Size(105, 63);
            this.Btn_Start.TabIndex = 4;
            this.Btn_Start.Text = "Inicio";
            this.Btn_Start.UseVisualStyleBackColor = true;
            this.Btn_Start.Click += new System.EventHandler(this.Btn_Start_Click);
            //
            // Btn_Stop
            //
            this.Btn_Stop.Location = new System.Drawing.Point(66, 190);
            this.Btn_Stop.Name = "Btn_Stop";
            this.Btn_Stop.Size = new System.Drawing.Size(105, 63);
            this.Btn_Stop.TabIndex = 5;
            this.Btn_Stop.Text = "Paro";
            this.Btn_Stop.UseVisualStyleBackColor = true;
            this.Btn_Stop.Click += new System.EventHandler(this.Btn_Stop_Click);
            //
            // Btn_Save
            //
            this.Btn_Save.Location = new System.Drawing.Point(66, 259);
            this.Btn_Save.Name = "Btn_Save";
            this.Btn_Save.Size = new System.Drawing.Size(105, 63);
            this.Btn_Save.TabIndex = 6;
            this.Btn_Save.Text = "Guardar";
            this.Btn_Save.UseVisualStyleBackColor = true;
            this.Btn_Save.Click += new System.EventHandler(this.Btn_Save_Click);
            //
            // Plt_Data
            //
            chartArea2.Name = "ChartArea1";
            this.Plt_Data.ChartAreas.Add(chartArea2);
            this.Plt_Data.Location = new System.Drawing.Point(273, 51);
            this.Plt_Data.Name = "Plt_Data";
            this.Plt_Data.Size = new System.Drawing.Size(454, 340);
            this.Plt_Data.TabIndex = 7;
            this.Plt_Data.Text = "chart1";
            //
            // Frm_Arduino
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Controls.Add(this.Plt_Data);
            this.Controls.Add(this.Btn_Save);
            this.Controls.Add(this.Btn_Stop);
            this.Controls.Add(this.Btn_Start);
            this.Controls.Add(this.CB_Port);
            this.Controls.Add(this.NUD_Samples);
            this.Controls.Add(this.Lbl_Samples);
            this.Controls.Add(this.Lbl_Port);
            this.Name = "Frm_Arduino";
            this.Text = "SPM Arduino";
            this.Load += new System.EventHandler(this.Frm_Arduino_Load);
            ((System.ComponentModel.ISupportInitialize)(this.NUD_Samples)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.Plt_Data)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label Lbl_Port;
        private System.Windows.Forms.Label Lbl_Samples;
```

```
        private System.Windows.Forms.NumericUpDown NUD_Samples;
        private System.Windows.Forms.ComboBox CB_Port;
        private System.Windows.Forms.Button Btn_Start;
        private System.Windows.Forms.Button Btn_Stop;
        private System.Windows.Forms.Button Btn_Save;
        private System.Windows.Forms.DataVisualization.Charting.Chart Plt_Data;
    }
}
```

Resultado de la GUI creada con Código C# en Visual Studio



Referencias:

National Instruments, “Referencia y catálogo de instrumentación de National Instruments”, 1997 [Manual]

National Instruments, “EDICIÓN DE LABVIEW”, 2023 [Online], Available: https://www.ni.com/es-mx/shop/labview/select-edition.html

Python, “History and License”, 2023 [Online], Available: https://docs.python.org/3/license.html