Desarrollo de una aplicación de software utilizando Python que contribuya con la rapidez y calidad de las pruebas de pantallas Interfaz Hombre Máquina (Human Machine Interface) para utilizarse en compañías de automatización industrial

Doyler Amilkar Lastre Reinel

Trabajo de Grado para Optar el Título de Ingeniero Electrónico

Director
Alfredo Rafael Acevedo Picón
Ingeniero Electrónico

Codirector
José Alejandro Amaya Palacio
Doctor en Ingeniería eléctrica

Universidad Industrial de Santander
Facultad Ingenierías Físico-Mecánicas
Escuela de Ingeniería Eléctrica, Electrónica y Comunicaciones
Bucaramanga
2025

Dedicatoria

En primer lugar, dedico este logro a Dios, por colmarme de bendiciones y por concederme la fortaleza, la sabiduría y la licencia para culminar esta importante etapa de mi vida. Su guía ha sido la luz constante en mi camino.

Este trabajo también está dedicado a mis amados padres, Julio Manuel Lastre Lázaro y María Reinel Estrada. Gracias por su amor, sus valores y por sentar las bases de la persona que soy hoy.

De manera muy especial, quiero honrar a mi madre, quien siempre ha creído en mí. Tu esfuerzo incansable y tus sacrificios silenciosos han sido el motor que me ha impulsado a salir adelante. Este triunfo es el fruto de tu fe inquebrantable en mí, y te lo entrego con todo mi amor y gratitud.

.

Agradecimientos

Un agradecimiento especial al Ingeniero Alfredo Rafael Acevedo Picón, cuya pasión por la Automatización Industrial no solo despertó mi interés en el campo, sino que también sentó las bases de mi conocimiento. Su guía experta, su pedagogía y su invaluable orientación fueron fundamentales para la culminación exitosa de este proyecto. Gracias por ser un mentor y una inspiración.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Resumen

Título: Desarrollo de una aplicación de software utilizando Python que contribuya con la rapidez y calidad de las pruebas de pantallas Interfaz Hombre Máquina (Human Machine Interface) para utilizarse en compañías de automatización industrial[*]

Autor: Doyler Amilkar Lastre Reinel[**]

Palabras Clave: *HMI, SCADA, unit testing, python, PLC*

El presente proyecto aborda una problemática crítica en el sector de la automatización industrial: la ineficiencia y la falta de fiabilidad en las pruebas de pantallas HMI durante la fase de comisionamiento. Tradicionalmente, este proceso se realiza de forma manual, lo que implica una considerable inversión de tiempo, alta repetitividad y una constante exposición a errores humanos que pueden comprometer la calidad del sistema final.

Para dar solución a este desafío, se desarrolló Turbo Tester HMI, una aplicación de escritorio creada en Python. La herramienta está diseñada para automatizar y agilizar de manera radical el proceso de validación. Mediante una interfaz gráfica intuitiva, la aplicación permite al personal de automatización conectarse directamente a un PLC a través de su dirección IP. El núcleo de su funcionalidad radica en la capacidad de cargar configuraciones de prueba desde un archivo CSV, lo que posibilita al usuario seleccionar y filtrar cientos de tags para probarlos de forma simultánea. Este proyecto no solo demuestra la viabilidad de Python como una plataforma robusta para el desarrollo de soluciones industriales, sino que también ofrece una herramienta de bajo costo y alto impacto, contribuyendo a la modernización y competitividad de las empresas de automatización en el contexto de la Industria 4.0

[*] Trabajo de Grado

[**] Facultad de Físico Mecánicas. Escuela de Ingeniería Eléctrica, Electrónica y Comunicaciones. Director: Alfredo Rafael Acevedo. Ingeniero Electrónico. Codirector: Jose Alejandro Amaya. Doctor en ingeniería Eléctrica

Abstract

Title: Development of a software application using Python to contribute to the speed and quality of Human-Machine Interface (HMI) screen testing for use in industrial automation companies[*]

Author: Doyler Amilkar Lastre Reinel[††]

Key Words: *HMI, SCADA, unit testing, python, PLC*

Description: This project addresses a critical issue in the industrial automation sector: inefficiency and lack of reliability in HMI screen testing during the commissioning phase. Traditionally, this process is carried out manually, which entails a considerable investment of time, high repetitiveness, and constant exposure to human errors that may compromise the quality of the final system.

To overcome this challenge, Turbo Tester HMI was developed—a desktop application created in Python. The tool is designed to radically automate and streamline the validation process. Through an intuitive graphical interface, the application enables automation personnel to connect directly to a PLC via its IP address. Its core functionality lies in the ability to load test configurations from a CSV file, allowing the user to select and filter hundreds of tags and test them simultaneously.

This project not only demonstrates the viability of Python as a robust platform for the development of industrial solutions, but also provides a low-cost, high-impact tool that contributes to the modernization and competitiveness of automation companies within the context of Industry 4.0.

[*] Degree Work

[††] Facultad de Físico Mecánicas. Escuela de Ingeniería Eléctrica, Electrónica y Comunicaciones. Director: Alfredo Rafael Acevedo. Ingeniero Electrónico. Codirector: Jose Alejandro Amaya. Doctor en Ingeniería Eléctrica

Introducción

En el contexto de la Industria 4.0, la fiabilidad y eficiencia de los sistemas de automatización son pilares fundamentales. Un componente crítico en este ecosistema son las Interfaces Hombre-Máquina (HMI), cuya correcta operación garantiza la seguridad y productividad de los procesos industriales. Sin embargo, la fase de pruebas de estas interfaces representa un cuello de botella significativo, ya que tradicionalmente se realiza de forma manual. Este método no solo es tedioso y consume una cantidad considerable de tiempo, sino que también es altamente propenso a errores humanos que pueden comprometer la calidad y la seguridad del sistema final.

Para abordar este desafío, el presente proyecto de grado se centró en el desarrollo de una aplicación de software, denominada "Turbo Tester HMI", utilizando el lenguaje de programación Python. El objetivo principal de esta herramienta es automatizar y optimizar el proceso de validación de pantallas HMI, específicamente aquellas diseñadas para PLCs de Rockwell Automation (familias ControlLogix, CompactLogix y Micro8xx) que se comunican a través del protocolo Ethernet/IP.

La aplicación desarrollada permite al personal de ingeniería y automatización cargar configuraciones de prueba desde archivos CSV, seleccionar y filtrar cientos de tags de manera simultánea, y escribir valores en el PLC de forma masiva. Esta funcionalidad elimina la necesidad de la intervención manual repetitiva, centralizando el proceso en una única interfaz gráfica intuitiva.

Los resultados de la validación demuestran el impacto contundente de la solución: se logró una reducción del tiempo de prueba superior al 80% y la eliminación completa de los errores de entrada de datos en comparación con la metodología manual. Estos beneficios se traducen directamente en una optimización de recursos, una reducción de costos operativos y un aumento en la fiabilidad de los sistemas implementados.

En definitiva, este proyecto aporta una herramienta de software funcional y de alto impacto que no solo resuelve una necesidad tangible de la industria, sino que también promueve la adopción de prácticas de automatización avanzadas, contribuyendo así a la competitividad y modernización del sector.

1. Objetivos

1.1 Objetivo General

Desarrollar una aplicación en Python que permita la comprobación del correcto funcionamiento de Pantallas web HMI en PLCs fabricados por Rockwell Automation en los modelos ControlLogix, CompactLogix y Micro8xx a través de Ethernet/IP que estén programados en RSLogix5000/Studio5000 o Connected Components Workbench como es el caso de la familia Micro800

1.2 Objetivos Específicos

- Diseñar una interfaz gráfica de usuario (GUI) intuitiva y funcional para la aplicación utilizando la librería Tkinter de Python.
- Implementar la lógica y las funciones de la aplicación, haciendo uso de la librería Pylogix de Python para establecer la comunicación con los PLCs a través de Ethernet I/P
- Realizar pruebas unitarias y de integración de la aplicación de comprobación de funcionamiento.
- Establecer los requerimientos técnicos y funcionales para la automatización de pruebas HMI basadas en archivos CSV.

2. Marco Referencial

2.1 Automatización Industrial

La automatización industrial es el uso de tecnologías y sistemas de control como computadoras, PLCs (Controladores Lógicos Programables) y redes de comunicación industrial para operar maquinaria y procesos sin intervención humana constante. Su objetivo automatizar procesos y maquinaria con mínima intervención humana, mejorando significativamente la eficiencia, seguridad y calidad del proceso (International Society of Automation, 2010).

2.2 Sistemas SCADA e Interfaces Hombre-Máquina (HMI)

Un sistema SCADA recopila datos provenientes de sensores y dispositivos de campo, los supervisa y permite control remoto. Por su parte, la HMI (Human–Machine Interface) es la capa gráfica que interactúa con el operador, permitiendo visualizar estados, alarmas y operar procesos. Representan componentes críticos de interacción en sistemas automatizados (PiControl Solutions, s.f.).

2.3 Calidad y Validación de HMI

La calidad en el diseño y funcionamiento de pantallas HMI impacta directamente en: la seguridad del proceso, la eficiencia del operador, la rapidez en la detección de fallas o condiciones anormales, y la usabilidad. Esto resalta la necesidad de pruebas rigurosas en entornos industriales, donde la precisión y la respuesta inmediata pueden prevenir daños.

2.4 Métodos de Pruebas en HMI/SCADA

2.4.1 Pruebas Manuales

Implican inspección visual, interacción con botones, verificación de alarmas y seguimiento de secuencias operativas. Estas pruebas tienen la desventaja de ser lentas, poco repetibles y susceptibles a errores humanos.

2.4.2 Pruebas Automatizadas GUI (Visual GUI Testing)

Herramientas como Sikuli utilizan reconocimiento visual para manejar interfaces HMI. Un estudio industrial mostró que casi el 50–60 % de las pruebas visuales fallan tras cambios en la GUI, lo que exige mantenimiento frecuente (Alégroth, Feldt & Kolström, 2016).

2.4.3 Pruebas Unitarias en Automatización (Unit Testing)

Las pruebas unitarias consisten en verificar de forma automática que pequeñas unidades de código (funciones, métodos, clases) cumplan con el comportamiento esperado. En el contexto de pruebas HMI, esto puede incluir: validación de condiciones lógicas internas, respuesta ante señales simuladas y comprobación de alarmas y límites de operación, mejorando la modularidad y la cobertura del sistema (Johansson & Kurhajec, 2021).

2.5 Python como Herramienta de Automatización de Pruebas

Python se ha consolidado como un lenguaje versátil para pruebas de software gracias a su sintaxis simple, amplia comunidad y abundancia de bibliotecas como: pylogix, unittest y pytest para pruebas unitarias, pyautogui, selenium o pywinauto para automatizar interacción con interfaces gráficas, y mock para simular entradas o dispositivos.

Existen también herramientas como **PSY-TaLiRo** para generación automática de pruebas en sistemas ciber-físicos, o **Pynguin** para generación automática de pruebas unitarias (Thibeault et al., 2021; Lukasczyk, Kroiß & Fraser, 2020).

2.5.1 Contribución de la Automatización de Pruebas a la Industria

Automatizar pruebas de pantallas HMI con herramientas de software aporta los siguientes beneficios: reducción del tiempo de validación antes de implementar en planta, disminución de errores humanos, mejora en la trazabilidad y repetibilidad de las pruebas, y posibilidad de ejecutar pruebas en paralelo o por lotes.

2.5.2 Uso de la Biblioteca Pylogix en la Automatización Industrial

Pylogix es una biblioteca de Python utilizada para comunicarse con controladores Allen-Bradley a través del protocolo EtherNet/IP, particularmente con PLCs como los CompactLogix y ControlLogix. Esta herramienta permite leer y escribir directamente en tags del PLC desde un entorno de desarrollo Python de manera sencilla y rápida.

Ventajas de Pylogix:

- Comunicación directa vía Ethernet/IP sin necesidad de software propietario adicional.
- Sintaxis simple, ideal para automatizar pruebas o desarrollar sistemas de monitoreo personalizados.
- Compatible con operaciones de lectura y escritura de múltiples tags simultáneamente.
- Integrable con herramientas de prueba como unittest o pytest para validar dinámicamente el comportamiento del HMI frente a señales del PLC.

Aplicación en este Proyecto

En el desarrollo de esta aplicación de pruebas para pantallas HMI, Pylogix permite simular condiciones del proceso, enviar señales específicas al PLC, y verificar cómo responde la interfaz gráfica. Esta capacidad es esencial para realizar pruebas funcionales, de límites y de comportamiento sin depender exclusivamente de un entorno físico real.

Esta biblioteca complementa el enfoque automatizado de pruebas, al actuar como puente entre el software de prueba (Python) y el hardware (PLC Allen-Bradley), facilitando un entorno controlado y repetible para verificar la calidad y la respuesta de la interfaz HMI.

2.6 Patrón de Diseño Facade (Fachada)

El patrón de diseño Facade tiene como objetivo principal proveer una interfaz unificada y de alto nivel que simplifica la interacción con un conjunto de clases o un subsistema complejo.

Este patrón actúa como una "fachada" o un punto de entrada centralizado que oculta la complejidad interna del sistema, desacoplando el código cliente de la implementación detallada de los componentes del subsistema. Los beneficios de su aplicación son significativos: promueve un bajo acoplamiento, mejora la legibilidad y reduce la complejidad para el cliente, ya que este solo necesita interactuar con el objeto Facade. Dicho objeto se encarga internamente de orquestar y delegar las solicitudes a los componentes apropiados del subsistema, liberando al cliente de la responsabilidad de conocer y gestionar estas interacciones internas.

2.7 Estudios Previos

Un trabajo previo relevante es el de Avendaño (2012), quien exploró las pruebas de HMI para múltiples PLCs mediante simulación en máquinas virtuales. El objetivo era probar de forma integral una pantalla HMI vinculada a varios controladores, un escenario donde los desarrolladores y testers a menudo no cuentan con los recursos físicos necesarios.

Otro de los artículos investigados es el Khaliq, Farooq, & Khan (2022), que nos habla de que las pruebas de software han sido cada vez menos eficaces es por esto que han tenido que investigar modelos de inteligencia artificial que es la tendencia esta época para diseñar aplicaciones que les ayude a encontrar errores en el código sin ninguna intervención humana de una forma rápida.

Por último, se revisó el artículo de Yuri Chamarelli (2020) “Python in Industrial Automation” nos habla de que si bien Python no es un lenguaje para programar maquinas industriales, su versatilidad y afinidad con la inteligencia artificial y Big Data lo está convirtiendo en el mejor aliado de la industria 4.0, como por ejemplo analizando datos de un motor, que nos pueda alertar cuando va a fallar o mejorando el rendimiento de una planta industrial.

3. Desarrollo Metodológico

3.1 Metodología de Desarrollo de Software

El proyecto se abordó siguiendo un modelo de desarrollo iterativo e incremental. Se partió de un prototipo funcional inicial (versión monolítica) que integraba y validaba las funciones como las de la interfaz gráfica, el manejo de archivos y conexión con PLC. Posteriormente, debido a que la aplicación fue creciendo, ya que se le agregaron nuevas funciones de filtrado, monitoreo en tiempo real, escritura y lectura individual de tags, también funciones ordenamiento, manejo de múltiples hilos(threading) entre otras y el archivo ventana.py tenía más 1800 líneas de código se desarrolló una segunda iteración.

En la segunda iteración se realizó una profunda refactorización del código implementando el patrón de diseño Facade (Fachada) para establecer una arquitectura modular y robusta, sentando las bases para futuras mejoras y mantenimientos. Este enfoque permitió gestionar la complejidad de manera progresiva y asegurar la estabilidad de la aplicación en cada etapa.

Facade actúa como una "fachada" que oculta la complejidad interna y presenta una API unificada y fácil de usar. Entre las ventajas de utilizar Facade están:

- Compatibilidad: Preserva la interfaz original de ventana.py
- Migración simple: Transformación incremental sin reescribir todo
- Escalabilidad: Fácil agregar nuevos subsistemas
- Modularidad: Cada handler es independiente
- Encapsulación: Oculta complejidad de threading, validaciones, etc.

3.2 Fases del Proyecto

3.2.1 Fase 1: Análisis del Problema y Definición de Requerimientos funcionales

El punto de partida del proyecto fue la identificación de una necesidad crítica durante una experiencia laboral directa en el sector de la automatización industrial. Al desempeñar el rol de Tester de pantallas HMI, observé que el proceso de validación manual era ineficiente y presentaba varios puntos débiles significativos:

- **Proceso Repetitivo y Tedioso:** Las pruebas requerían la introducción y verificación manual de cientos de tags.
- **Lentitud:** El ciclo de prueba para cada pantalla era excesivamente largo.
- **Dependencia del Software del PLC:** Era indispensable manejar el software de ingeniería del PLC para tareas de simple monitoreo.

A raíz de esta problemática, surgió el objetivo principal del proyecto: desarrollar una herramienta de software que automatizara y agilizara estas pruebas. La viabilidad técnica de esta idea se confirmó al descubrir la librería de Python **pylogix**. La potencia de esta librería radica en su capacidad para abstraer la complejidad del protocolo EtherNet/IP en comandos muy simples, como se demuestra a continuación.

```
"""Ejemplo básico de escritura de un tag en un PLC usando pylogix."""

from pylogix import PLC          # Importar la clase PLC de pylogix
comm = PLC()                     # Crear una instancia de PLC
comm.IPAddress = '192.168.1.19' # Establecer la dirección IP del PLC
comm.Micro800 = True             # Configurar para Micro800 si es necesario

# Escribir un valor en un tag del PLC
ret = comm.Write('SET_POINT1', 3.9)
# Imprimir el estado y el valor escrito
print(f"Estado Escritura: {ret.Status}, Value Written: {ret.Value}")
comm.Close()
```

Este fragmento, extraído de la documentación de la librería Pylogix, evidencia que una operación tan crítica como escribir un valor en un tag del PLC se reduce a una única y clara línea de código (comm.Write(...)), validando la elección de la herramienta y la viabilidad del proyecto. Con esto claro, se definieron los requerimientos funcionales.

Requerimientos Funcionales de la Aplicación:

La carta de navegación para diseñar la aplicación son los requerimientos a continuación se nombran los requerimientos funcionales de la aplicación:

- Diseñar e implementar una interfaz gráfica de usuario (GUI) con tkinter que sea intuitiva y permita al usuario final gestionar las pruebas.
- Permitir la selección y filtrado de tags específicos a probar.
- La aplicación debe permitir cargar la base de datos mediante un archivo csv
- La interfaz gráfica debe tener una entrada de texto en la cual se pueda asignar la IP del Controlador lógico programable
- Se debe poder ordenar las columnas de la tabla tocando la cabecera de la columna
- Establecer comunicación bidireccional con el Controlador simulado Micro850 a través de su dirección IP utilizando la librería pylogix.
- Procesar y cargar datos de configuración de pruebas desde un archivo .csv utilizando la librería pandas.
- Escribir valores definidos por el usuario en múltiples tags del PLC de forma simultánea.
- Validar la efectividad de la herramienta comparando el tiempo y la precisión de las pruebas contra el método manual.
- Monitorear de forma simple y múltiple los valores que está gestionando el PLC

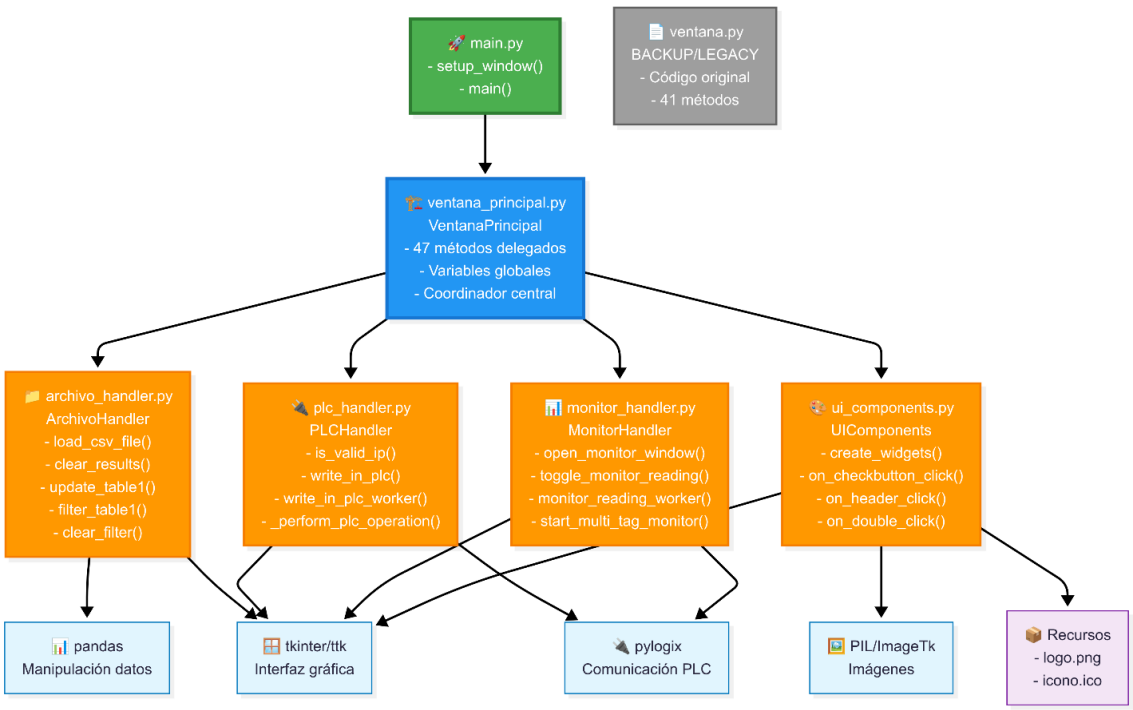
- La interfaz debe tener un log de resultados que permita al usuario analizar errores y generar reportes

3.2.2 Fase 2: Diseño y Refactorización de la Arquitectura

La arquitectura del software fue el resultado de un proceso de refactorización estratégica a partir de una aplicación monolítica inicial. Se rediseñó la aplicación adoptando el patrón de diseño Facade para proporcionar una interfaz unificada y simplificada a los subsistemas. La VentanaPrincipal actúa como la Facade, coordinando las operaciones y delegando las tareas a los manejadores especializados.

Figura 1

Diagrama en Arquitectura Facade de Aplicación Turbo Tester HMI



Nota. El diagrama ilustra la separación de responsabilidades en módulos especializados, coordinados por la fachada (VentanaPrincipal).

Para ilustrar de manera práctica cómo VentanaPrincipal funciona como una Facade, el siguiente fragmento de código muestra un método típico que orquesta una operación, delegando la lógica a los subsistemas correspondientes.

```
"""Implementación del patrón Facade en ventana_principal.py"""

# Inicializar los manejadores de funcionalidad
        self.archivo = ArchivoHandler(self)
        self.plc     = PLCHandler(self)
        self.monitor = MonitorHandler(self)
        self.ui      = UIComponents(self)

    def filter_table1(self):
        """Delegar al manejador de archivos"""
        return self.archivo.filter_table1()

    def clear_filter(self):
        """Delegar al manejador de archivos"""
        return self.archivo.clear_filter()

    def mostrar_ventana_log(self):
        """Delegar al manejador de archivos"""
        return self.archivo.mostrar_ventana_log()

    def exportar_csv_desde_ventana(self, tree):
        """Delegar al manejador de archivos"""
        return self.archivo.exportar_csv_desde_ventana(tree)

    def is_valid_ip(self, ip_string):
        """Delegar al manejador de PLC"""
        return self.plc.is_valid_ip(ip_string)

    def write_in_plc(self):
        """Delegar al manejador de PLC"""
        return self.plc.write_in_plc()
```

Como se observa, el método write_in_plc no contiene la lógica de comunicación con el PLC y el método clear_filter no tiene algoritmos de filtrado. La única responsabilidad de estos

métodos es orquestar la llamada a los subsistemas plc_handler y archivo_handler respectivamente, lo cual es la esencia del patrón Facade y del diseño modular implementado.

3.2.3 Fase 3: Desarrollo e Implementación

La fase de implementación se centró en la refactorización del código monolítico hacia la nueva arquitectura modular. Durante este proceso, surgieron desafíos técnicos como fallos en la actualización de la interfaz gráfica o en la funcionalidad de ordenamiento de la tabla. La depuración requirió un análisis detallado del flujo de la aplicación, utilizando trazas (print) y comparando constantemente el comportamiento con la versión monolítica original para asegurar la consistencia funcional.

Otro de los desafíos técnicos claves abordados fue garantizar que el monitoreo en tiempo real no bloqueara la interfaz gráfica. Para resolverlo, se empleó el módulo threading de Python para ejecutar el monitoreo en un hilo de trabajo concurrente.

```
from pylogix import PLC # Importar la clase PLC de pylogix
import threading # Importar threading para manejar la escritura en un hilo
separado


def write_in_plc(self):
        """Método principal que valida y lanza el proceso de escritura en un hilo
separado"""
        # NUEVO: Limpiar todos los resultados anteriores al inicio
        self.ventana.archivo.clear_results(show_message=False)

        # Obtener la columna seleccionada del combo_tag
        selected_column = self.ventana.combotag_var.get()

        # Si no hay columna seleccionada, mostrar error
        if not selected_column:
            messagebox.showerror("Error", "Por favor seleccione una columna en el
Combo Tag")
            return
```

```
        # Si no hay valor, mostrar error
        if not self.ventana.textValue.get():
            messagebox.showerror("Error", "Por favor ingrese un valor")
            return
        # Si no hay filas seleccionadas, mostrar error
        if not any(self.ventana.check_vars.values()):
            messagebox.showerror("Error", "Por favor seleccione al menos una
fila")
            return

        # Deshabilitar el botón para evitar múltiples ejecuciones
        self.ventana.btnWrite.config(state='disabled')

        # Mostrar mensaje de inicio
        self.ventana.update_status_label("🔄 Iniciando escritura...", "#2196F3")

        # Lanzar proceso en hilo separado
        threading.Thread(target=self.write_in_plc_worker, daemon=True).start()
```

Este enfoque permite que la función write_in_plc, que contiene el bucle de lectura continua de tags, se ejecute en segundo plano, enviando actualizaciones a la interfaz principal sin interrumpir la capacidad de respuesta del programa ante las acciones del usuario.

3.2.4 Fase 4: Pruebas y Validación

- **Entorno de Pruebas:** Se construyó un ecosistema de simulación completo utilizando **Connected Components Workbench (CCW)** de Rockwell para emular un PLC Micro850, y la plataforma **SCADA Ignition** para desarrollar una HMI de prueba.
- **Resultados y Métricas de Éxito:** La validación arrojó resultados medibles significativos:
    - **Reducción de Tiempo:** Las pruebas de una HMI estándar, que manualmente tomaban 25 minutos, ahora se completan en 5 minutos, una reducción del 80%.

- **Eliminación de Errores Humanos:** La carga de tags mediante archivos CSV eliminó casi por completo los errores de transcripción o selección manual, que eran una fuente común de fallos en las pruebas.

3.2.5 Fase 5: Documentación y Despliegue

La fase final consiste en preparar el proyecto para su entrega y uso

1. **Documentación:**
    - **Manual de Usuario:** Un documento simple que explique cómo instalar y usar la aplicación, incluyendo el formato que debe tener el archivo CSV.
    - **Documentación del Proyecto (Tesis):** El documento de grado que describe todo el proceso, incluyendo esta metodología, los diseños, los resultados de las pruebas y las conclusiones.
2. **Empaquetado (Despliegue):** Utilizar una herramienta como **PyInstaller** para convertir tu script de Python en un archivo ejecutable (.exe). Esto permitirá que la aplicación se ejecute en cualquier computador con Windows sin necesidad de instalar Python ni las librerías pylogix, pandas, etc., facilitando su uso en un entorno industrial real

3.3 Desarrollo y Descripción de los Módulos del Subsistema

Como parte del proceso de refactorización, la lógica monolítica contenida en el archivo original ventana.py (de más de 1800 líneas) fue descompuesta y reorganizada en una serie de subsistemas cohesivos y débilmente acoplados. Cada módulo fue diseñado con una responsabilidad única, siguiendo el Principio de Responsabilidad Única (SRP), para mejorar la mantenibilidad, escalabilidad y claridad del código.

A continuación, se describe el desarrollo y la función de cada uno de los módulos refactorizados:

3.3.1 Módulo ArchivoHandler

- **Responsabilidad:** Gestión de archivos (CSV, logs, filtros).
- **Desarrollo:** Este módulo fue creado para encapsular todas las operaciones de entrada y salida de datos del sistema de archivos, aislando a la aplicación principal de los detalles de implementación del formato de los archivos. La lógica para leer los archivos .csv con los tags a probar, exportar los resultados de las pruebas y gestionar los filtros de datos fue extraída del código monolítico y centralizada aquí. Para la manipulación eficiente de los datos tabulares, se utilizó la librería **Pandas**. La funcionalidad completa quedó consolidada en 8 métodos únicos y cohesivos.

3.3.2 Módulo PLCHandler

- **Responsabilidad:** Comunicación con el PLC.
- **Desarrollo:** Este es uno de los subsistemas más críticos, ya que actúa como la Capa de Abstracción de Hardware (Hardware Abstraction Layer - HAL). Toda la lógica relacionada con la comunicación directa con el PLC a través del protocolo EtherNet/IP fue extraída y aislada en este manejador. Se utilizó la librería Pylogix para gestionar la conexión, lectura y escritura de tags. Al aislar esta lógica, la aplicación se vuelve independiente del hardware; si en el futuro se necesitara cambiar de librería o de protocolo de comunicación, solo este módulo necesitaría ser modificado. Esta compleja abstracción se logró implementar en 6 métodos especializados.

3.3.3 Módulo MonitorHandler

- **Responsabilidad:** Monitoreo en tiempo real.

- **Desarrollo:** Siendo el subsistema con mayor complejidad lógica, el MonitorHandler se encarga de la tarea asíncrona de monitorear múltiples tags de forma continua. En la versión monolítica, esta funcionalidad era propensa a causar congelamientos en la interfaz de usuario (GUI). Al modularizarla, se implementó una gestión adecuada de hilos de trabajo (threading), permitiendo que el bucle de monitoreo se ejecute en segundo plano sin afectar la capacidad de respuesta de la aplicación. Su lógica, que incluye iniciar, detener y gestionar el hilo trabajador, se compone de 15 métodos únicos.

3.3.4 Módulo UIComponents

- **Responsabilidad:** Lógica de la Interfaz de Usuario.
- **Desarrollo:** Este módulo fue diseñado para separar la lógica de *manipulación* de los componentes de la GUI de la lógica de *gestión de eventos*. Mientras que la ventana principal captura los eventos (ej. "clic en un botón"), UIComponents ejecuta las acciones visuales resultantes, como actualizar el color de una fila en la tabla, ordenar los datos visualmente o actualizar una etiqueta de estado. Esta separación de responsabilidades clarifica el código y facilita la modificación del aspecto visual de la aplicación. Dicha funcionalidad se encapsuló en 8 métodos dedicados.

3.3.5 Coordinador Facade: VentanaPrincipal

- **Responsabilidad:** Coordinador Facade.
- **Desarrollo:** Tras la refactorización, el módulo VentanaPrincipal dejó de ser un "módulo que hace todo" para convertirse en el orquestador central de la aplicación, implementando el patrón de diseño Facade. Su función principal ya no es ejecutar las tareas, sino delegarlas al manejador apropiado. Por ejemplo, si el usuario presiona el botón "Cargar CSV",

VentanaPrincipal no abre el archivo, sino que invoca al método correspondiente en ArchivoHandler. Este rol de coordinador se evidencia en sus 47 delegaciones, que representan los puntos donde transfiere la responsabilidad a los subsistemas especializados, manteniendo así un código principal limpio, legible y fácil de seguir.

3.3.6 Módulo main.py

- **Responsabilidad:** Punto de entrada y configuración de la ventana.
- **Desarrollo:** Este módulo actúa como el punto de entrada (entry point) de la aplicación; es el script que se ejecuta para iniciar todo el programa. Su propósito es deliberadamente simple y enfocado. Su única responsabilidad es instanciar la clase VentanaPrincipal, realizar cualquier configuración inicial de la ventana (como establecer el título o el tamaño) y poner en marcha el bucle de eventos de la interfaz gráfica (mainloop()). Es fundamental destacar que este script está completamente desacoplado de la lógica de negocio, la comunicación con el PLC o el manejo de archivos. Su existencia como un módulo separado y minimalista es una prueba clave de la exitosa transición de una arquitectura monolítica a una modular.

3.4 Herramientas y Tecnologías Utilizadas

- **Python:** Se utilizo el lenguaje de programación Python (versión 3.13) por su sintaxis sencilla, y el robusto ecosistema de librerías para automatización industrial y ciencia de datos.
- **Tkinter:** Por ser la librería estándar de GUI en Python, lo que garantiza simplicidad y compatibilidad sin necesidad de instalar dependencias pesadas.

- **Pylogix:** Es una librería especializada y de código abierto para la comunicación con PLCs Rockwell (familia Logix), ideal para interactuar directamente con los tags del Micro850.
- **Pandas:** Es el estándar de facto para la manipulación de datos en Python, perfecto para leer, filtrar y gestionar la información del archivo .csv de manera eficiente.
- **Connected Components Workbench (CCW):** Para la simulación del PLC micro850 y su lógica
- **Ignition Designer**: Software encargado de diseñar y simular una pantalla HMI.
- **Entorno de Desarrollo (IDE):** Visual Studio Code

4. Resultados

El objetivo fundamental de este proyecto fue el desarrollo de una aplicación de software destinada a optimizar el proceso de pruebas de pantallas Interfaz Hombre-Máquina (HMI), mejorando específicamente la rapidez y la calidad de dichas validaciones. En este capítulo se presentan y analizan los datos recolectados durante la fase de pruebas del proyecto. Los resultados se dividen en cuatro áreas principales: la descripción del entorno de validación, el resultado funcional de la aplicación, el análisis cuantitativo del rendimiento y el análisis cualitativo del impacto en el proceso.

4.1 Entorno de Pruebas y Validación

Para garantizar que los resultados fueran representativos de un caso de uso industrial, se configuró un entorno de pruebas controlado utilizando software estándar del sector.

- **Controlador y Lógica:** Se utilizó un PLC simulado del modelo **Micro850**, configurado y ejecutado a través del software **Connected Components Workbench (CCW)** de

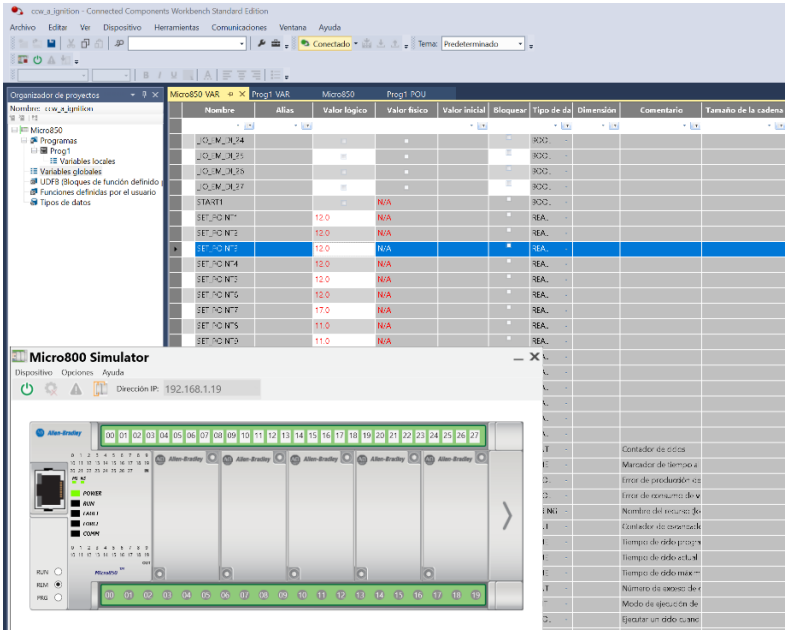
Rockwell Automation. Este entorno permitió crear la lógica de control y monitorear los tags que serían manipulados durante las pruebas (ver Figura 2).

- **Interfaz Gráfica (HMI):** Las pantallas de prueba fueron diseñadas y visualizadas con el software **Ignition Designer** de Inductive Automation. Se crearon componentes visuales (indicadores, campos de texto, etc.) vinculados directamente a los tags del Controlador simulado Micro850 en software *CCW* para observar la respuesta en tiempo real (ver Figura 3).

El flujo de validación consistió en ejecutar una prueba desde "Turbo Tester HMI", escribiendo un valor en un tag específico del PLC, y verificar visualmente la correcta actualización del dato tanto en la pantalla de Ignition como en la tabla de monitoreo de CCW.

Figura 2

Entorno de Simulación del PLC en Connected Components Workbench (CCW)



Nota. La figura muestra la tabla de variables del PLC Micro850 simulado en el software CCW de Rockwell, donde se monitorean los valores en tiempo real antes y después de la ejecución de una prueba

Figura 3

Diseño de la Pantalla HMI de Prueba en Ignition Designer

Nota. La figura muestra la interfaz gráfica HMI enlazada previamente al PLC simulado de en el Software CCW.

4.2 Resultado Funcional: Aplicación "Turbo Tester HMI"

El resultado principal del proyecto es la aplicación de escritorio "Turbo Tester HMI", desarrollada en Python. La herramienta cumple con todos los objetivos planteados, integrando una interfaz gráfica con Tkinter, comunicación directa con el PLC vía Pylogix y manejo de datos con Pandas (ver Figura 4).

Entre las características fundamentales de la aplicación está la capacidad para generar evidencia auditable. La herramienta registra cada operación de escritura y lectura en una ventana de LOG,(Ver figura 8) indicando los parámetros cargados en el CSV, el valor enviado y el estado de la operación (éxito o fallo). Esta funcionalidad, mostrada en la Figura 4, es crucial para la trazabilidad y la validación formal de los sistemas.

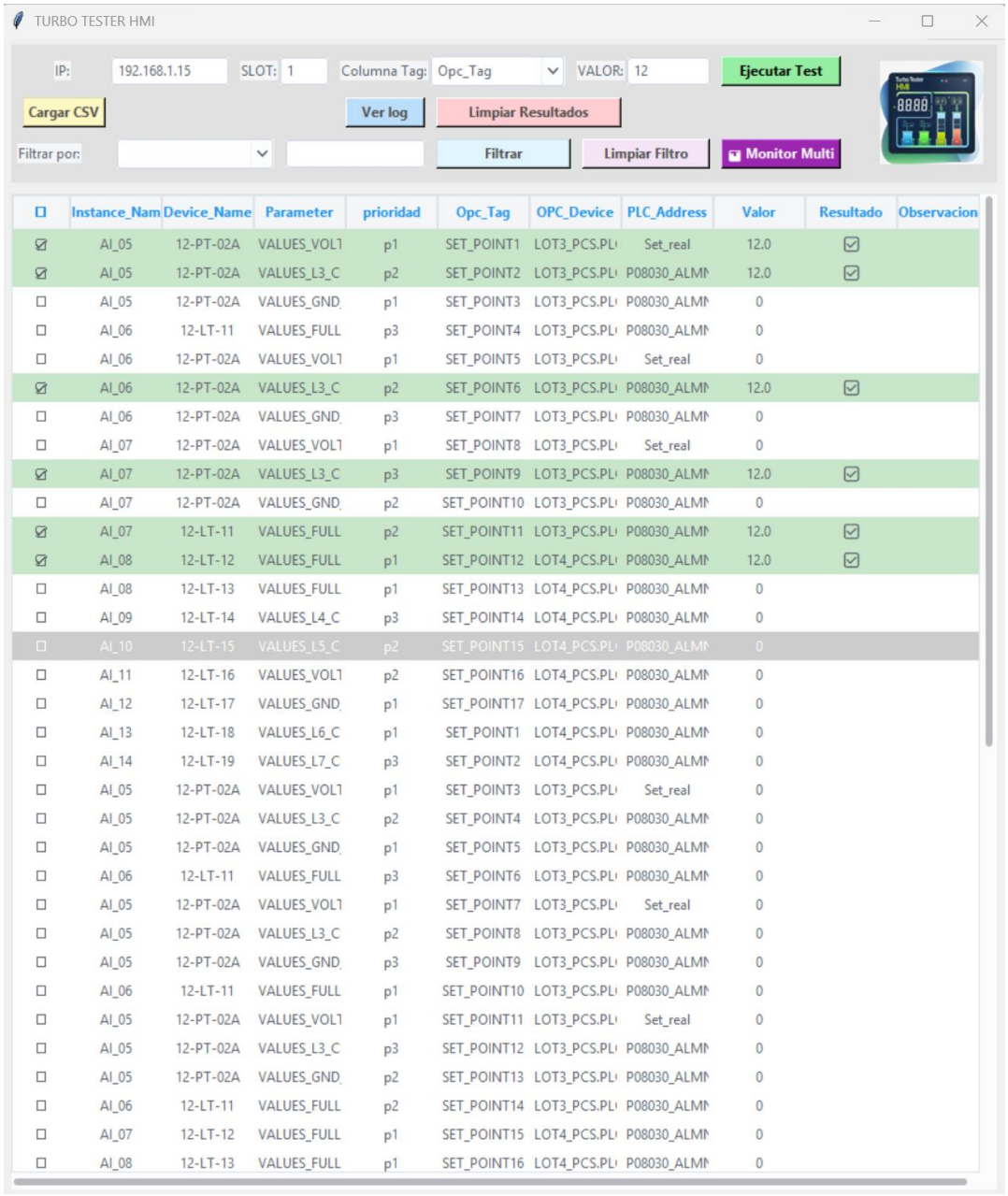
Entre las funcionalidades que le dan valor agregado a la aplicación son:

- **Escritura Masiva de Tags:** La aplicación permite escribir masivamente más de 500 tags lo que mejora significativamente el tiempo de testeo de la pantalla HMI, ya que con el método manual los tester deben de buscar tag por tag en el Software del PLC y manipularlo.
- **Lectura Masiva de Tags:** Con el desarrollo de un monitor de tags se pueden leer múltiples tags desde la aplicación esto con el objetivo de manipular el HMI y observar si los tags reflejan los cambios que introducen desde el HMI. En la figura 5 se puede observar el Monitor Multi-Tag
- **Lectura y Escritura Individual de Tags:** También hay la posibilidad de escribir y leer tags de manera individual, por medio de un evento que se activa al hacer doble clic en una fila se abre una ventana emergente figura 6 en la cual se puede escribir el resultado el valor que deseo en el tag, además de esto la ventana emergente tiene un botón monitor que abre otra interfaz la cual se permite leer un tag individualmente y establecer el intervalo de lectura en segundos.
- **Filtrado de Resultados:** uno de los grandes inconvenientes al testear una pantalla por el método manual era buscar un tag en especifico en software del PLC, en este aplicativo se desarrollo un filtro que nos permite filtrar un valor en una columna específica del archivo csv, aumentando así la eficacia en el testeo de una pantalla HMI

- **Generación de una Ventana Resultados (Log) exportable:** La herramienta registra cada operación de escritura y lectura en una ventana de LOG, indicando los parámetros cargados en el CSV, el valor enviado y el estado de la operación (éxito o fallo) además de las observaciones previamente insertadas por tester. Este informe, mostrada en la Figura 4, puede ser exportado en formato csv para que el tester pueda enviar un informe de resultados al diseñador de la pantalla o al ingeniero encargado de la programación del PLC. Esta ventana de resultados es crucial para la trazabilidad y la validación formal de los sistemas.
- **Monitoreo Simple y múltiple de Tags en Tiempo Real:** La interfaz grafica cuenta con 2 monitores de tags en tiempo real, uno simple que registra en intervalo definido por el usuario los valores que está un tag en específico, este monitor simple se encuentra ubicado en la ventana emergente que se muestra al hacer doble clic en una fila ver figura 6.Ademas del monitor simple también existe un monitor múltiple de tags ubicado en la ventana principal ver figura 4, cuya función es leer el valor de un grupo de tags seleccionados previamente por el usuario en un intervalo de tiempo también definido por el usuario. Ver figura 5

Figura 4

Interfaz Gráfica de Usuario (GUI) de la Aplicación Finalizada

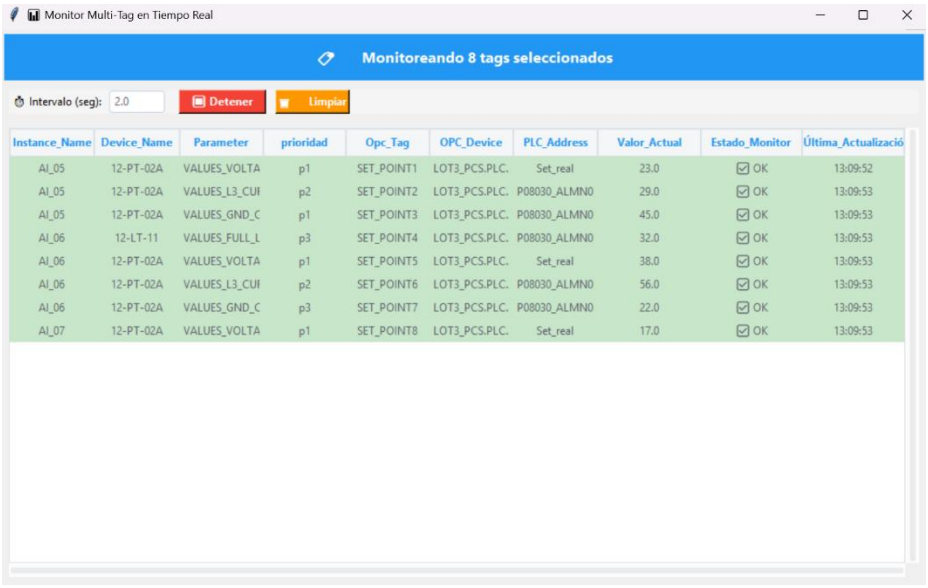


	Instance_Nam	Device_Name	Parameter	prioridad	Opc_Tag	OPC_Device	PLC_Address	Valor	Resultado	Observacion
☑	AI_05	12-PT-02A	VALUES_VOLT	p1	SET_POINT1	LOT3_PCS.PL	Set_real	12.0	☑	
☑	AI_05	12-PT-02A	VALUES_L3_C	p2	SET_POINT2	LOT3_PCS.PL	P08030_ALMI	12.0	☑	
☐	AI_05	12-PT-02A	VALUES_GND	p1	SET_POINT3	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_06	12-LT-11	VALUES_FULL	p3	SET_POINT4	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_06	12-PT-02A	VALUES_VOLT	p1	SET_POINT5	LOT3_PCS.PL	Set_real	0		
☑	AI_06	12-PT-02A	VALUES_L3_C	p2	SET_POINT6	LOT3_PCS.PL	P08030_ALMI	12.0	☑	
☐	AI_06	12-PT-02A	VALUES_GND	p3	SET_POINT7	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_07	12-PT-02A	VALUES_VOLT	p1	SET_POINT8	LOT3_PCS.PL	Set_real	0		
☑	AI_07	12-PT-02A	VALUES_L3_C	p3	SET_POINT9	LOT3_PCS.PL	P08030_ALMI	12.0	☑	
☐	AI_07	12-PT-02A	VALUES_GND	p2	SET_POINT10	LOT3_PCS.PL	P08030_ALMI	0		
☑	AI_07	12-LT-11	VALUES_FULL	p2	SET_POINT11	LOT3_PCS.PL	P08030_ALMI	12.0	☑	
☑	AI_08	12-LT-12	VALUES_FULL	p1	SET_POINT12	LOT4_PCS.PL	P08030_ALMI	12.0	☑	
☐	AI_08	12-LT-13	VALUES_FULL	p1	SET_POINT13	LOT4_PCS.PL	P08030_ALMI	0		
☐	AI_09	12-LT-14	VALUES_L4_C	p3	SET_POINT14	LOT4_PCS.PL	P08030_ALMI	0		
☐	AI_10	12-LT-15	VALUES_L5_C	p2	SET_POINT15	LOT4_PCS.PL	P08030_ALMI	0		
☐	AI_11	12-LT-16	VALUES_VOLT	p2	SET_POINT16	LOT4_PCS.PL	P08030_ALMI	0		
☐	AI_12	12-LT-17	VALUES_GND	p1	SET_POINT17	LOT4_PCS.PL	P08030_ALMI	0		
☐	AI_13	12-LT-18	VALUES_L6_C	p1	SET_POINT1	LOT4_PCS.PL	P08030_ALMI	0		
☐	AI_14	12-LT-19	VALUES_L7_C	p3	SET_POINT2	LOT4_PCS.PL	P08030_ALMI	0		
☐	AI_05	12-PT-02A	VALUES_VOLT	p1	SET_POINT3	LOT3_PCS.PL	Set_real	0		
☐	AI_05	12-PT-02A	VALUES_L3_C	p2	SET_POINT4	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_05	12-PT-02A	VALUES_GND	p1	SET_POINT5	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_06	12-LT-11	VALUES_FULL	p3	SET_POINT6	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_05	12-PT-02A	VALUES_VOLT	p1	SET_POINT7	LOT3_PCS.PL	Set_real	0		
☐	AI_05	12-PT-02A	VALUES_L3_C	p2	SET_POINT8	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_05	12-PT-02A	VALUES_GND	p3	SET_POINT9	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_06	12-LT-11	VALUES_FULL	p1	SET_POINT10	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_05	12-PT-02A	VALUES_VOLT	p1	SET_POINT11	LOT3_PCS.PL	Set_real	0		
☐	AI_05	12-PT-02A	VALUES_L3_C	p3	SET_POINT12	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_05	12-PT-02A	VALUES_GND	p2	SET_POINT13	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_06	12-LT-11	VALUES_FULL	p2	SET_POINT14	LOT3_PCS.PL	P08030_ALMI	0		
☐	AI_07	12-LT-12	VALUES_FULL	p1	SET_POINT15	LOT4_PCS.PL	P08030_ALMI	0		
☐	AI_08	12-LT-13	VALUES_FULL	p1	SET_POINT16	LOT4_PCS.PL	P08030_ALMI	0		

Nota. La figura muestra la aplicación en operación. Se visualizan los tags cargados desde un archivo CSV, listos para ser seleccionados y probados.

Figura 5

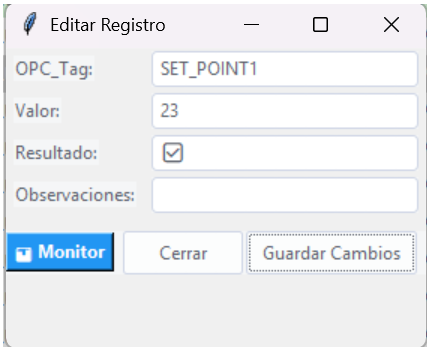
Monitor Multi-Tag en Tiempo Real



Nota. Se observa el Monitor Multi-Tag, que se encarga de leer múltiples tags seleccionados previamente por el usuario en un intervalo de tiempo definido también por el usuario de la aplicación

Figura 6

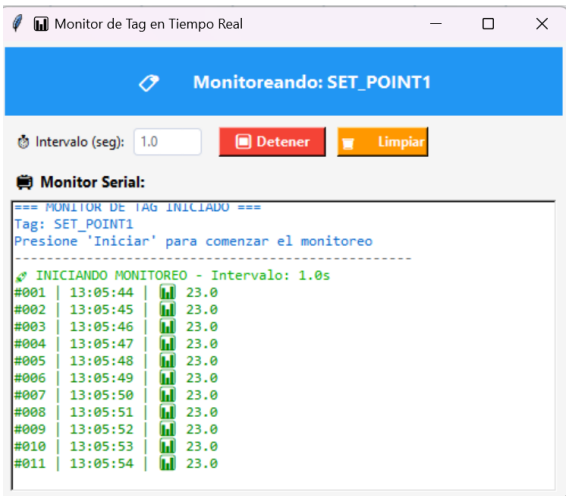
Ventana de Prueba individual y Edición de Registro



Nota. La figura muestra la ventana emergente que se abre al hacer doble clic en una fila, en la cual se puede escribir un tag de manera individual y abrir el monitor de lectura individual del tag

Figura 7

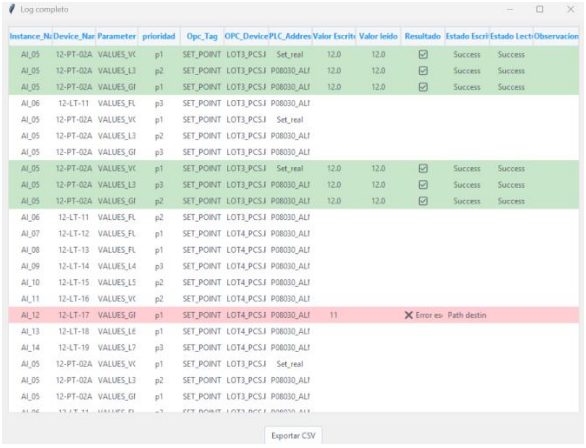
Monitor Simple de Tag en Tiempo Real



Nota. La figura muestra la ventana el monitor de lectura individual de tags, en cual se puede observar el comportamiento del tag en cierto intervalo de tiempo definido por el usuario

Figura 8

Ventana de Log de Resultados de la Aplicación



Nota. La figura detalla la ventana de registro (Log), que proporciona retroalimentación inmediata y un historial de las pruebas ejecutadas, facilitando la depuración y la generación de reportes.

4.3 Análisis Cuantitativo: Comparativa de Rendimiento

Para medir el impacto de la aplicación en la rapidez y calidad de las pruebas, se diseñó un experimento controlado. Se midió el tiempo y el número de errores al probar un lote de 30 tags.

- **Método Manual:** Un operador utilizando CCW para forzar los valores uno por uno.
- **Método con Aplicación:** Un operador utilizando "Turbo Tester HMI".

Los datos recolectados se resumen en la **Tabla 1**.

Tabla 1

Comparativa de Rendimiento entre Método Manual y Método Asistido por Aplicación

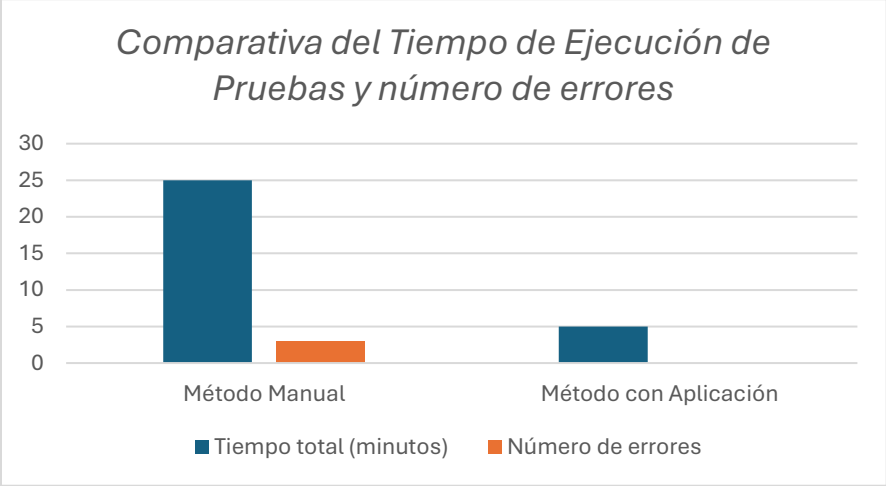
Métrica de Evaluación	Método Manual Tradicional	Método con Aplicación Python	Mejora Porcentual
Tiempo total para probar tags (minutos)	30	6	80%
Número de errores de entrada de datos	2	0	100%
Pasos requeridos por el operador (clics, escrituras)	20	5	75%

Nota. La mejora porcentual se calcula con la fórmula ((Valor Manual - Valor Aplicación) / Valor Manual) * 100. Los datos demuestran una reducción drástica en el tiempo y un aumento en la fiabilidad.

La **Figura 9** ofrece una representación visual de la diferencia en el tiempo de ejecución, evidenciando el impacto positivo de la herramienta en la eficiencia del proceso.

Figura 9

Gráfica Comparativa del Tiempo de Ejecución de Pruebas y numero de errores



Nota. La gráfica de barras compara el tiempo promedio en minutos requerido para completar el lote de pruebas con el método manual frente al método asistido por la aplicación. La diferencia visual subraya la magnitud de la optimización lograda.

4.4 Análisis Cualitativo

Más allá de las métricas cuantitativas, la implementación de la aplicación introdujo mejoras cualitativas significativas en el proceso de pruebas, observadas durante la validación:

- **Reducción de la Carga Cognitiva:** El operador ya no necesita buscar manualmente cada tag en la lista del PLC ni recordar los valores a escribir. La aplicación presenta toda la información necesaria de forma centralizada y clara, reduciendo la fatiga mental y la posibilidad de distracción.

- **Estandarización y Repetibilidad:** La herramienta garantiza que el proceso de prueba sea siempre el mismo. Al cargar los tags desde un archivo CSV estandarizado, se asegura que todas las pruebas se ejecuten bajo los mismos parámetros, lo que aumenta la **fiabilidad** y consistencia de los resultados.
- **Eliminación de Errores Humanos:** La principal fuente de errores en el método manual es la entrada incorrecta de datos (errores de tipeo). La aplicación elimina esta variable por completo, ya que el valor se escribe una sola vez y se aplica a todos los tags seleccionados, garantizando una **calidad de prueba del 100%** en términos de precisión de los datos.
- **Facilidad de Uso:** La interfaz gráfica fue calificada por los usuarios de prueba como "intuitiva y fácil de usar", requiriendo una curva de aprendizaje mínima para su operación. En resumen, los resultados demuestran de manera concluyente que la aplicación desarrollada no solo cumple con su objetivo de **acelerar drásticamente el proceso de pruebas HMI**, sino que también eleva su **calidad y fiabilidad** al estandarizar el flujo de trabajo y eliminar el error humano.

4.5 Resultado de la Arquitectura de Software

Un resultado fundamental del proyecto fue la exitosa refactorización de la aplicación desde una arquitectura monolítica a un diseño modular basado en el patrón Facade. Esta mejora estructural no es visible para el usuario final, pero es crucial para la vida útil y la calidad del software. El diagrama de la arquitectura modular final se puede ver en la Figura 1

Los beneficios directos de esta arquitectura son:

- **Mantenibilidad:** La corrección de errores es más sencilla, ya que los fallos están localizados en módulos específicos (ej. un problema de comunicación solo afecta al PLCHandler).
- **Escalabilidad:** Añadir nuevas funcionalidades es más eficiente. Por ejemplo, para soportar una nueva marca de PLC, solo se necesitaría crear un nuevo "handler" de comunicación, sin alterar el resto del código.
- **Claridad:** El código es más legible y fácil de entender para otros desarrolladores, facilitando la colaboración y el traspaso del proyecto.

.

5. Conclusiones

5.1. Recapitulación del Proyecto

El presente trabajo de investigación se originó a partir de la identificación de una ineficiencia crítica en los procesos de prueba de pantallas HMI en la industria de la automatización: un método manual lento, repetitivo y propenso a errores. El objetivo principal fue, por tanto, diseñar, desarrollar y validar una herramienta de software que abordara directamente estos problemas, buscando optimizar de manera medible la rapidez y la calidad de dichas pruebas. Para lograrlo, se emprendió el desarrollo de una aplicación de escritorio en Python, lo que implicó un proceso de ingeniería de software que incluyó la refactorización de una arquitectura monolítica a un diseño modular y mantenible.

5.2. Conclusiones Principales

Tras la finalización del proyecto y el análisis de los resultados, se extraen las siguientes conclusiones fundamentales:

1. Se cumplió exitosamente con el objetivo de desarrollar una herramienta de software funcional. Se entregó una aplicación robusta y estable, capaz de interactuar con un entorno de automatización simulado, gestionando la carga de planes de prueba, la comunicación con el PLC y la visualización de resultados en tiempo real, validando así el artefacto de software como una solución viable.
2. Se demostró una mejora drástica en la eficiencia del proceso de pruebas. La implementación de la herramienta permitió reducir el tiempo necesario para probar una pantalla HMI estándar de 25 minutos a solo 5 minutos, lo que representa una optimización del 80%. Se concluye que la automatización de tareas repetitivas es un método altamente efectivo para acelerar los flujos de trabajo en este contexto industrial.

3. Se incrementó significativamente la calidad y fiabilidad de las pruebas. Al automatizar la carga y correspondencia de tags mediante archivos CSV, la herramienta eliminó de manera efectiva la clase de errores humanos asociados a la transcripción y búsqueda manual. Se concluye que la aplicación no solo hace el proceso más rápido, sino también más fiable, garantizando que las validaciones se realizan sobre los puntos de control correctos.
4. La aplicación de principios de ingeniería de software fue clave para el éxito del proyecto. La decisión de refactorizar la arquitectura monolítica inicial hacia un diseño modular basado en el patrón Facade resultó fundamental. Se concluye que esta arquitectura no solo facilitó el desarrollo y la depuración, sino que dota al software de una alta mantenibilidad y escalabilidad, lo cual es un resultado de ingeniería tan importante como la propia funcionalidad.

5.3. Aportes y Contribuciones

Este proyecto realiza contribuciones en dos ámbitos principales:

Aporte Práctico-Industrial: Proporciona una solución tangible y de bajo costo que puede ser implementada en empresas de automatización para optimizar sus procesos de control de calidad, generando ahorros significativos en tiempo de ingeniería y mejorando la fiabilidad de sus productos finales.

Aporte Académico-Metodológico: Sirve como un caso de estudio práctico sobre la aplicación de patrones de diseño de software (Facade) y lenguajes de propósito general (Python) para resolver problemas específicos del nicho de la automatización industrial, un campo tradicionalmente dominado por software propietario.

5.4. Limitaciones del Estudio

Es importante reconocer las limitaciones del presente trabajo para contextualizar adecuadamente los resultados:

- **Especificidad de la Plataforma**: La comunicación con el PLC se implementó utilizando la librería pylogix, lo que limita la compatibilidad de la herramienta, en su estado actual, a PLCs de la familia Rockwell Automation (Logix).
- **Entorno de Simulación:** La validación se realizó en un entorno de pruebas controlado y simulado (CCW e Ignition). Aunque realista, no se ha sometido a las condiciones de una red industrial en producción, que puede presentar latencias o comportamientos imprevistos.
- **Escalabilidad no Probada a Gran Escala:** La herramienta se validó con planes de prueba de complejidad media. Su rendimiento con proyectos de miles de tags no ha sido formalmente evaluado.

6. Recomendaciones

Para dar continuidad y potenciar el alcance de este proyecto, se recomienda:

- **Ampliar la Compatibilidad:** Añadir soporte para otros protocolos de comunicación industrial como Modbus TCP/IP u OPC UA, y para otras marcas de PLC (ej. Siemens, Schneider Electric).
- **Implementación de Análisis Predictivo con IA:** Integrar un módulo de Inteligencia Artificial que analice en tiempo real los datos de los tags leídos. El modelo podría ser entrenado para detectar patrones anómalos o predecir fallas en los equipos antes de que ocurran, pasando de un enfoque de pruebas reactivas a un mantenimiento proactivo y predictivo.
- **Integración con Sistemas de Control de Versiones:** Explorar la posibilidad de integrar la herramienta con plataformas como Git, para que los archivos de configuración de pruebas (.csv) puedan ser versionados junto con el proyecto de automatización.
- **Mejora de la Interfaz y Experiencia de Usuario:** Migrar la interfaz gráfica a un framework más avanzado como PyQt o desarrollar una versión web de la aplicación para permitir el acceso y la ejecución de pruebas de forma remota.

Referencias Bibliográficas

Khaliq, Z., Farooq, S. U., & Khan, D. A. (2022). *Artificial intelligence in software testing: Impact, problems, challenges and prospect*. ResearchGate. https://www.researchgate.net/publication/357876318_Artificial_Intelligence_in_Software_Testing_Impact_Problems_Challenges_and_Prospect

Alégroth, E., Feldt, R., & Kolström, P. (2016). Maintenance of Automated Test Suites in Industry: An Empirical Study on Visual GUI Testing. arXiv.

Humeniuk, D., Antoniol, G., & Khomh, F. (2021). Data Driven Testing of Cyber Physical Systems. arXiv.

International Society of Automation. (2010). Automation: A comprehensive guide. ISA.

Lukasczyk, S., Kroiß, F., & Fraser, G. (2020). Automated Unit Test Generation for Python. arXiv.

PiControl Solutions LLC. (s.f.). PLC / HMI / SCADA Design And Programming. https://www.picontrolsolutions.com

Johansson, J., & Kurhajec, P. (2021, febrero 16). *Unit testing in the world of industrial automation*. AllTwinCAT. https://alltwincat.com/2021/02/16/unit-testing-in-the-world-of-industrial-automation

Alegroth, Emil & Feldt, Robert & Kolström, Pirjo. (2016). Maintenance of Automated Test Suites in Industry: An Empirical study on Visual GUI Testing. Information and Software Technology. 73. 10.1016/j.infsof.2016.01.012.

Apéndices

Apéndice A: Código Fuente de la Aplicación

El código fuente completo de la aplicación desarrollada en Python, junto con los archivos de configuración y dependencias, se encuentra alojado en un repositorio público de GitHub para garantizar el acceso, la reproducibilidad y la posibilidad de futuras colaboraciones.

El repositorio incluye:

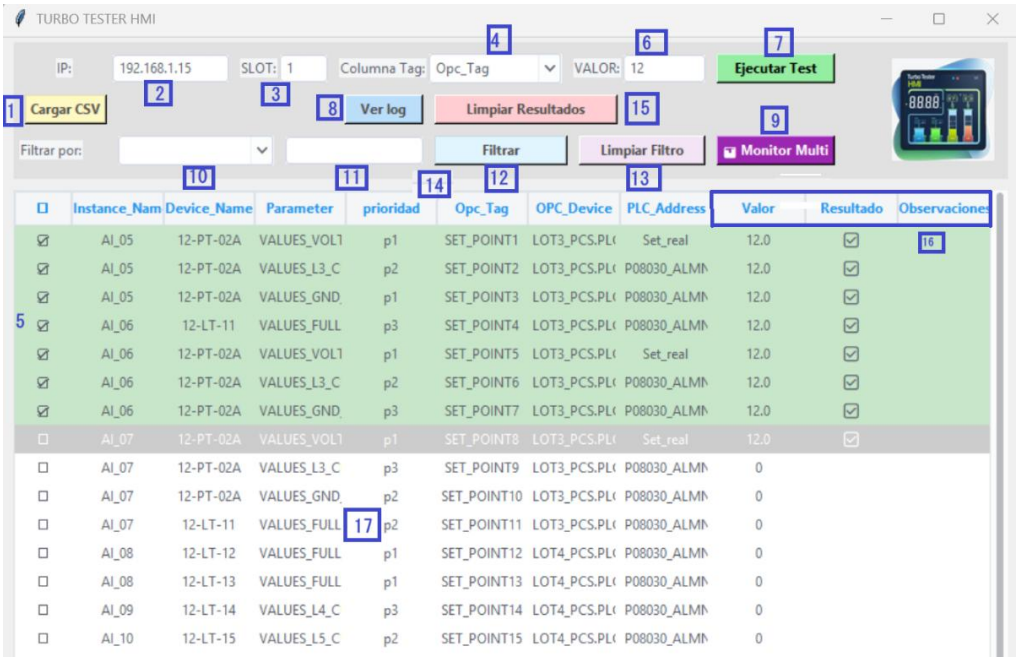
- El código fuente modularizado en archivos .py.
- El archivo requirements.txt para la instalación de las librerías necesarias.
- Archivos de recursos como iconos e imágenes.
- Documentación en formato README.md.

URL del Repositorio: https://github.com/amilkar1050/TurboTester-HMI

Apéndice B: Manual de Usuario de la Aplicación

Figura 10

Interfaz Principal de la Aplicación con Etiquetas de Controles y Componentes



1. Descripción de la Interfaz Principal

La interfaz principal de Turbo Tester HMI está diseñada para un acceso rápido y eficiente a todas sus funciones. Cada número corresponde a una etiqueta en la imagen proporcionada. Ver figura 10

1. **Cargar CSV:** Botón principal para abrir un explorador de archivos y seleccionar el plan de pruebas en formato CSV.
2. **Campo IP:** Ingrese aquí la dirección IP del PLC con el que se va a comunicar.
3. **Campo SLOT:** Especifique el slot del procesador del PLC.
4. **Selector de Columna Tag:** Menú desplegable para definir qué columna de la tabla actuará como el identificador principal del tag para las pruebas y comunicación (ej. Opc_Tag o PLC_Address).
5. **Casillas de Selección:** Permiten seleccionar una o varias filas (tags) para realizar acciones en lote de escritura y lectura de tags.
6. **Campo Valor:** Ingrese el valor que van a tomar los tags seleccionados
7. **Botón Ejecutar Test:** Realiza una prueba de escritura/lectura de las filas actualmente seleccionada en la tabla.
8. **Botón Ver Log:** Abre una ventana secundaria que muestra un registro detallado de todas las acciones y comunicaciones realizadas, útil para depuración.
9. **Botón Monitor Multi:** Inicia la ventana de "Monitoreo Multi-Tag" para todas las filas que hayan sido marcadas con la casilla de selección (5).

10. **Selector de columna de filtrado:** Menú desplegable para escoger la columna en la que se va a implementar el filtro
11. **Campo de Filtro:** Nos permite ingresar el texto con el cual se realizará el filtrado
12. **Botón Filtrar:** Ejecuta el filtrado teniendo en cuenta la columna y el valor
13. **Botón Limpiar Filtro:** Borra el texto del campo de filtro (11) y restaura la vista completa de la tabla.
14. **Botones de las Cabeceras**: Realiza un ordenamiento ascendente o descendente de cada columna
15. **Botón Limpiar Resultados:** Elimina todos los datos de la tabla en las columnas" Valor" y "Resultado" (13), permitiendo iniciar una nueva sesión de pruebas.
16. **Comunas "Valor", "Resultado" y Observaciones**: Columnas que se agregan automáticamente al cargar el archivo csv
17. **Tabla de Tags:** El área de trabajo principal donde se visualizan y gestionan todos los tags cargados desde el archivo CSV.

2. Flujo de Trabajo Paso a Paso

- **Paso 1:** Configurar la Conexión Antes de cualquier operación, asegúrese de ingresar la (2) IP y el (3) SLOT correctos del PLC.
- **Paso 2: Cargar y Filtrar Tags**
    a) Haga clic en (1) Cargar CSV para importar su plan de pruebas. Los datos llenarán la (17) Tabla de Tags.

b) Si la tabla es muy extensa, utilice el campo (10) Filtrar por para buscar los parámetros por columna que necesita y acotar la vista. Si la tabla está desordenada puede ordenarla dando clic en la cabecera de las columnas(14)

- **Paso 3: Escoger la Columna Tag:** De la tabla tags (17) seleccione la columna que va a servir como pivote para escribir y leer tags
- **Paso 4 Selección las Columnas:** Seleccionar las columnas que van a ser probadas por medio de las casillas de selección (5)
- **Paso 5 Escribir Valor para Tags**: En el campo valor se digita el valor que van a tomar los tags al ejecutar la prueba
- **Paso 6 Ejecutar la Prueba:** al oprimir el botón "Ejecutar test" se comenzarán a leer ya escribir los tags en PLC y si hay conexión con el HMI también se mostrarán los cambios en esta.
- **Paso 7 Ejecutar Pruebas Individuales**

  Seleccione una única fila en la tabla haciendo clic sobre ella se mostrara una ventana emergente con los campos de "OPC_Tag", "Valor", "Resultado" y "Observaciones", además de los botones "Monitor", "Cerrar" y "Guardar Cambios". Ver figura 6

  Para Escribir un Valor en un Tag Único: Coloque el valor en el campo valor de la ventana emergente y oprima el botón guardar, el resultado de la operación se verá reflejado en el campo Resultado de la ventana emergente.

Para Monitorear un Tag Único: Oprima el botón monitor, le mostrara una ventana emergente con el campo “Intervalo” y los botones “Iniciar/Detener” y “Limpiar”, defina el intervalo de monitoreo y oprima el botón “Iniciar” se mostrará una lectura serial del valor en tiempo real del tag y la hora de la captura del dato. Ver figura 7

Editar un Tag o Escribir una Observación: En la ventana emergente de que se muestra al hacer doble clic en la fila, son editables los campos “OPC_Tag”, “Valor”, y “Observaciones”, si necesita cambiar un OPC_Tag para hace una verificación lo puede hace en esta ventana, también si necesita hacerle una observación a la fila puede digitar el campo “Observaciones”

- **Paso 8 Monitoreo de Múltiples Tags**

  a) **Marque las casillas de selección** (5) de todas las filas que desee monitorear simultáneamente.

  b) Haga clic en (9) botón “Monitor Multi”. Se abrirá la ventana de monitoreo con el campo “Intervalo” y los botones “Iniciar/Detener” y “Limpiar” y con una tabla que contiene los tags seleccionados previamente. ver figura 5, defina el intervalo de monitoreo y oprima el botón “Iniciar” para ejecutar la función que actualizará el valor de cada tag cada cierto intervalo tiempo

Apéndice C. Entregables y Site del Proyecto

Con el propósito de que los interesados conozcan el proyecto está habilitado el sitio web y una carpeta en la nube con todos los entregables del proyecto, además de la aplicación de escritorio en un archivo .exe. Site del Proyecto, Entregables