Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica
Departamento de Control Automático

ASIGNATURA: Ingeniería Automática

INFORME TÉCNICO DEL PROYECTO

Mejoras al software de configuración de registradores de datos basados en Arduino y ESP32

Estudiante: Daniel de la Cruz Medina
Grupo: 4to de Automática
Profesor: Dr.C Alain Martínez Laguardia

Santa Clara, Cuba

Contents

1. Introducción

Los registradores de datos basados en microcontroladores de bajo costo han demostrado ser una alternativa viable para el monitoreo ambiental y otras aplicaciones donde se requieren mediciones continuas, autonomía energética y despliegue en contextos con recursos limitados. La tesis doctoral de Erik Hernández Rodríguez **[2]** presenta un método de diseño para registradores de datos de bajo costo orientados a la calidad del aire, incluyendo una arquitectura modular y un software HMI para configurar la plataforma y los sensores mediante archivos en formato JSON.

El presente informe técnico se centra en la aplicación de configuración ya desarrollada en C# (Windows Forms), utilizada como base para configurar registradores de datos basados en Arduino y ESP32. A partir de la síntesis funcional obtenida directamente del código fuente, se formulan recomendaciones de mejoras de software, de bajo costo y alta viabilidad, alineadas con los requisitos del curso de Ingeniería Automática.

2. Objetivos y alcance

2.1 Objetivo general

Sugerir mejoras al software de configuración de registradores de datos basados en Arduino y ESP32.

2.2 Objetivos específicos (derivados de los requisitos del proyecto)

- Analizar y sintetizar el funcionamiento de la aplicación de configuración existente (C#, WinForms) a partir del código fuente.
- Documentar la arquitectura de datos y el flujo de configuración, incluyendo los archivos JSON generados por el software.
- Definir un conjunto de pruebas funcionales reproducibles que verifiquen el comportamiento implementado en la aplicación.
- Proponer mejoras de software, de bajo costo, priorizadas por impacto y facilidad de implementación.

2.3 Alcance y restricciones

El alcance del proyecto se limita a la capa de software (aplicación HMI en PC, estructura de archivos de configuración y mecanismos de comunicación expuestos por la aplicación). No se rediseña hardware ni se asume la disponibilidad de un prototipo físico adicional. La descripción del hardware se presenta como referencia técnica derivada de la arquitectura modular de la tesis y de las opciones de 'Socket' integradas en la interfaz.

3. Marco teórico

3.1 Registradores de datos y arquitectura modular

En la tesis de referencia **[2]** se establece una arquitectura para sistemas de monitoreo basada en plataformas de desarrollo de bajo costo, donde un microcontrolador actúa como núcleo del sistema y se integra con módulos de comunicación, servidor de datos, visualización, notificaciones e interfaz de configuración. Esta separación por componentes favorece la escalabilidad, la sostenibilidad y la adaptación a restricciones de recursos.

En dicha arquitectura, la interfaz de configuración permite ajustar parámetros del sistema y del sensor. La configuración se materializa en archivos (por ejemplo, JSON) que son consumidos por el firmware de operación en el microcontrolador, reduciendo la necesidad de recompilación para ajustes operativos y apoyando la trazabilidad de la configuración empleada en cada experimento.

3.2 Formatos de configuración y JSON

En [2] se describe el uso de archivos JSON generados por el software HMI para configurar tanto parámetros generales de la plataforma (unidad de procesamiento, periodo de muestreo, baudios de comunicación serial, parámetros Wi-Fi, etc.) como especificaciones inherentes a los sensores seleccionados. La estructura tipo “arreglo de elementos” facilita incorporar múltiples características de forma organizada y soporta la ontología del sensor (descripciones y observaciones).

3.3 Comunicación serial y configuración de microcontroladores

La comunicación serial (UART a través de un puerto virtual COM por USB) es un mecanismo habitual para la configuración y depuración de registradores de datos basados en Arduino/ESP32. Desde el punto de vista del software de configuración, las operaciones típicas incluyen la selección del puerto, la razón de baudios y el envío de mensajes estructurados (por ejemplo, JSON) hacia el dispositivo.

3.4 Validación, verificación y pruebas en software embebido

Los procesos de verificación y validación buscan asegurar que el software cumple sus especificaciones y que satisface las expectativas del usuario final. En un entorno embebido, este ciclo se extiende a la consistencia de la configuración, al manejo de fallos en comunicación y a la reproducibilidad experimental.

4. Descripción del sistema.

4.1 Descripción del hardware de referencia y mapeo de sockets

La aplicación de configuración expone un mapeo de conectividad por "socket" dentro de la interfaz (DataForm), que permite documentar la convención de asignación de buses/interfases utilizada por el configurador. En el formulario de configuración se ofrecen 14 opciones de socket, agrupadas por tipo de interfaz, lo que facilita la organización modular de la configuración desde la HMI.

Socket	Interfaz / descripción (según la aplicación)
1–5	I2C
6–8	UART
9	SPI
10	AFE (voltage analog sensor)
11	AFE (4 Alphasense gas sensors)
12–14	AFE (voltage analog sensor)

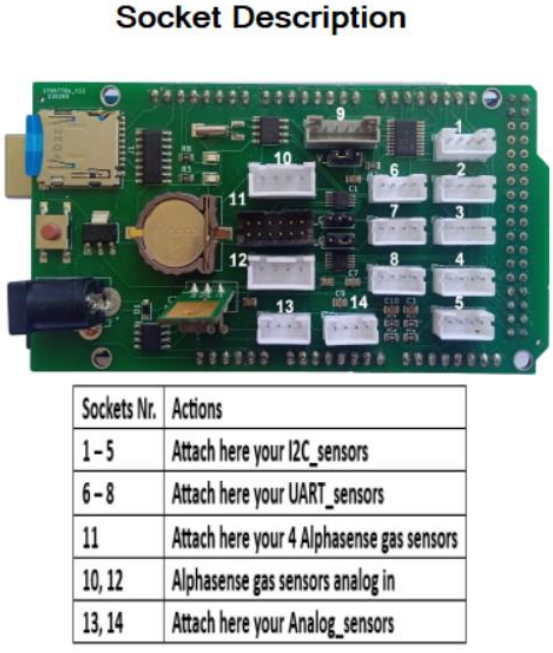

Figura 1. Descripción de sockets

4.2 Software de configuración existente (C#, Windows Forms)

4.2.1 Estructura general del software

La aplicación está desarrollada en C# sobre Windows Forms y se organiza en un formulario principal (Form1) y un formulario secundario de edición (DataForm). Además, define clases de datos para representar configuraciones (MatrixConfig, SystemConfig) y una clase estática (DataConfiguration) que mantiene la lista de configuraciones en memoria durante la ejecución.

4.2.2 Form1: configuración general, visualización y exportación

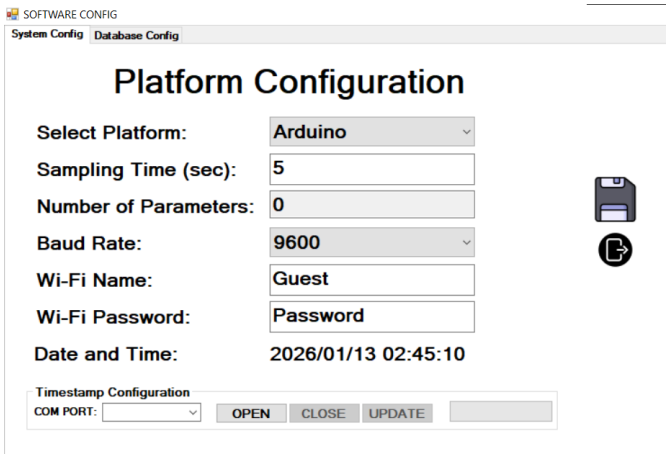

Figura 2. Form1

El formulario principal implementa las siguientes funciones:

- Inicialización de parámetros por defecto: tiempo de muestreo, plataforma (Arduino), Wi-Fi y lista de puertos COM.
- Gestión de configuración general (SystemConfig) y exportación a archivo JSON (guardar mediante diálogo).
- Gestión de base de datos de parámetros: visualizar en DataGridView, añadir/editar/eliminar filas.
- Gestión de puerto serie: abrir/cerrar COM y enviar un JSON de fecha/hora ('update').
- Exportación de un archivo JSON simplificado con {Property_Id, Sensor_Id, Statistic_Id} por parámetro configurado.

4.2.3 DataForm: alta/edición de parámetros y validación

El formulario secundario permite construir o editar una configuración por parámetro (MatrixConfig). El usuario selecciona: propiedad/parámetro, sensor, socket, análisis matemático

y unidad/título del parámetro. El software incluye validación básica: impide confirmar si alguno de los campos obligatorios está vacío, mostrando un mensaje de error.

4.2.4 Modelo de datos

Las clases de datos definen la estructura mínima de configuración utilizada por la aplicación:

- MatrixConfig: Id, Sensors, Parameters, Socket_Number, Math_Analysis, Parameter_Title (unidad), Property_Id, Sensor_Id, Statistic_Id.
- SystemConfig: Devices, Sampling_Time, Baudrate, WiFi_Name, WiFi_Password, Date_Time.
- Updatedatetime: DTYear, DTMonth, DTDay, DTHour, DTMinute, DTSecond.

4.3 Archivos de configuración generados (JSON)

A partir del comportamiento implementado, se generan tres tipos principales de JSON:

4.3.1 Archivo de configuración general (SystemConfig)

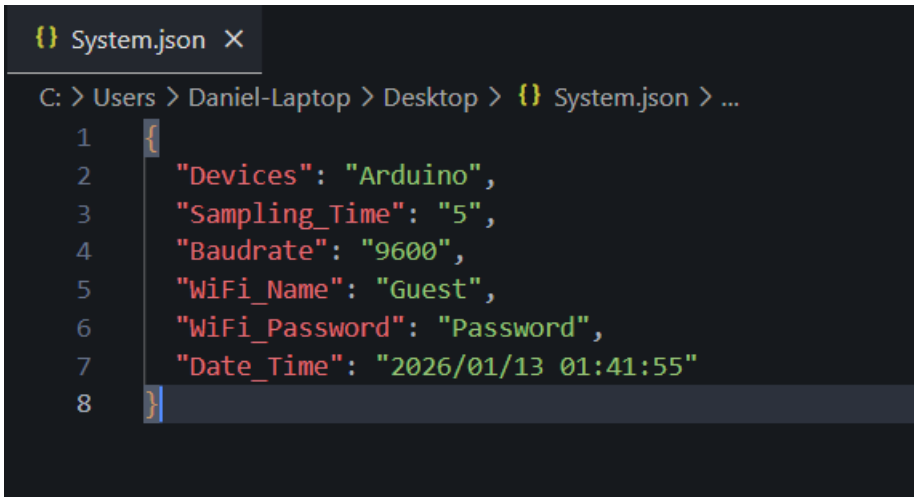

Figura 3. Configuración de sistema

4.3.2 Archivo de configuración de parámetros (base de datos simplificada)

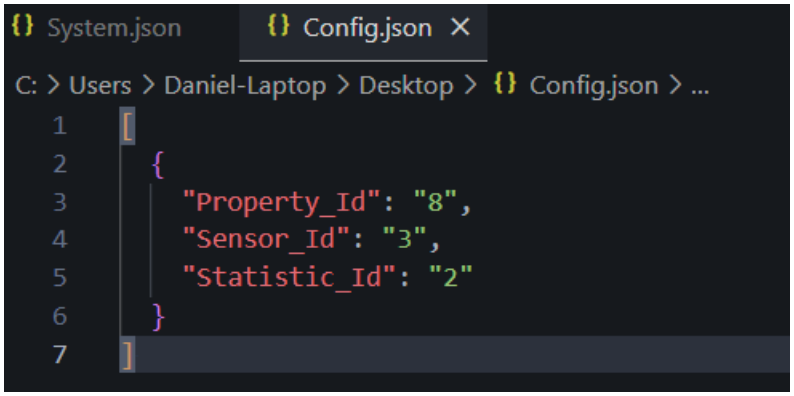

Figura 4. Ejemplo de configación de parámetros

4.3.3 Mensaje de actualización de fecha/hora (Updatedatetime)

```
{
  "DTYear": "2026",
  "DTMonth": "1",
  "DTDay": "13",
  "DTHour": "12",
  "DTMinute": "34",
  "DTSecond": "56"
}
```

5. Metodología de desarrollo y pruebas

5.1 Enfoque metodológico

La elaboración del presente informe técnico se sustenta en un enfoque de ingeniería inversa y documentación, basado en: (i) análisis estático del código fuente para extraer arquitectura, flujo y modelos de datos; (ii) definición de una matriz de pruebas funcionales reproducibles; y (iii) formulación de recomendaciones de mejora alineadas con los requisitos de configurabilidad y diagnóstico.

5.2 Plan de pruebas funcionales (PoC)

La Tabla 5-1 describe pruebas reproducibles para verificar el comportamiento implementado en la aplicación.

ID	Prueba	Entrada / condición	Procedimiento	Resultado esperado
T1	Añadir	App abierta	Icono Añadir → completar	Nueva fila en tabla y

	parámetro		DataForm → Confirmar	actualización de contador
T2	Editar parámetro	Una fila seleccionada	Icono Editar → modificar campos → Confirmar	Fila actualizada en tabla
T3	Eliminar parámetro	Una fila seleccionada	Icono Eliminar	Fila eliminada y reindexación de IDs
T4	Guardar SystemConfig	Campos válidos	Icono Guardar (general) → elegir ruta	Archivo JSON de SystemConfig creado
T5	Guardar base de datos	Lista con N parámetros	Icono Guardar (database) → elegir ruta	Archivo JSON con N objetos {Property_Id, Sensor_Id, Statistic_Id}
T6	Serial: abrir/cerrar	COM disponible	Seleccionar COM → Abrir → Cerrar	Barra 100% al abrir, 0% al cerrar; botones habilitados
T7	Serial: update datetime	Puerto abierto	Presionar Update	Envío por serial de JSON Updatedatetime

5.3 Gestión del repositorio GitHub

El repositorio público del proyecto se encuentra en:

(https://github.com/TU_USUARIO/proyecto-ia-mejoras-software-registrador-datos).

Contiene el código fuente (/src), el informe y documentación (/docs), evidencias visuales (/assets) y ejemplos de archivos JSON exportados (/examples). El archivo README.md describe los pasos de compilación/ejecución y la reproducción de las pruebas T1–T7.

6. Resultados y análisis

6.1 Resultados del análisis del software

Del análisis del código fuente se obtuvieron los siguientes resultados técnicos verificables:

- El software implementa un flujo CRUD para configuraciones por parámetro (MatrixConfig), visualizadas en una tabla (DataGridView).
- Existe exportación a JSON de configuración general (SystemConfig) y de una base de datos simplificada de parámetros.
- La interfaz incluye opciones de socket que codifican un diseño modular por bus (I2C/UART/SPI/AFE).
- La validación de DataForm es de tipo 'campos no vacíos'; si falta un campo, la aplicación muestra un mensaje de error.

6.2 Resultados de referencia reportados en la tesis

La tesis **[2]** reporta el uso de un software HMI para generar archivos de configuración JSON tanto del sistema como de los sensores. En el experimento descrito, se seleccionó Arduino como unidad de procesamiento, se configuró un periodo de muestreo de 2 minutos, se introdujeron 19 parámetros y se estableció una razón de baudios de 9600 para la comunicación serial. Además, se describe una segunda pantalla vinculada a las especificaciones de sensores, con funcionalidades de adición, edición, eliminación y almacenamiento, así como algoritmos de autocompletamiento y validación de campos.

6.3 Limitaciones identificadas

Las siguientes limitaciones se identifican directamente por inspección del código fuente:

- Persistencia incompleta: existen trazas de lectura/escritura de la lista matrixConfigs, pero se encuentran comentadas; por tanto, la configuración no se carga automáticamente al iniciar.
- Rutas por defecto dependientes de equipo: Config_PATH y System_PATH se inicializan con rutas absolutas (por ejemplo, D:\temp y un escritorio de usuario), lo cual reduce portabilidad.
- Posible sobrescritura del archivo de sistema: la acción de 'update datetime' escribe el JSON de fecha/hora en System_PATH, que también se usa para guardar SystemConfig.

- Exportación de base de datos simplificada: el archivo guardado en 'database' conserva solo IDs (Property/Sensor/Statistic) y no guarda metadatos como socket o unidad, lo que puede limitar diagnósticos posteriores.
- Catálogo de sensores y parámetros hardcode: listas extensas en la interfaz dificultan añadir sensores sin recompilar.

7. Conclusiones

La síntesis realizada a partir del código fuente permite documentar el funcionamiento del software de configuración existente para registradores basados en Arduino/ESP32, identificando su arquitectura WinForms, su modelo de datos y los formatos JSON generados. El flujo implementado es coherente con el enfoque descrito en la tesis de referencia: una interfaz HMI que habilita la selección de parámetros, la configuración de sensores y la generación de archivos de configuración consumibles por el microcontrolador.

Como resultado, se dispone de una base documental lista para sustentar la demostración del sistema y servir como punto de partida para mejoras de software de bajo costo orientadas a: (i) facilitar la incorporación de nuevos sensores; (ii) aumentar robustez y portabilidad; y (iii) fortalecer diagnóstico y trazabilidad de la configuración.

8. Recomendaciones

Las recomendaciones siguientes se plantean como mejoras de bajo costo y alta viabilidad, priorizadas por impacto operativo y por alineación con el objetivo de incorporar sensores de forma más amigable.

8.1 Importación de configuraciones y validación de esquema JSON

- Incorporar una función 'Importar' que lea archivos JSON previamente generados y reconstruya la lista de configuraciones.
- Añadir validación de esquema (campos requeridos, tipos, rangos) antes de cargar, mostrando mensajes claros al usuario.

Beneficio: cierra el ciclo de configuración (export/import) y mejora reproducibilidad experimental.

8.2 Catálogo de sensores parametrizable (eliminar hardcode)

- Extraer la lista de parámetros, sensores y unidades a un archivo externo (por ejemplo, sensors_catalog.json).
- Cargar el catálogo al iniciar la aplicación y poblar combobox dinámicamente.

Beneficio: habilita añadir sensores sin recompilar y reduce el costo de mantenimiento del configurador al trasladar el catálogo de sensores/parámetros a un archivo editable.

8.3 Corrección de persistencia y portabilidad de rutas

- Reactivar (o reimplementar) la carga/guardado automático de matrixConfigs desde un archivo en una ruta relativa del proyecto.
- Evitar rutas absolutas; usar rutas en carpeta de aplicación o en Documentos del usuario.

Beneficio: mejora portabilidad y continuidad entre sesiones.

8.4 Diagnóstico y registro (logging) de comunicación serial

- Añadir panel de estado: puerto seleccionado, estado abierto/cerrado, errores y última trama enviada.
- Registrar eventos a un archivo .log (apertura/cierre, envíos, errores).

Beneficio: incrementa trazabilidad y facilita soporte técnico.

8.5 Consistencia de archivos de configuración

- Separar rutas/archivos: SystemConfig.json y DateTime.json (o incorporar fecha/hora como parte del SystemConfig).
- Evitar sobrescrituras y documentar la convención de nombres en el README.

Beneficio: elimina un riesgo de pérdida de configuración y simplifica el flujo del usuario.

9. Referencias bibliográficas

[1] Departamento de Control Automático, “Proyectos de Ingeniería Automática” (guía de orientación y criterios de evaluación), Universidad Central “Marta Abreu” de Las Villas, Santa Clara, Cuba, s.f.

[2] E. Hernández Rodríguez, “Método para el diseño de registradores de datos de bajo costo para aplicaciones de calidad del aire”, Tesis doctoral, Universidad Central “Marta Abreu” de Las Villas, Santa Clara, Cuba, 2025.

[3] Espressif Systems, “ESP32 Series: Datasheet”, Espressif Systems, s.f.

[4] B. Blanchon, “ArduinoJson: biblioteca para serialización/deserialización JSON en microcontroladores”, documentación oficial, s.f.

[5] Microsoft, “System.IO.Ports.SerialPort Class,” documentación oficial de .NET, s.f.