Universidad Autónoma del Caribe

|  |
| --- |
| MANUAL DE USUARIO |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Yamith J. Romero Aldana  Elian D. Ruidiaz Marino | Saul A. Pérez Pérez  Carlos G. Díaz Sáenz  Pablo D. Bonaveri |  |

TABLA DE CONTENIDO

[INTRODUCCIÓN 4](#_Toc208139727)

[Información General 4](#_Toc208139728)

[Descripción General 4](#_Toc208139729)

[Arquitectura del código principal: 4](#_Toc208139730)

[Gestión Dinámica de Sensores: 4](#_Toc208139731)

[Aplicaciones 5](#_Toc208139732)

[Instalación y Configuración 6](#_Toc208139733)

[Requisitos del Sistema 6](#_Toc208139734)

[Inicio desde Ejecutable 6](#_Toc208139735)

[Instalación desde Código Fuente 6](#_Toc208139736)

[Ejecución de Desarrollo 6](#_Toc208139737)

[Estructura de Importaciones: 7](#_Toc208139738)

[Configuración de Red 7](#_Toc208139739)

[Interfaz de Usuario 8](#_Toc208139740)

[Implementación del Splash Screen: 8](#_Toc208139741)

[Layout Principal 8](#_Toc208139742)

[Elementos de Interfaz 8](#_Toc208139743)

[Conexión con ESP32 10](#_Toc208139744)

[Procedimiento de Conexión 10](#_Toc208139745)

[Configuración de Red 11](#_Toc208139746)

[SELECCIÓN Y USO DE SENSORES 12](#_Toc208139747)

[Procedimiento general 12](#_Toc208139748)

[Selección de Sensor 12](#_Toc208139749)

[Cambio de Sensor 12](#_Toc208139750)

[Sistema de Logging 13](#_Toc208139751)

[Interpretación de Logs 14](#_Toc208139752)

[Formato de Mensajes 14](#_Toc208139753)

[Secuencias Típicas de Operación 14](#_Toc208139754)

[Cambio de Sensor 14](#_Toc208139755)

[Especificaciones Técnicas 15](#_Toc208139756)

[Arquitectura del Software 15](#_Toc208139757)

[Lenguajes y Frameworks 15](#_Toc208139758)

[Componentes Arquitecturales 15](#_Toc208139759)

[Capa de Presentación: 15](#_Toc208139760)

[Capa de Comunicación: 15](#_Toc208139761)

[Capa de Datos: 15](#_Toc208139762)

[Protocolos de Comunicación 16](#_Toc208139763)

[Apéndices 17](#_Toc208139764)

# INTRODUCCIÓN

## Información General

**Nombre del Software:**SENSORA\_CONNECT

**Fecha de Desarrollo:** 2025  
**Plataforma:** Python 3.x con PySide6  
**Categoría:** Software Educativo

## Descripción General

SensoraCore es una plataforma de instrumentación científica avanzada que proporciona control centralizado para un sistema distribuido de sensores basado en microcontrolador ESP32. La aplicación principal actúa como centro de comando unificado, permitiendo el monitoreo, control y análisis de datos de 12 diferentes de sensores especializados.

## Arquitectura del código principal:

El archivo main.py (2387 líneas) implementa la lógica central del sistema con componentes:

* **Función** app\_path(): Sistema inteligente de resolución de rutas compatible con PyInstaller
* **Clase**LoadingSplashScreen: Pantalla de inicio profesional con barra de progreso
* **Clase**ui(QMainWindow): Controlador principal de la aplicación con gestión de 12 widgets de sensores
* **Clase**ESP32Client: Cliente TCP para comunicación robusta con microcontrolador
* **Clase**EmittingStream: Redirección de logs a interfaz gráfica

## Gestión Dinámica de Sensores:

La aplicación maneja dinámicamente 12 módulos de sensores mediante:

* **Interfaz Gráfica Intuitiva**: Desarrollada en Qt6 para máxima usabilidad y rendimiento
* **Conectividad Robusta**: Comunicación TCP/IP estable con ESP32 para aplicaciones industriales
* **Modularidad**: Sistema de plugins para fácil extensión y personalización

## Aplicaciones

* **Educación Superior**: Laboratorios de ingeniería, física y ciencias aplicadas
* **Investigación Científica**: Adquisición de datos experimentales y validación de modelos
* **Desarrollo**: Prototipado rápido de sistemas de instrumentación

# Instalación y Configuración

## Requisitos del Sistema

* Sistema Operativo: Windows 10/11
* Procesador: Dual-core 2.0 GHz (Intel i3, AMD A8, ARM Cortex-A72)
* Memoria RAM: 4 GB
* Almacenamiento: 500MB de espacio libre
* Red: WiFi 802.11n o Ethernet 100Mbps

## Inicio desde Ejecutable

**Windows**

1. **Descargar** el ejecutable SensoraCore\_XXXX.exe desde el directorio de versiones
2. **Ejecutar** para inicio de programa.
3. **Verificar** firewall para permitir comunicaciones en puerto 8080

## Instalación desde Código Fuente

Preparación del Entorno

*# Clonar el repositorio*

git clone <url-repositorio> SensoraCore

cd SensoraCore

*# Crear entorno virtual*

python -m venv venv

*# Activar entorno virtual*

*# Windows:*

venv\Scripts\activate

*# Linux/macOS:*

source venv/bin/activate

*# Instalar dependencias*

pip install -r requirements.txt

## Ejecución de Desarrollo

Terminal

cd SC\_DesktopApp

python main.py

## Estructura de Importaciones:

Python

from Modules.simpleAngle.simpleAngle\_logic import SimpleAngleLogic

from Modules.angleArm.angleArm\_logic import AngleArmLogic

from Modules.infrared.infrared\_logic import InfraredLogic

*# ... (9 módulos adicionales)*

## Configuración de Red

1. **Conectar** ESP32 a la red WiFi de instrumentación
2. **Anotar** la dirección IP asignada al dispositivo
3. **Verificar** conectividad mediante ping al ESP32
4. **Configurar** reglas de firewall si es necesario

# Interfaz de Usuario

### Implementación del Splash Screen:

Python

class LoadingSplashScreen(QWidget):

# Widget sin marco con QVBoxLayout

# Timer de 30ms con incremento +1% hasta 100%

# Estilo CSS personalizado: fondo blanco, borde gris

# Tamaño fijo: 420x110 pixels

### Layout Principal

La interfaz principal está organizada en áreas funcionales claramente definidas:

**Panel Superior:**

1. **Campo de IP**: Entrada para dirección IP del ESP32 con validación en tiempo real
2. **Botón de Log**: Acceso al sistema de logging y debugging
3. **Botón Conectar**: Establece conexión con feedback visual inmediato
4. **Indicador de Estado**: LED virtual que muestra estado de conectividad

**Panel de Sensores:**

1. **12 botones de Sensor**: Organizados en lista ScrollArea para acceso rápido
2. **Habilitación Dinámica**: Solo sensores disponibles están activos
3. **Identificación Visual**: Iconos distintivos para cada tipo de sensor

**Área de Trabajo Central:**

1. **Widget de Bienvenida**: Información de inicio y guías rápidas
2. **Área de Sensor Activa**: Carga dinámicamente la interfaz del sensor seleccionado
3. **Área de Estado**: Muestra información de sensor actual y estado de conexión

## Elementos de Interfaz

**Campo de Dirección IP**

* Formato: Validación automática de IPv4 (ej: 192.168.1.100)
* Placeholder: Texto de ayuda "Ingrese IP del ESP32"
* Validación: Comprobación en tiempo real con feedback visual

**Lista de Sensores Disponibles**

1. **SENSORA\_ANGLE\_ARM -** Medición de ángulos con brazo mecánico
2. **SENSORA\_BRIGHTNESS -** Sensor de luminosidad ambiente
3. **SENSORA\_CAPACITIVE -** Detección capacitiva de proximidad
4. **SENSORA\_COLOR\_CNY -** Sensor de color CNY70
5. **SENSORA\_COLOR\_TCS -** Sensor de color TCS3200
6. **SENSORA\_GAS\_REGULATION -** Monitoreo de gases
7. **SENSORA\_INFRARED -** Detección infrarroja
8. **SENSORA\_IR\_STEERING -** Control direccional IR
9. **SENSORA\_OPTICAL\_SPEED -** Medición de velocidad óptica
10. **SENSORA\_SIMPLE\_ANGLE -** Medición angular simple
11. **SENSORA\_THERMOREGULATION -** Medición térmica
12. **SENSORA\_ULTRASONIC -** Medición ultrasónica de distancia

#### **Indicadores de Estado**

* **LED de Conexión**: Segundo estado (rojo, azul, verde)
* **Barra de Estado**: Información textual del estado actual
* **Contador de Datos**: Muestra cantidad de mediciones recibidas en log
* **Indicador de Latencia**: Tiempo de respuesta de comunicación

# Conexión con ESP32

## Procedimiento de Conexión

**Paso 1: Preparación**

1. **Verificar** que el ESP32 esté encendido y conectado a la red
2. **Obtener** la dirección IP del ESP32 (consultar router o monitor serie en Thonny)
3. **Confirmar** que el ESP32 ejecuta el firmware SensoraCore correcto
4. **(Opcional) Verificar** conectividad de red mediante ping

**Paso 2: Configuración en la Aplicación**

1. **Introducir** la dirección IP del ESP32 en el campo correspondiente
2. **Verificar** que el formato IP sea válido (aparecerá borde verde)
3. **Hacer clic** en el botón "Conectar"
4. **Observar** el indicador de estado durante el proceso de conexión

**Paso 3: Verificación de Conexión**

1. **Esperar** la confirmación visual (LED de conexión)
2. **Verificar** que los botones de sensor se habiliten automáticamente
3. **Comprobar** en el log que aparezca "Conexión exitosa con ESP32"
4. **Probar** selección de sensor para validar comunicación bidireccional

**Solución de Problemas de Conexión**

**Error: "IP no válida"**

* **Causa**: Formato de IP incorrecto
* **Solución**: Verificar formato IPv4 (ej: 192.168.1.100)
* **Ejemplo válido**: 192.168.4.1 (red AP del ESP32)

**Error: "No se puede conectar al ESP32"**

* **Verificar conectividad de red**:

ping 192.168.1.100

PowerShell

* **Comprobar puerto 8080**:

PowerShell

telnet 192.168.1.100 8080

**Error: "Timeout de conexión"**

* **Aumentar timeout** en configuración avanzada
* **Verificar carga** del ESP32 (puede estar ocupado)
* **Reiniciar** el ESP32 y volver a intentar

**Error: "Respuesta inesperada del ESP32"**

* **Verificar versión** del firmware ESP32
* **Comprobar integridad** de la comunicación
* **Reiniciar** tanto la aplicación como el ESP32

## Configuración de Red

Implementación del Cliente TCP:

class ESP32Client:

def \_\_init\_\_(self, esp32\_ip, port=8080):

self.esp32\_ip = esp32\_ip

self.port = port

def send\_command(self, command):

*# Socket TCP con timeout de 3 segundos*

*# Codificación UTF-8 para comandos*

*# Buffer de recepción: 1024 bytes*

*# Manejo de excepciones: return "ERROR: {e}"*

Red de Infraestructura (Station Mode)

*# Configuración para red WiFi existente*

SSID: "Red\_Laboratorio"

Password: "clave\_segura"

IP: DHCP asignado automáticamente

Puerto: 8080

# SELECCIÓN Y USO DE SENSORES

## Procedimiento general

### Selección de Sensor

1. **Asegurar** conexión activa con ESP32 (LED ON)
2. **Hacer clic**  en el botón del sensor deseado
3. **Esperar** carga de la interfaz del sensor
4. **Verificar** que la información del sensor aparezca correctamente

### Cambio de Sensor

1. **Hacer clic** en u botón de sensor diferente
2. **Confirmar** automáticamente la limpieza del sensor anterior
3. **Esperar** carga de la nueva interfaz
4. **Verificar** funcionamiento del nuevo Sensor

**Proceso Técnico de cambio de sensor:** La función sensorSeleccionado(sensor\_id) ejecuta la siguiente secuencia:

1. **Limpieza automática**: \_cleanup\_active\_sensors() detiene procesos activos
2. **Ocultación de welcome**: welcome\_widget.setVisible(False)
3. **Gestión de layout**: Eliminación de widgets con takeAt(0) y deleteLater()
4. **Carga de widget específico**: Instanciación de lógica correspondiente
5. **Verificación de integridad**: Test con objectName() y recreación si falla
6. **Fallback automático**: reconstruirSensorUi() si hay errores de widget

# Sistema de Logging

Es un componente para monitoreo y diagnóstico, implementado con la clase `EmittingStream` que redirige stdout y stderr a un widget de texto en la interfaz gráfica.

**Apertura de Ventana de Log**

1. **Hacer clic** en el botón "Log" al lado del campo IP
2. **Observar** apertura de ventana independiente de logging
3. **Redimensionar** la ventana según necesidades de visualización

**Cierre de Ventana de Log**

1. **Hacer clic** nuevamente en el botón "Log" para alternar visibilidad
2. **Usar** la X de la ventana para cerrar manualmente
3. **El logging** continúa funcionando en segundo plano

**Tipos de Mensajes**

**Mensajes de Información (INFO)**

* **Conexiones exitosas** con ESP32
* **Cambios de sensor** realizados correctamente
* **Inicio y fin** de procesos de calibración
* **Estados de sistema** y confirmaciones de operación

**Mensajes de Advertencia (WARNING)**

* **Timeouts de comunicación** no críticos
* **Valores fuera de rango** esperado pero válidos
* **Reconexiones automáticas** exitosas
* **Configuraciones subóptimas** pero funcionales

**Mensajes de Error (ERROR)**

* **Fallos de conexión** con ESP32
* **Errores de comunicación** críticos
* **Fallos de validación** de datos
* **Excepciones de sistema** capturadas

**Mensajes de Debug (DEBUG)**

* **Trazas de función** para desarrollo
* **Estados internos** de variables críticas
* **Secuencias de comandos** enviados/recibidos
* **Información detallada** de objetos

## Interpretación de Logs

### Formato de Mensajes

PoweShell

[TIMESTAMP] [NIVEL] [MÓDULO] - Descripción del mensaje

[2024-01-15 14:30:25] [INFO] [ESP32Client] - Conexión establecida con 192.168.1.100:8080

[2024-01-15 14:30:26] [WARNING] [SensorManager] - Timeout en respuesta de sensor, reintentando...

[2024-01-15 14:30:27] [ERROR] [UltrasonicSensor] - Valor de distancia fuera de rango: 450 cm

### Secuencias Típicas de Operación

Conexión Exitosa (PowerShell)

[INFO] [MainWindow] - Iniciando conexión con ESP32...

[INFO] [ESP32Client] - Validando dirección IP: 192.168.1.100

[INFO] [ESP32Client] - Estableciendo socket TCP...

[INFO] [ESP32Client] - Enviando comando LED\_ON para handshake

[INFO] [ESP32Client] - Respuesta recibida: LED\_ON\_OK

[INFO] [MainWindow] - Conexión exitosa, habilitando sensores

### Cambio de Sensor

PowerShell

[INFO] [SensorManager] - Limpiando sensor activo anterior

[INFO] [BrightnessSensor] - Deteniendo proceso de monitoreo

[INFO] [BrightnessSensor] - Liberando recursos de interfaz

[INFO] [SensorManager] - Cargando SENSORA\_ULTRASONIC

[INFO] [UltrasonicSensor] - Inicializando interfaz de usuario

[INFO] [UltrasonicSensor] - Configuración cargada exitosamente

# Especificaciones Técnicas

## Arquitectura del Software

### Lenguajes y Frameworks

* **Lenguaje Principal**: Python 3.8+ con type hints
* **GUI Framework**: PySide6 (Qt 6.x) para interfaz nativa
* **Networking**: Socket TCP nativo de Python
* **Threading**: QThread para operaciones asíncronas
* **Packaging**: PyInstaller para distribución standalone

## Componentes Arquitecturales

### Capa de Presentación:

* Interfaz gráfica modular con carga dinámica de widgets
* Sistema de splash screen con indicadores de progreso
* Gestión de eventos Qt con signals y slots
* Responsive design para diferentes resoluciones

### Capa de Comunicación:

* Cliente TCP con manejo de reconexión
* Protocolo de comandos personalizado para ESP32
* Gestión de timeouts y errores de red
* Serialización de datos para transmisión

### Capa de Datos:

* Almacenamiento de configuraciones en archivos locales
* Cache de datos en memoria para rendimiento
* Exportación a formatos estándar (CSV, JSON)
* Logging estructurado para auditoría

## Protocolos de Comunicación

Protocolo TCP/IP con ESP32

Puerto: 8080

Formato: ASCII text commands

Terminador: \n (newline)

Timeout: 5 segundos (configurable)

Codificación: UTF-8

Comandos Estándar

LED\_ON - Comando de handshake, enciende LED del ESP32

LED\_OFF - Apaga LED del ESP32

GET\_STATUS - Obtiene estado general del microcontrolador

RESET\_SYSTEM - Reinicia el ESP32

# Apéndices

**Apéndice A: Glosario de Términos**

**ADC (Analog-to-Digital Converter)**: Convertidor analógico-digital que transforma señales analógicas en valores digitales.

**API (Application Programming Interface)**: Interfaz de programación que define métodos de comunicación entre componentes de software.

**ESP32**: Microcontrolador de Espressif Systems con WiFi y Bluetooth integrados, base del sistema de instrumentación.

**GPIO (General Purpose Input/Output)**: Pines de propósito general configurables como entrada o salida digital.

**GUI (Graphical User Interface)**: Interfaz gráfica de usuario que permite interacción visual con la aplicación.

**Handshake**: Proceso de establecimiento de comunicación entre dos dispositivos para verificar conectividad.

**I2C (Inter-Integrated Circuit)**: Protocolo de comunicación serie síncrono para comunicación entre microcontroladores y sensores.

**IoT (Internet of Things)**: Red de dispositivos físicos conectados que pueden intercambiar datos.

**Latencia**: Tiempo de retardo entre el envío de un comando y la recepción de su respuesta.

**LDR (Light Dependent Resistor)**: Resistor dependiente de luz, sensor básico de luminosidad.

**PID (Proportional-Integral-Derivative)**: Algoritmo de control automático usado en sistemas de regulación.

**PWM (Pulse Width Modulation)**: Modulación por ancho de pulso, técnica para controlar potencia de salida.

**Qt**: Framework multiplataforma para desarrollo de interfaces gráficas de usuario.

**SensoraCore**: Nombre del sistema completo de instrumentación científica distribuida.

**TCP/IP (Transmission Control Protocol/Internet Protocol)**: Conjunto de protocolos de comunicación para redes.

**Timeout**: Tiempo máximo de espera antes de considerar una operación como fallida.

**UART (Universal Asynchronous Receiver-Transmitter)**: Protocolo de comunicación serie asíncrono.

**UI (User Interface)**: Interfaz de usuario, punto de interacción entre humano y máquina.

**WiFi**: Tecnología de comunicación inalámbrica basada en estándares IEEE 802.11.

**Apéndice B: Referencias y Documentación Adicional**

**Documentación Técnica**

* **Qt6 Documentation**: <https://doc.qt.io/qt-6/>
* **Python Official Documentation**: <https://docs.python.org/3/>
* **ESP32 Technical Reference**: <https://docs.espressif.com/projects/esp-idf/>
* **PySide6 Reference**: <https://doc.qt.io/qtforpython/>

**Recursos de Desarrollo**

* **GitHub Repository**: <https://github.com/YamithR/SensoraCore>

**Contacto y Soporte**

* **Email de Soporte**: [yamith.romero@uac.edu.co](mailto:yamith.romero@uac.edu.co)

Copyright © 2025 SensoraCore Project. **Versión del documento: 3.2 | Fecha: Agosto 2025** Para la versión más actualizada de este manual, visite: <https://github.com/YamithR/SensoraCore.git>