# INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉSTUXTLA

**INVESTIGACIÓN DOCUMENTAL UII**

**“APLICACIONES REALES**

**DE LA INSTRUMENTACIÓN VIRTUAL”**

**CARRERA**

**INGENIERÍA MECATRÓNICA**

**PRESENTA**

**FRANCISCO EDUARDO AZAMAR**

**GRUPO:**

**911-A**

**CATEDRÁTICO**

**DR. JOSÉ ÁNGEL NIEVES VÁSQUEZ**

**ASIGNATURA:**

**INSTRUMENTACIÓN VIRTUAL**

**SAN ANDRES TUXTLA VER, A 06 DE OCTUBRE DEL 2025**

## Introducción

En la actualidad, la instrumentación virtual representa una de las herramientas más importantes dentro del campo de la Ingeniería Mecatrónica, ya que permite integrar la medición, el control y la automatización de procesos mediante el uso del software y el hardware programable. A diferencia de los instrumentos tradicionales, los instrumentos virtuales ofrecen la capacidad de diseñar, simular y ejecutar funciones de adquisición y análisis de datos a través de entornos de programación flexibles, tales como LabVIEW, MATLAB o Python, lo que reduce costos, optimiza recursos y mejora la precisión en la toma de decisiones técnicas.

El bloque “Instrumentos Virtuales” tiene como objetivo comprender los principios de programación y manejo de datos aplicados a la instrumentación. A lo largo de los subtemas se estudian los ambientes de programación y las estructuras fundamentales, tales como funciones, subrutinas, ciclos y temporización, que son indispensables para el desarrollo lógico de los programas.
Además, se profundiza en el uso de arreglos, grupos de datos, cadenas y archivos de entrada/salida, los cuales permiten organizar, almacenar y comunicar la información generada por los sistemas de medición. Finalmente, se integran estos conceptos en ejemplos prácticos con software especializado, demostrando su aplicabilidad en entornos reales de monitoreo y control.

En conjunto, estos conocimientos brindan al estudiante una base sólida para diseñar soluciones de instrumentación modernas, eficientes y escalables, capaces de adaptarse a los requisitos actuales de la industria 4.0 y de los sistemas inteligentes de fabricación.

## Instrumentos Virtuales.

La rápida adopción de la PC en los últimos 20 años catalizó una revolución en la instrumentación para pruebas, medición y automatización. Un desarrollo importante derivado de la ubicuidad de la PC es el concepto de instrumentación virtual, que ofrece diversas ventajas a ingenieros y científicos que requieren mayor productividad, precisión y rendimiento.

Un instrumento virtual consiste en una computadora o estación de trabajo estándar de la industria equipada con un potente software de aplicación, hardware rentable, como placas de conexión, y software de controlador, que en conjunto realizan las funciones de los instrumentos tradicionales. Los instrumentos virtuales representan una transición fundamental de los sistemas de instrumentación tradicionales centrados en hardware a sistemas centrados en software que aprovechan la potencia de procesamiento, la productividad, la visualización y las capacidades de conectividad de las computadoras de escritorio y estaciones de trabajo más populares. Si bien la PC y la tecnología de circuitos integrados han experimentado avances significativos en las últimas dos décadas, es el software el que realmente proporciona la base para construir sobre esta potente base de hardware y crear instrumentos virtuales, lo que ofrece mejores maneras de innovar y reducir significativamente los costos. Con instrumentos virtuales, los ingenieros y científicos construyen sistemas de medición y automatización que se adaptan exactamente a sus necesidades (definidos por el usuario) en lugar de estar limitados por los instrumentos tradicionales de función fija (definidos por el proveedor).

## 2.1Ambientes de programación

Los entornos de desarrollo pueden variar según el lenguaje de programación y el tipo de aplicación que se desea construir. Sin embargo, todos comparten algunas características clave:

- Editor de código: Es una de las partes más importantes. Permite escribir y editar el código fuente con funciones avanzadas como resaltado de sintaxis, autocompletado y corrección de errores.
- Compilador o intérprete: Se encarga de traducir el código fuente a un lenguaje que la máquina pueda entender y ejecutar.
- Depurador: Facilita la identificación y corrección de errores en el código.

- Herramientas de prueba: Algunos entornos incluyen opciones para realizar pruebas automáticas y asegurar que el software funcione correctamente.
- Control de versiones: Permite llevar un seguimiento de los cambios en el código, facilitando el trabajo en equipo.

### Tipos de entornos de desarrollo de programación

- Existen diferentes tipos de entornos de desarrollo según su funcionalidad y el tipo de programación que soportan. A continuación, se mencionan algunos de los más comunes:
- Entornos de desarrollo integrados (IDE): Son plataformas completas que incluyen todas las herramientas necesarias para programar, como Visual Studio Code, Eclipse o JetBrains IntelliJ IDEA.
- Editores de texto avanzados: Programas como Sublime Text o Atom que, aunque no son IDE completos, ofrecen funciones avanzadas para la escritura de código.
- Entornos en la nube: Permiten programar sin necesidad de instalar software en la computadora, como Replit o GitHub Codespaces.
- Entornos especializados: Diseñados para lenguajes o aplicaciones específicas, como Android Studio para desarrollo de aplicaciones móviles en Android.

## Ventajas de usar un entorno de desarrollo adecuado

Trabajar con un entorno de desarrollo de programación adecuado trae numerosos beneficios, entre ellos:

- **Mayor productividad**: Al contar con herramientas que automatizan tareas repetitivas y facilitan la depuración, se reduce el tiempo de desarrollo.
- **Mejor organización del código:** Facilita la estructuración de proyectos y la colaboración entre desarrolladores.
- **Reducción de errores:** Gracias a las funciones de corrección y depuración, se minimizan los errores en el código.
- **Compatibilidad y escalabilidad:** Algunos entornos permiten el desarrollo en múltiples lenguajes y la integración con otras herramientas.

## ¿Cómo elegir el mejor entorno de desarrollo de programación?

Para seleccionar el mejor entorno de desarrollo, es importante considerar:

**El lenguaje de programación**: No todos los entornos son compatibles con todos los lenguajes. Es clave elegir uno que se adapte al lenguaje que se va a utilizar.

**La facilidad de uso:** Algunos entornos son más intuitivos que otros. Para principiantes, puede ser recomendable optar por un IDE con una interfaz sencilla.

**La compatibilidad con herramientas externas:** Si se requiere integrar bases de datos, librerías o frameworks, es importante que el entorno lo permita.

**El rendimiento y la estabilidad**: Un buen entorno debe ser rápido y estable para evitar interrupciones en el desarrollo.

Los ambientes de programación en instrumentación virtual son plataformas que permiten diseñar, simular y controlar sistemas de medición o automatización mediante software. En lugar de depender únicamente de hardware físico, se utilizan entornos gráficos o textuales para crear “instrumentos virtuales” que emula el comportamiento de equipos reales como multímetros, osciloscopios o controladores.

Algunos ejemplos son:

- **LabVIEW (National Instruments):** entorno gráfico ampliamente utilizado en ingeniería para crear interfaces de adquisición de datos y control de procesos.
- **MATLAB/Simulink:** permite la simulación matemática y la integración con hardware.
- **Python (con librerías como PyVISA o NumPy):** ideal para instrumentación con código abierto.

## 2.2 Funciones y subrutinas.

Las funciones y subrutinas son bloques de código reutilizables que facilitan la organización de un programa.
En instrumentación virtual, se usan para dividir tareas complejas en secciones más pequeñas, como la adquisición de datos, el procesamiento de señales o la visualización de resultados.

- **Funciones:** Devuelven un valor y se utilizan cuando se requiere un cálculo específico (por ejemplo, calcular el promedio de una señal).
- **Subrutinas (o subVIs en LabVIEW):** ejecutan una tarea pero no necesariamente devuelven un valor; Sirven para modularizar el código y hacerlo más legible.

El uso de funciones y subrutinas mejora la eficiencia, reduce errores y permite reutilizar código en diferentes proyectos.

### Subrutinas

Una subrutina es un procedimiento que ejecuta cierta acción y obtiene un resultado. Las subrutinas pueden o no recibir parametros para su ejecución y no devuelven un resultado.

La sintáxis para escribir una subrutina es la siguiente:

```
Public Sub nombreSubrutina (p1 As Tipo_variable, p2 As Tipo_variable, ...)
  código que ejecuta la subrutina
End
```

Observe que la sintaxis de la subrutina requiere de un nombre único con el que se identificará dentro del programa. Opcionalmente se le puede enviar algún parámetro el cual la subrutina lo recibe con las variables que se declaran dentro de los paréntesis que están inmediatamente despues del nombre. El código que forma parte de la subrutina se debe escribir hasta antes de la palabra clave End.

El código de la subrutina debe escribirse al final del programa, justo después de que termina la rutina principal.

**Ejemplo**. Esta Subrutina calcula el promedio de dos números que recibe como argumento.

```
Public Sub Main()
  media(4,8)
End

Public Sub media(valor1 As Integer, valor2 As Integer)
  Print (valor1 + valor2) / 2
End
```

El programa funciona de la siguiente forma: La rutina principal incia y cuando la ejecución alcanza la líena donde se ecuentra media(4,8), Gambas busca una función o subrutina que se haya declarado al final de la rutina principal y brinca la ejecución hasta ese punto del programa. Al brincar, se envían los números 4 y 8 como argumentos de la subrutina.

Ya en la subrutina, se reciben los dos números 4 y 8 como argumentos y se almacenan en las variables valor1 y valor2 respectivamente. Observe que en ese momento se convierte ese argumento al tipo de dato que se delcara en la subrutina, en este caso Integer. Por esta razón si el argumento no fuera entero, se convertiría a Integer al guardarse en la variable.

Dentro de la subrutina se realiza el cálculo y la impresión del resultado. Una vez que se alcanza el End de la subrutina, se brinca de vuelta al punto donde ésta fue llamada y como ya no hay más intrucciones se completa así la ejecución del programa.

Las variables valor1 y valor2 solamente existen dentro de la subrutina. Esto sigifica que cuando se manda llamar la subrutina, se declaran tales variables y ellas solo existen en el *ámbito* de la subrutina. Una vez que termina la ejecución de la subrutina estas variables se destruyen. A este tipo de variables se les llama **variables locales**.

Es posible que una subrutina llame a otras subrutinas a su vez, incluso puede llamarse a sí misma.

**Funciones**

Una función es un procedimiento que ejecuta cierta acción y obtiene un resultado. Las funciones pueden o no recibir parametros para su ejecución y pueden o no devolver algún resultado.

La sintáxis para escribir una función es la siguiente:

```
Public Function nombreFuncion (p1 As Tipo_variable, p2 As Tipo_variable, ...) As tipoDato
  código que ejecuta la función
  Return
End
```

La sintaxis de la función requiere de un nombre único con el que se identificará dentro del programa. Opcionalmente se le puede enviar algún parámetro el cual la función lo recibe con las variables que se declaran dentro de los paréntesis que están inmediatamente despues del

nombre. El código que forma parte de la función se debe escribir hasta antes de la palabra clave End.

Tal como la subrutina, el código de la función deberá escribirse después de la rutina principal.

**Ejemplo**. Esta Función calcula el promedio de dos números que recibe como argumento.

```
Public Sub Main()
  dim x as Float
  x = media(4,8)
  Print "El promedio es: "; x
End

Public Function media(valor1 As Integer, valor2 As Integer) As Float
  dim p as Float
  p = (valor1 + valor2) / 2
  Return p
End
```

El programa funciona de la siguiente forma: La rutina principal incia, se declara la variable x y cuando la ejecución alcanza la líena donde se ecuentra x = media(4,8), Gambas busca una función o subrutina que se haya declarado al final de la rutina principal y brinca la ejecución hasta ese punto del programa. Al brincar, se envían los números 4 y 8 como argumentos de la subrutina.

Una vez que la ejecución llegó a la función, se reciben los dos números 4 y 8 como argumentos y se almacenan en las variables locales valor1 y valor2 respectivamente. Observe que en ese momento se convierte ese argumento al tipo de dato que se delcara en la función, en este caso Integer. Por esta razón si el argumento no fuera entero, se convertiría a Integer al guardarse en la variable.

En la función se declara la variable local p como Float. Luego se realiza el cálculo y se almacena en la variable p. Posteriomente se invoca a la instrucción Return, la cual se encarga de devolver el contenido de la variable que se le pasa como argumento, en este caso p. Return devuelve el contenido de la variable x al punto donde fue invocada la función, en este caso ese valor se almacena en la variable local x de la subrutina principal.

Las variables valor1 y valor2 solamente existen dentro de la función. Esto sigifica que cuando se manda llamar la función, se declaran tales variables y ellas solo existen en el *ámbito* de la función. Una vez que termina la ejecución de la función estas variables se destruyen. A este tipo de variables se les llama **variables locales**.

## 2.3 Ciclos y Temporización

En el ámbito de la instrumentación virtual , los ciclos y la temporización son elementos esenciales para el control del flujo de ejecución de un programa, especialmente cuando se trabaja con procesos de adquisición de datos, monitoreo continuo o control en tiempo real. Estos conceptos permiten que los instrumentos virtuales respondan de forma sincronizada a las señales del mundo físico, manteniendo una secuencia lógica, precisa y eficiente de ejecución.

## 1. Concepto de Ciclos en Instrumentación Virtual

Los **ciclos** o **estructuras de repetición** son bloques de programación que permiten ejecutar una o varias instrucciones Múltiples veces, ya sea mientras se cumple una condición o durante un número específico de iteraciones.
Su propósito principal es **automatizar tareas repetitivas** , como leer continuamente un sensor, actualizar una gráfica o realizar cálculos periódicos.

En el contexto de programas como **LabVIEW** , los ciclos son elementos gráficos dentro del diagrama de bloques y pueden contener tanto procesos lógicos como de adquisición o procesamiento de señales.

*Tipos principales de ciclos:*

1. **Ciclo For (Bucle For):**
    - Ejecuta un bloque de código un número determinado de veces.
    - Es útil cuando se conoce previamente el número de iteraciones, por ejemplo, al realizar una lectura de 100 muestras de un sensor.
    - En LabVIEW, el ciclo *For* incluye un contador que indica la cantidad de repeticiones.

    **Ejemplo práctico:**
    Generar un promedio de temperatura a partir de 50 lecturas consecutivas de un sensor digital.

2. **Ciclo While (bucle While):**
    - Repita el código mientras una condición sea verdadera (TRUE).
    - Es ideal para aplicaciones de monitoreo continuo o sistemas en tiempo real que deben ejecutarse indefinidamente hasta que el usuario decida detenerlos.

**Ejemplo práctico:**
Leer continuamente el valor de un sensor de presión y mostrarlo en un indicador gráfico hasta que el usuario presione el botón de “Stop”.

3. **Ciclo Hacer-Mientras o Repetir-Hasta:**
    - Se ejecuta al menos una vez y luego verifica la condición.
    - Se usa cuando sea necesario garantizar que el proceso suceda al menos una vez antes de la evaluación de la condición.

**Ejemplo práctico:**
Activar un actuador una vez y verificar su estado antes de continuar con el monitoreo.

## 2. Importancia de la Temporización

La **temporización** se refiere al control del tiempo entre iteraciones o ejecuciones dentro de un ciclo.
En instrumentación virtual, el tiempo de muestreo o de espera entre cada lectura es un parámetro crítico que afecta directamente la **precisión, estabilidad y velocidad del sistema** .

Sin una temporización adecuada, el programa podría:

- Saturar el procesador al ejecutar ciclos sin pausa.
- Perder información si la frecuencia de muestreo es demasiado baja.
- Descronizarse del proceso físico si no hay control temporal.

*Elementos comunes de temporización:*

1. **Retardos temporales (Delay):**
    - Se emplea para pausar la ejecución un número determinado de milisegundos.
    - En LabVIEW, se usa el bloque **“Wait (ms)”** para definir el intervalo de espera entre iteraciones.
2. **Temporizadores de hardware o software:**
    - En sistemas avanzados, los temporizadores permiten sincronizar la adquisición de datos con eventos externos, como señales de reloj o interrupciones.
3. **Frecuencia de muestreo:**
    - Defina cuántas veces por segundo se adquiere una señal.
    - De acuerdo con el teorema de Nyquist, la frecuencia de muestreo debe ser al menos el doble de la frecuencia máxima de la señal para evitar aliasing.
    - Ejemplo: si se mide una señal de 100 Hz, la frecuencia mínima de muestra debe ser de 200 Hz.

## Aplicaciones Prácticas en Instrumentación Virtual

Los ciclos y la temporización tienen aplicaciones directas en sistemas de control, adquisición y monitoreo industrial, entre las cuales destacan:

- **Monitoreo continuo de sensores:**
  Uso de un ciclo *Mientras* con un retardo de 100 ms para leer en tiempo real variables como temperatura, presión o nivel de líquido.
- **Control PID digital:**
  Implementación de un ciclo que calcula de forma periódica las acciones proporcionales, integrales y derivativas con base en un tiempo de muestreo definido.
- **Registro de datos (Data Logging):**
  Un ciclo ejecuta la lectura de un sensor y guarda los valores en un archivo cada intervalo de tiempo determinado, asegurando una base de datos ordenada y con marcas de tiempo precisas.
- **Simulación de señales:**
  Un ciclo *For* puede generar una onda senoidal o cuadrada en intervalos regulares, simulando la respuesta de un sensor o actuador.

## Consideraciones de diseño

- La duración del ciclo debe ser menor o igual al período de muestreo del sistema físico para evitar retrasos.
- Es recomendable utilizar estructuras de temporización no bloqueantes , de modo que el sistema pueda seguir respondiendo a eventos del usuario.
- En aplicaciones críticas, se deben usar temporizadores de hardware sincronizados con señales externas (disparadores) para garantizar la precisión.

## 2.4 Arreglos y grupos de datos.

En el desarrollo de aplicaciones de instrumentación virtual, la gestión eficiente de la información es fundamental. Las señales provenientes de sensores, actuadores o sistemas de control generan grandes volúmenes de datos que deben organizarse y procesarse adecuadamente.
Para lograrlo, los arreglos y grupos de datos (clusters) son estructuras de almacenamiento que permiten manipular información de forma ordenada, optimizando tanto la velocidad de procesamiento como la claridad del programa.

Estas estructuras son la base para el análisis de señales, la adquisición de datos, la visualización gráfica y la interconexión entre módulos de software y hardware.

### Arreglos (Arrays)

Un arreglo es una estructura de datos que almacena múltiples elementos del mismo tipo (por ejemplo, una lista de valores numéricos, cadenas de texto o booleanos). Los arreglos permiten acceder, modificar, calcular y graficar grandes conjuntos de datos de manera más eficiente que si se manejaa cada valor por separado.

En instrumentación virtual, los arreglos son esenciales para almacenar secuencias de mediciones tomadas a intervalos regulares, como temperatura, voltaje, corriente, presión o desplazamiento.

**Tipos de arreglos:**

Unidimensionales: contienen una sola fila o columna de datos (por ejemplo, 100 lecturas de voltaje).

- ✓ Bidimensionales: organizan los datos en filas y columnas, semejantes a una tabla (por ejemplo, una matriz de datos experimentales).
- ✓ Multidimensionales: se usan para representar datos en más de dos dimensiones, como mapas tridimensionales de temperatura o presión.
- ✓ Propiedades operaciones y comunes en arreglos:
- ✓ Índices: permiten acceder a una posición específica del arreglo (por ejemplo, el valor número 50 de 100 muestras).
- ✓ Subconjuntos: se pueden extraer segmentos o subconjuntos de datos para análisis específico.
- ✓ Funciones matemáticas: se pueden aplicar operaciones como suma, promedio, desviación estándar o filtrado a todos los elementos del arreglo.
- ✓ Gráficas: los arreglos se utilizan para alimentar gráficas de tiempo real o representaciones de señales adquiridas.

**Ejemplo práctico en LabVIEW:**

Supongamos que un sistema de adquisición de datos toma 100 lecturas de temperatura cada segundo.
En el diagrama de bloques, se puede utilizar un For Loop para adquirir los datos y construir un arreglo de 100 elementos.
Luego, este arreglo se envía a una gráfica de onda (Waveform Graph ) para visualizar la evolución temporal de la señal.

**Ejemplo en Python (con NumPy):**

```
pitón

import numpy as np
temperaturas = np.array([23.5, 23.8, 24.0, 24.3, 24.1])
promedio = np.mean(temperaturas)
print("Temperatura promedio:", promedio)
```

Este fragmento crea un arreglo con lecturas de temperatura y calcula su promedio, tal como se haría en una aplicación de monitoreo.

## Grupos de Datos (Clusters)

Un cluster (grupo de datos) es una estructura que permite agrupar diferentes tipos de datos en un solo contenedor.

A diferencia de los arreglos, donde todos los elementos deben ser del mismo tipo, los clusters pueden contener valores numéricos, booleanos, cadenas, arrays u otros clusters .

Esto los convierte en una herramienta ideal para empaquetar información compleja proveniente de un mismo proceso o medición.

Por ejemplo, una lectura de sensor puede incluir el valor medido (numérico), la unidad (cadena de texto), la hora de registro (timestamp) y el estado del sensor (booleano).

**Características de los clusters:**

- Permitirán mantener la coherencia entre distintos tipos de datos asociados.
- Mejoran la legibilidad del programa al agrupar información relacionada.
- Pueden anidarse (clusters dentro de clusters) para organizar jerárquicamente los datos.
- Facilitan la comunicación de datos entre subrutinas o subVIs.

**Ejemplo práctico en LabVIEW:**

Un cluster puede contener:

Un valor de temperatura (numérico).

El estado del sistema (booleano), indicando si el sensor está activo.

Una cadena de texto con la etiqueta “Sensor 1”.

Una marca de tiempo.

Este cluster puede conectarse a un solo terminal de entrada/salida, simplificando el diagrama de bloques y haciendo el código más modular.

**Ejemplo práctico en Python (usando diccionarios o tuplas):**

pitón

```
sensor = {
    "temperatura": 24.6,
    "activo": True,
    "etiqueta": "Sensor 1",
    "hora": "14:23:10"
}
print(sensor["temperatura"])
```

**Aplicaciones de Arreglos y Clusters en Instrumentación Virtual**

- Adquisición de datos: Los valores obtenidos por tarjetas DAQ se almacenan en arreglos para su análisis estadístico y graficación.
- Procesamiento de señales: Los arreglos permiten aplicar filtros digitales, transformadas de Fourier o cálculos RMS sobre los datos adquiridos.
- Visualización de información: Los clusters se usan para mostrar múltiples variables (como temperatura, presión y tiempo) en una sola interfaz.
- Transmisión y almacenamiento de datos: Los grupos de datos se pueden enviar entre subprogramas o guardar en archivos JSON, TXT o CSV para análisis posterior.
- Control de sistemas mecatrónicos: Un cluster puede contener tanto las señales de entrada de sensores como las salidas de control hacia actuadores, permitiendo un manejo estructurado del sistema completo

## 2.5 Cadenas y Archivos de Entrada / Salida

### Cadenas

### Definición

Una cadena es una secuencia ordenada de caracteres alfanuméricos que puede incluir letras, números, espacios y símbolos. En instrumentación virtual, las cadenas son utilizadas para representar información textual dentro de los programas, como nombres de sensores, mensajes en interfaces gráficas, comandos de comunicación o rutas de archivos.

En entornos como LabVIEW , las cadenas pueden visualizarse mediante indicadores y controles específicos, mientras que en lenguajes como Python , se manejan mediante comillas ( " ") o ( ' ').

Usos comunes de las cadenas

Mensajes e interfaces de usuario: mostrar texto informativo o estados del sistema (“Sistema conectado”, “Lectura completada”, etc.).

Comandos de comunicación: enviar y recibir instrucciones a través de puertos de comunicación (por ejemplo, RS-232 o USB).

Rutas de archivos: defina la ubicación de los archivos que se van a leer o escribir.

Conversión de datos: transforma valores numéricos a texto o viceversa para su visualización o almacenamiento.

Etiquetas y unidades: mostrar junto a los datos medidos (por ejemplo, “Voltaje = 12.5 V”).

### Funciones comunes aplicadas a cadenas

Concatenación: unir varias cadenas en una sola (por ejemplo, "Sensor: " + "Temperatura"→ "Sensor: Temperatura").

Búsqueda y reemplazo: localizar una palabra o símbolo dentro de un texto y sustituirlo.

Subcadenas: extraer partes específicas de una cadena (por ejemplo, solo la fecha dentro de una marca de tiempo).

Conversión numérica: transformar cadenas con números (“23.4”) en valores numéricos reales para cálculos.

**Ejemplo en LabVIEW:**

En un VI de monitoreo de sensores, una cadena puede combinarse con un valor numérico para mostrar en pantalla: "Temperatura actual: " + valor_numérico + " °C"

**Ejemplo en Python:**

pitón

```
temperatura = 26.8
mensaje = "Temperatura actual: " + str(temperatura) + " °C"
print(mensaje)
```

Salida:

```
Temperatura actual: 26.8 °C
```

**Archivos de Entrada y Salida (File I/O)**

La entrada/salida de archivos (Input/Output o I/O) consiste en los procesos de lectura y escritura de información hacia o desde archivos almacenados en un medio físico, como un disco duro o memoria USB.
Esta operación permite que un instrumento virtual almacene registros, recuperar configuraciones previas o exportar resultados para análisis posteriores.

En instrumentación virtual, los archivos son esenciales para lograr persistencia de datos, trazabilidad de mediciones y comunicación entre sistemas.

**Tipos de archivos más comunes**

- Archivos de texto (.txt, .csv):

Fáciles de leer y manipular.

Ideales para guardar datos numéricos o registros de sensores.

Los archivos .csv(valores separados por comas) son ampliamente usados para compatibilidad con Excel o MATLAB.

- Archivos binarios (.dat, .bin):

Almacenan datos en formato binario, ocupando menos espacio.

Permiten una lectura y escritura más rápida, útil en grandes volúmenes de datos o señales de alta frecuencia.

- Archivos de configuración (.ini, .json):

Contiene parámetros de calibración o ajustes del sistema que el programa puede leer al iniciar.

- Archivos de registro o log (.log):

Guardan eventos del sistema, errores o advertencias para diagnóstico y mantenimiento.

**Operaciones básicas de archivos**

Apertura del archivo: especifique si se leerá, escribirá o modificará.

Lectura o escritura: realizar la operación correspondiente.

Cierre del archivo: liberar los recursos y asegurar que los datos se guarden correctamente.

**Ejemplo en LabVIEW:**

Write to Text File.vi: permite escribir cadenas o datos numéricos en un archivo de texto.

Leer desde Spreadsheet File.vi: lee datos de archivos .csvo .txty los convierte en arreglos numéricos.

File Dialog.vi: abre un cuadro para seleccionar archivos desde el explorador del sistema.

**Ejemplo en Python:**

```
pitón

# Escritura de datos
with open("lecturas.txt", "w") as archivo:
    archivo.write("Temperatura: 26.8 °C\nPresión: 101.3 kPa")

# Lectura de datos
with open("lecturas.txt", "r") as archivo:
    contenido = archivo.read()
    print(contenido)
```

**Aplicaciones en Instrumentación Virtual**

Registro de datos experimentales: Guarde las mediciones obtenidas por sensores (por ejemplo, temperatura, voltaje o caudal) con marcas de tiempo.

Análisis post-proceso: Los datos almacenados pueden importarse a software como Excel, MATLAB o Python para graficar o realizar análisis estadístico.

Configuración dinámica de sistemas: Los archivos de texto o JSON pueden contener parámetros (como límites de alarma o constantes de calibración) que el programa carga al iniciar, adaptando su comportamiento automáticamente.

Comunicación entre instrumentos virtuales: Varios VIs o scripts pueden compartir información escribiendo y leyendo un mismo archivo de datos.

Generación de informes automáticos: Mediante cadenas, se construyen informes con los resultados de mediciones, que luego se guardan o imprimen en formato de texto o PDF.

**Ventajas del uso de cadenas y archivos I/O**

Trazabilidad: los datos quedan registrados para validación o auditorías.

Flexibilidad: permiten configurar programas sin necesidad de modificar el código.

Interoperabilidad: los archivos generados pueden leerse en otros programas o sistemas.

Automatización: posibilitan la creación de informes automáticos o el almacenamiento de periódicos de información sin intervención humana.

## Conclusiones

El estudio de los instrumentos virtuales permite comprender la forma en que la programación y la digitalización han transformado la instrumentación y el control de procesos industriales. A través del análisis de los distintos temas —desde los ambientes de programación hasta la gestión avanzada de datos— se evidencia que la instrumentación virtual no solo facilita la adquisición y procesamiento de señales, sino que también optimiza la interacción entre el operador y el sistema mediante interfaces personalizadas, automatización y almacenamiento eficiente de información.

Los conceptos de funciones, subrutinas, ciclos, temporización, arreglos, clusters, cadenas y archivos I/O conforman los fundamentos que permiten estructurar aplicaciones robustas y precisas. Estos elementos, combinados con un software adecuado, hacen posible desarrollar herramientas virtuales que sustituyen o complementan a los instrumentos físicos tradicionales, mejorando la flexibilidad, escalabilidad y capacidad de análisis de los sistemas mecatrónicos.

## Referencias bibliográficas


[1] N. Navani, S. Sharma y M. Sapra, *Instrumentación virtual con LabVIEW* . Nueva Delhi, India: Tata McGraw-Hill Education, 2011.

[2] R. Bishop, *Aprendiendo con LabVIEW 2020.* Upper Saddle River, NJ, EE. UU.: Pearson Education, 2020.

[3] JW Dally, WF Riley y KG McConnell, *Instrumentation for Engineering Measurements* , 2.ª ed. Nueva York, NY, EE. UU.: Wiley, 2022.

[4] National Instruments, “Introducción a la instrumentación virtual”, *NI Technical Report* , Austin, TX, EE. UU., 2023. [En línea]. Disponible en: https://www.ni.com

[5] A. Gilat y V. Subramaniam, *Introducción a la programación con MATLAB* , 6.ª ed. Hoboken, NJ, EE. UU.: Wiley, 2018.

# INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉSTUXTLA

**PRACTICA UII**

**CARRERA**

**INGENIERÍA MECATRÓNICA**

**PRESENTA**

**FRANCISCO EDUARDO AZAMAR**

**GRUPO:**

**911-A**

**CATEDRÁTICO**

**DR. JOSÉ ÁNGEL NIEVES VÁSQUEZ**

**ASIGNATURA:**

**INSTRUMENTACIÓN VIRTUAL**

**SAN ANDRES TUXTLA VER, A 04 DE OCTUBRE DEL 2025**

## Introducción

En los procesos industriales, el control de variables es un aspecto fundamental para garantizar la eficiencia, la seguridad y la calidad de los productos. Variables como la temperatura, la presión, el caudal o el nivel deben mantenerse dentro de rangos específicos para asegurar el correcto funcionamiento de los equipos y la estabilidad del proceso. Para lograrlo, se emplean sistemas de control automático que permiten regular estas variables en tiempo real, compensando perturbaciones externas y minimizando errores.

Dentro de los diferentes métodos de control, el control PID (Proporcional, Integral y Derivativo**)** se ha consolidado como uno de los más utilizados en la industria debido a su simplicidad, robustez y capacidad de adaptación a una amplia variedad de sistemas. Este controlador ajusta continuamente la señal de salida de acuerdo con la diferencia entre el valor deseado (setpoint) y el valor real de la variable, corrigiendo desviaciones de forma eficiente.

En esta práctica se desarrolla una simulación del control de temperatura mediante un controlador PID implementado en el lenguaje Python, con el objetivo de comprender el comportamiento dinámico del sistema y observar cómo las acciones proporcionales, integrales y derivativas influyen en la estabilidad y respuesta del proceso. De esta manera, se destaca la relevancia del control automático como una herramienta esencial para la optimización y automatización de los sistemas industriales modernos.

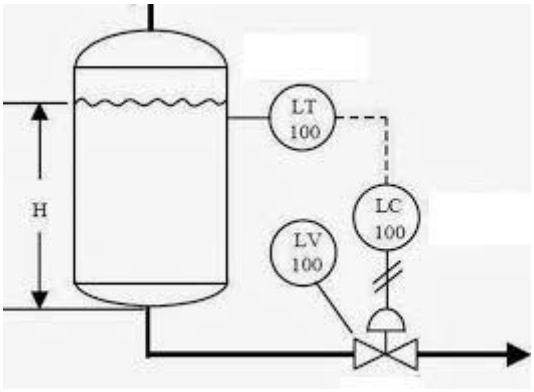
## Objetivo de la práctica

Diseñar una simulación de un proceso industrial donde se mida alguna variable física (temperatura, nivel, flujo o presión), utilizando cualquier software de diseño o programación.

## Marco teórico

Una variable de proceso es una condición física o química que es de interés medir y controlar, ya que puede alterar la producción o manufactura. En la instrumentación industrial se consideran las siguientes cuatro variables como las principales:

- Presión
- Temperatura
- Nivel
- Flujo




### Presión

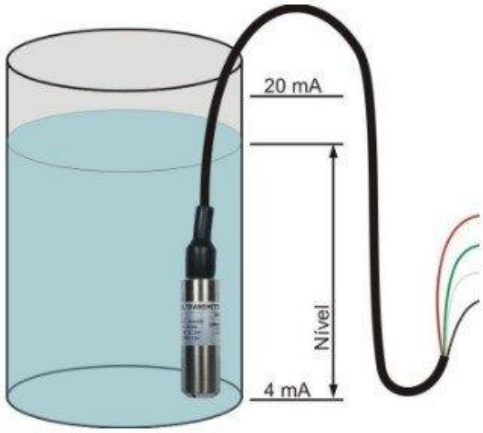
En ingeniería, el término presión se refiere generalmente a la fuerza ejercida por un fluido por unidad de área de la superficie que lo encierra. Existen muchas razones por las cuales en un determinado proceso se debe medir presión. La calidad del producto frecuentemente depende de ciertas presiones que se deben mantener en un proceso. Por seguridad, en recipientes presurizados donde no debe exceder un valor máximo dado por las especificaciones del diseño, también en aplicaciones de medición de nivel,

**Temperatura**

La temperatura es una de las variables más usadas en la industria de control de procesos y básica para medición y control de flujo, así como de la densidad, entre otros. Su medición y control, son vitales para asegurar uniformidad en la calidad de los productos terminados y mantenerse dentro de los límites seguros en operaciones que tengan riesgos de fuego y/o explosión.

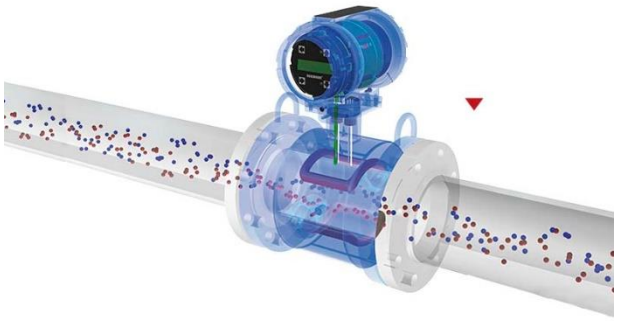
**Nivel**

Su aplicación más frecuente es la medición del volumen de líquidos, aunque también se emplea en almacenamiento de materiales sólidos. Algunos ejemplos de su utilidad son el control y la medición para evitar que un líquido se derrame, la medición de nivel de un depósito, el control de contenido corrosivo, abrasivo, en altas presiones, radiactivo, entre otros. En algunos fluidos junto con el nivel se mide la temperatura para calcular el volumen ya compensado.



**Flujo**

Es la cantidad de fluido que pasa en una unidad de tiempo. Normalmente se identifica con el flujo volumétrico o másico que pasa por un área dada en una unidad de tiempo. Las aplicaciones son muchas, desde las más sencillas, como la medición de flujo de agua en estaciones de tratamiento y residencias, hasta medición de gases industriales y combustibles, pasando por mediciones más complejas.



**Simulación de variables físicas mediante software**

Las simulaciones de física son herramientas educativas que permiten recrear fenómenos físicos de forma interactiva y visual. Su utilidad en el aprendizaje radica en que facilitan la comprensión de conceptos abstractos, estimulan el razonamiento científico y fomentan la experimentación. Además, pueden ayudar a los estudiantes a desarrollar habilidades como la resolución de problemas, el pensamiento crítico y la creatividad.

## Beneficios de utilizar simulaciones de física en el aula

- Permiten más experiencias de laboratorio

Con simulaciones virtuales, los estudiantes pueden visualizar e interactuar con conceptos fuera del ámbito de un laboratorio tradicional. Por ejemplo, pueden manipular parámetros o variables como la gravedad y la fricción. También pueden recrear experimentos importantes de Física, pero que son demasiado laboriosos, riesgosos o costosos de realizar.

- Facilitan la participación de todos los estudiantes.

Las simulaciones permiten que los estudiantes puedan practicar e interactuar con sus compañeros al compartir sus experiencias y discutir los resultados obtenidos.

- Permiten que los estudiantes retengan mejor lo que han aprendido.

Actualmente, las simulaciones son más que aplicaciones de hacer clic y reproducir; algunas permiten una experiencia de laboratorio completa, ya que son diseñadas para funcionar como un laboratorio real. Utilizarlas en el aula de clases, ya sea de forma presencial o virtual, puede mejorar el desarrollo de habilidades de los estudiantes, así como su conocimiento conceptual.

- Proporcionan retroalimentación inmediata

Las simulaciones permiten que los estudiantes, en caso de cometer errores, puedan corregir conceptos o datos erróneos, así como a los profesores intervenir en el proceso de aprendizaje cuando sea necesario.

- Disponibilidad las 24 horas

Los estudiantes puede acceder en cualquier momento del día a las simulaciones virtuales, lo que brinda una excelente opción para recuperar clases perdidas o repetir experimentos sin preocuparse por costos de mantenimiento o materiales limitados.

- Son seguras y rentables

En el caso de simulaciones de experimentos, los profesores pueden brindar asistencia a sus estudiantes de forma individual sin tener que estar monitoreando continuamente el cumplimiento de los procedimientos y la seguridad. Además, complementar las clases con simulaciones proporciona a cada estudiante una experiencia práctica y reduce la cantidad gastada en costosos instrumentos y suministros de laboratorio.

## Softwares de simulación de uso libre

### Python

Python es un lenguaje de programación informática que se utiliza a menudo para crear sitios web y software, automatizar tareas y realizar análisis de datos. Python es un lenguaje de propósito general, lo que significa que se puede utilizar para crear una variedad de programas diferentes y no está especializado en ningún problema específico. Esta versatilidad, junto con su facilidad para los principiantes, lo ha convertido en uno de los lenguajes de programación más utilizados en la actualidad.

### Algodoo

Algodoo es un software de simulación de física 2D al estilo “sandbox” (caja de arena), desarrollado por Algorix Simulation AB, que permite a los usuarios crear escenarios interactivos y experimentar con diferentes situaciones físicas, ya que permite controlar una gran cantidad de variables y parámetros, como la gravedad, fricción, índice de refracción, densidad, presión, flotabilidad, entre otros. Es fácil de usar y tiene una interfaz intuitiva, tiene amplio soporte para acelerómetros, pantallas táctiles, e Intel Classmate PC. El motor físico del software está basado en la constante lineal SPOOL. También incluye contenido como planes de lecciones, escenas prediseñadas y tutoriales.

**Physion**

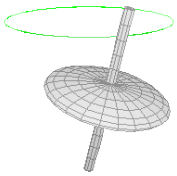
Physion es un software de simulación de física gratuito que permite diseñar y simular experimentos de física realistas en un mundo físico virtual. Puede ser considerado como una aplicación CAD combinada con un motor de física 2D donde los objetos que se diseñan pueden simularse instantáneamente.

**Tracker. Video Analysis and modeling tool**

Tracker. Video Analysis and modeling tool es una herramienta gratuita de análisis de video y construcción de modelos diseñados para ser usado en la enseñanza de la Física. Está basada en el marco Java Open Source Physics (OSP). Entre sus características se encuentra:

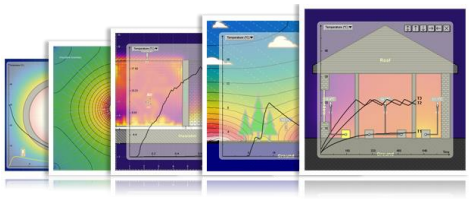
- Seguimiento de objetos manual y automatizado con superposiciones y datos de posición, velocidad, y aceleración.
- La opción Model Builder crea modelos cinemáticos y dinámicos de partículas de masa puntual y sistemas de dos cuerpos.
- El motor de video gratuito Xuggle reproduce y graba muchos formatos (mov, avi, flv, mp4, wmv.) en Windows, MacOS y Linux.
- Posee herramientas de análisis de datos que incluyen ajuste de curvas manual o automático.
- Cuenta con una extensa biblioteca de recursos digitales.

**Easy Javascript Simulations**



Easy Javascript Simulations (EJSS), anteriormente conocido como Easy Java Simulations, es una herramienta de autoría gratuita escrita en Java que ayuda a los no programadores a crear simulaciones interactivas en Java o Javascript, principalmente con fines educativos. EJS ha sido creado por Francisco Esquembre y forma parte del proyecto Open Source Physics.

**Energy2D**



Energy2D es un software de simulación de física que se basa en la investigación de la física computacional y modela los tres mecanismos de transferencia del calor: conducción, convección y radiación. Es una herramienta útil para la enseñanza de las ciencias experimentales, ya que permite a los estudiantes comprender mejor los conceptos abstractos y complejos de la física, y les brinda la oportunidad de experimentar con diferentes situaciones y escenarios.

Simphy

Simphy es un software de simulación de física que ofrece simulaciones interactivas en 3D de electricidad, óptica, mecánica, fluidos, entre otros. Los profesores pueden crear sus propias simulaciones con herramientas de arrastrar y soltar, y acceder a una gran biblioteca de simulaciones (alguna de ellas gratis).

## Desarrollo

Para la realización de esta práctica, se decidió realizar un código en Python que permita visualizar una simulación del control de una variable física en tiempo real, para este ejercicio se utilizó la variable de temperatura por ser una de las más manipulables.

Dicho proceso se trató de desarrollar mediante un controlador PID para obtener un a mejor monitorización de la variable y de esta forma llegar a un mejor equilibro en un menor tiempo.

A continuación, se muestra el código diseñado en el entorno de Python.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct  4 18:06:40 2025
@author: Francisco
"""

import tkinter as tk
from tkinter import ttk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg
import numpy as np

# Parámetros del sistema
Kp = 2.0
Ki = 0.1
Kd = 1.0

# Variables de simulación
setpoint = 60.0
temperatura = 25.0
```

```
error_prev = 0.0
integral = 0.0
running = False
tiempo = [0]
valores_temp = [temperatura]
valores_set = [setpoint]

# Ventana principal
root = tk.Tk()
root.title("Simulación de Control de Temperatura con PID")
root.configure(bg="#eef2f3")

# Crear figura de Matplotlib
fig, ax = plt.subplots(figsize=(6, 3))
ax.set_title("Evolución de la temperatura")
ax.set_xlabel("Tiempo (s)")
ax.set_ylabel("Temperatura (°C)")
linea_temp, = ax.plot(tiempo, valores_temp, "r-",
label="Temperatura")
linea_set, = ax.plot(tiempo, valores_set, "b--",
label="Setpoint")
ax.legend()
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().grid(row=0, column=1, rowspan=10,
padx=15, pady=10)

# --- Elementos GUI (lado izquierdo) ---
tk.Label(root, text="Setpoint (°C):",
bg="#eef2f3").grid(row=0, column=0, pady=5)
entry_setpoint = ttk.Entry(root)
entry_setpoint.insert(0, "60.0")
```

```
entry_setpoint.grid(row=1, column=0)


label_temp = tk.Label(root, text=f"Temperatura actual:
{temperatura:.1f} °C", font=("Arial", 12), bg="#eef2f3")
label_temp.grid(row=2, column=0, pady=5)


label_potencia = tk.Label(root, text="Potencia del
calentador: 0%", font=("Arial", 10), bg="#eef2f3")
label_potencia.grid(row=3, column=0, pady=5)


label_estado = tk.Label(root, text="Calentador: APAGADO",
font=("Arial", 11, "bold"), fg="gray", bg="#eef2f3")
label_estado.grid(row=4, column=0, pady=5)


# --- Canvas para la caldera ---
canvas_caldera = tk.Canvas(root, width=200, height=200,
bg="#dfe6e9")
canvas_caldera.grid(row=5, column=0, pady=10)
rect_caldera = canvas_caldera.create_rectangle(60, 80, 140,
140, fill="blue")
label_caldera = tk.Label(root, text="Caldera", bg="#eef2f3")
label_caldera.grid(row=6, column=0)


# --- Botones ---
def iniciar():
    global running
    running = True
    simular()


def detener():
    global running
    running = False
```

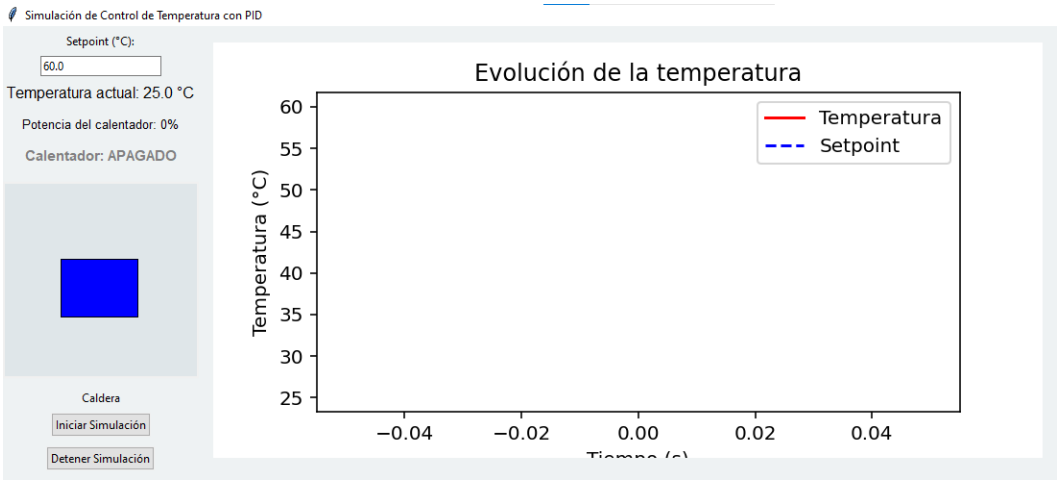
```
btn_iniciar = ttk.Button(root, text="Iniciar Simulación",
command=iniciar)
btn_iniciar.grid(row=7, column=0, pady=5)

btn_detener = ttk.Button(root, text="Detener Simulación",
command=detener)
btn_detener.grid(row=8, column=0, pady=5)

# --- Lógica PID y simulación ---
def simular():
    global temperatura, integral, error_prev, running, tiempo

    if not running:
        return

    try:
        sp = float(entry_setpoint.get())
    except ValueError:
        sp = setpoint

    error = sp - temperatura
    integral += error
    derivada = error - error_prev
    potencia = Kp * error + Ki * integral + Kd * derivada
    potencia = max(0, min(potencia, 100))  # Límite 0-100%
    error_prev = error

    # Dinámica del sistema
    temperatura += (potencia * 0.05) - (temperatura - 25) *
0.02
```

```
    # Actualización visual
    label_temp.config(text=f"Temperatura actual: {temperatura:.1f} °C")
    label_potencia.config(text=f"Potencia del calentador: {potencia:.1f}%")
    label_estado.config(text="Calentador: ENCENDIDO", fg="green" if potencia > 0 else "gray")

    # Color de caldera según temperatura
    if temperatura < 25:
        color = "#87CEFA"  # Azul claro
    elif 25 <= temperatura <= 35:
        color = "blue"
    elif 36 <= temperatura <= 45:
        color = "yellow"
    elif 46 <= temperatura <= 60:
        color = "orange"
    else:
        color = "red"

    canvas_caldera.itemconfig(rect_caldera, fill=color)

    # Actualizar gráfico
    tiempo.append(tiempo[-1] + 1)
    valores_temp.append(temperatura)
    valores_set.append(sp)
```

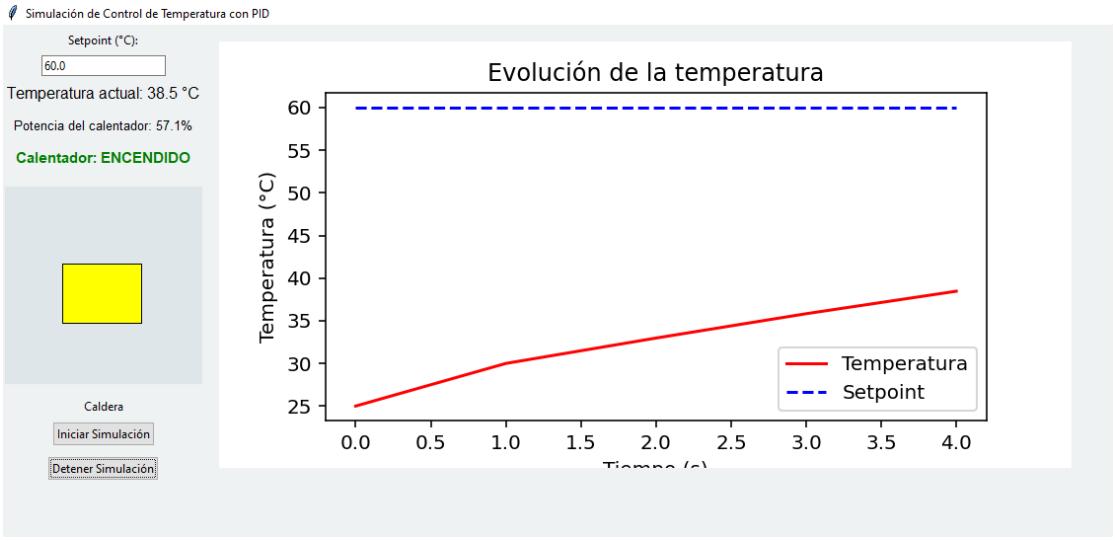
```
    linea_temp.set_data(tiempo, valores_temp)
    linea_set.set_data(tiempo, valores_set)
    ax.relim()
    ax.autoscale_view()
    canvas.draw()

    root.after(200, simular)  # Llamada periódica (200 ms)

root.mainloop()
```

## Resultados

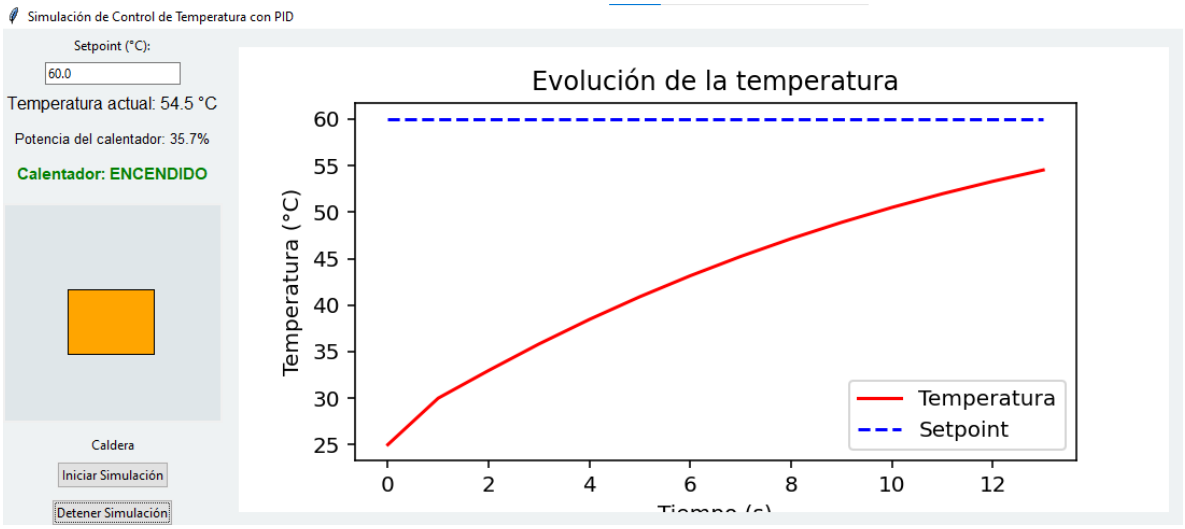
Después de revisar posibles errores en el código, procedemos a ejecutarlo en la consola y visualizar los datos obtenidos.



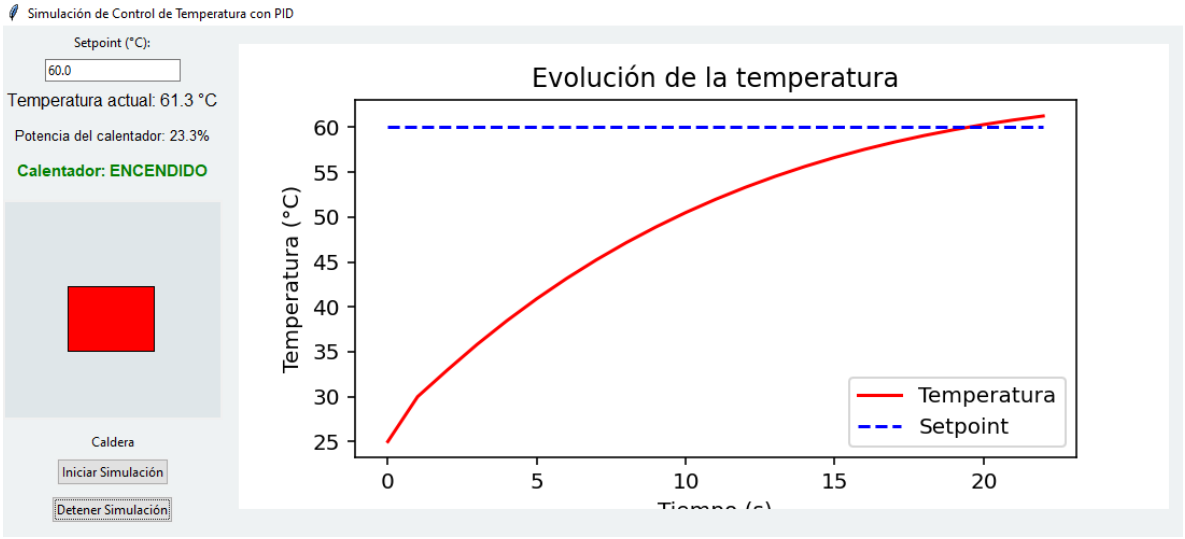

Esta es la interfaz diseñada, en la cual se pueden apreciar los valores iniciales de la temperatura de nuestro proceso, en este caso se eligió una caldera hipotética, de igual forma podemos apreciar el valor del setpoint, en este caso es de 60°, la ilustración de la caldera cambiara de color conforme aumente o disminuya la temperatura del proceso.



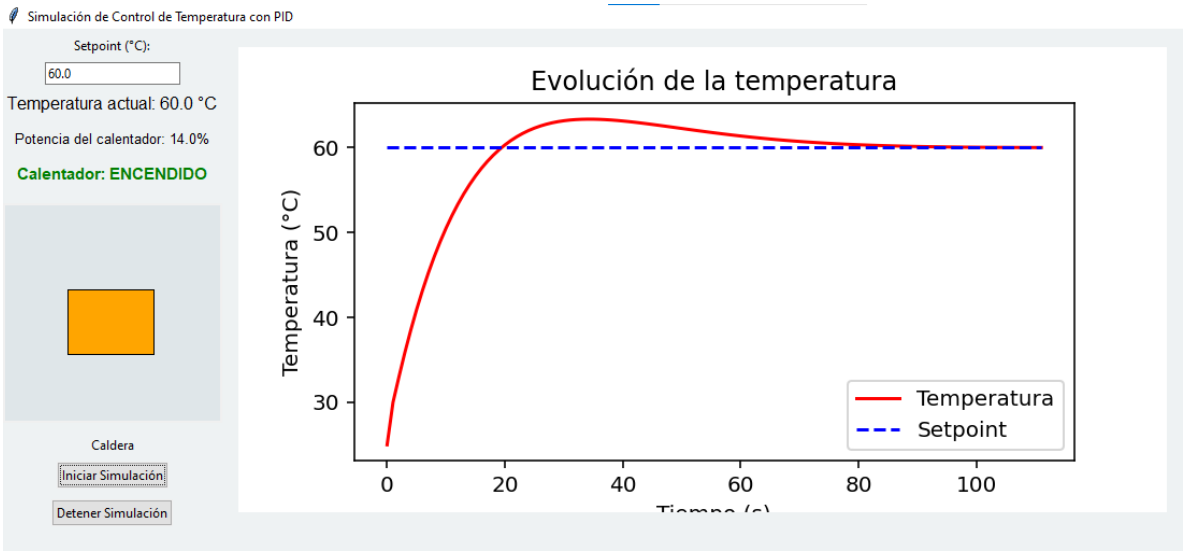

Al iniciar con la simulación, se aprecia que la caldera cambia a un color amarillo cuando la temperatura sobrepasa los 35°, mientras que la potencia del calentador se encuentra a un 57% y sigue en acenso para llegar al valor deseado.




Conforme pasa el tiempo, la temperatura sigue subiendo y nuestra representación de la caldera cambia de color.




Después de sobrepasar el valor del setpoint establecido, la tonalidad del grafico es roja, simulando un estado de alerta, el cual inicia un proceso inverso para controlar el valor de la temperatura del proceso.




De esta forma tenemos la simulación completa, apreciamos la forma que adopta la gráfica de temperatura con respecto al tiempo, para este proceso podemos afirmar que nuestro proceso se estabiliza alrededor de los 80s.

## Conclusiones

La práctica realizada permitió comprobar la utilidad de Python como herramienta para el modelado y simulación de sistemas de control en procesos industriales. A través de la implementación de un algoritmo de regulación de temperatura, se observó el comportamiento dinámico del proceso y la manera en que el controlador ajusta la variable manipulada para mantener la variable de salida dentro de los valores deseados. Esta experiencia no solo reforzó los conceptos teóricos del control de procesos, sino que también evidenció la importancia de la simulación computacional como etapa previa a la implementación real, ya que permite analizar el desempeño del sistema, detectar posibles mejoras y optimizar parámetros de manera segura y eficiente.

## Referencias bibliográficas


1. Cietsa_Web, “Principales variables de los procesos industriales,” *Cietsa*, Aug. 01, 2023. https://cietsa.com.mx/principales-variables-de-los-procesos-industriales/
2. A. Interactiva and A. Interactiva, “Los 6 mejores programas para crear tus propias simulaciones de física,” *Aula Interactiva*, Nov. 23, 2023. [Online]. Available: https://aulainteractiva.com.ve/simulaciones-de-fisica/
3. C. Staff, “¿Qué es Python y para qué se usa? Guía para principiantes,” *Coursera*, Nov. 29, 2023. https://www.coursera.org/mx/articles/what-is-python-used-for-a-beginners-guide-to-using-python

# LISTA DE COTEJO INVESTIGACION

# INSTRUMENTACIÓN VIRTUAL.

Nombre del estudiante: FRANCISCO EDUARDO AZAMAR.

Tema: APLICACIONES REALES DE LA INSTRUMENTACIÓN VIRTUAL.

| | | |
|---|---|---|
| Portada | 2 % | 2 % |
| Introducción | 5 % | 5 % |
| Desarrollo | 10 % | 10 % |
| Conclusiones | 5 % | 5% |
| Referencias | 3 % | 3 % |
| Entrega en tiempo y forma | 5 % | 5 % |
| Examen diagnostico | 10 % | 10 % |
| Total | 40 % | 40% |

# LISTA DE COTEJO DE PRÁCTICAS

# INSTRUMENTACIÓN VIRTUAL.

Nombre del estudiante: FRANCISCO EDUARDO AZAMAR.

Tema: PRÁCTICA NÚMERO U2

| | | |
|---|---|---|
| Portada | 5 % | 5 % |
| Introducción | 10 % | 10 % |
| Desarrollo | 30 % | 30 % |
| Conclusiones | 5 % | 5 % |
| Referencias | 2 % | 2 % |
| Entrega en tiempo y forma | 8 % | 8 % |
| Total | 60 % | 60 % |