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 3](#_Toc208241195)

[Información General 3](#_Toc208241196)

[Descripción General 3](#_Toc208241197)

[CARACTERISTICAS TECNICAS 4](#_Toc208241198)

[HARDWARE COMPATIBLE 5](#_Toc208241199)

[REQUERIMIENTOS DE INSTALACIÓN Y CONFIGURACIÓN 6](#_Toc208241200)

[Requisitos del Sistema 6](#_Toc208241201)

[Dependencias de Software 6](#_Toc208241202)

[Proceso de Instalación 6](#_Toc208241203)

[INICIO RÁPIDO 7](#_Toc208241204)

[**Inicio del Sistema** 7](#_Toc208241205)

[ALGORITMOS IMPLEMENTADOS 8](#_Toc208241206)

[APLICACIONES EDUCATIVAS 9](#_Toc208241207)

[APÉNDICES 10](#_Toc208241208)

# INTRODUCCIÓN

## Información General

**Nombre del Software:**SENSORA\_BRIGHTNESS

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

## Descripción General

SENSORA\_BRIGHTNESS es un módulo software especializado diseñado para el monitoreo en tiempo real de niveles de luminosidad mediante sensores de fotorresistencia (LDR - Light Dependent Resistor). Este software forma parte del ecosistema SensoraCore, una plataforma integral de módulos didácticos para el aprendizaje de sistemas embebidos, sensores analógicos y adquisición de datos.

El software proporciona una interfaz gráfica intuitiva que permite a estudiantes y profesionales monitorear la intensidad lumínica, visualizar datos en formato numérico y de barras de progreso, facilitando el entendimiento práctico de conceptos como conversión analógico-digital, caracterización de sensores ópticos y sistemas de medición luminosa.

# CARACTERISTICAS TECNICAS

**Arquitectura del Software**

* Lenguaje de Programación: Python 3.8+
* Framework GUI: PySide6 (Qt6) con widgets especializados
* Comunicación: Sockets TCP (puerto 8080) hacia firmware MicroPython en ESP32
* Visualización: QLCDNumber, QProgressBar y QLabel para datos múltiples
* Exportación: openpyxl para generación de archivos Excel

**Componentes Principales**

Interfaz de Usuario (brightness\_ui.py)

class Ui\_brightness:

"""

Interfaz principal del módulo con:

- Display LCD para porcentaje de luminosidad

- Barra de progreso visual (0-100%)

- Labels de datos analógicos crudos

- Controles de monitoreo e inicio/pausa

"""

Hilo de Comunicación (\_BrightnessThread

class \_BrightnessThread(QObject):

"""

Manejo asíncrono de comunicación con ESP32:

- Recepción de datos ADC y voltaje

- Configuración de intervalo de muestreo

- Señales Qt para actualización de interfaz

- Control de estados de conexión

"""

Lógica de Control (BrightnessLogic)

class BrightnessLogic(QObject):

"""

Control principal del módulo:

- Gestión de estados de monitoreo

- Procesamiento de datos de luminosidad

- Actualización de displays múltiples

- Exportación de datos a Excel

"""

# HARDWARE COMPATIBLE

**Microcontrolador Principal**

**Modelo**: ESP32 DevKit V1

**Conectividad**: WiFi integrado para comunicación TCP

* **GPIO**: Pin analógico ADC

**Sensor Requerido**

* Tipo: Fotorresistencia (LDR - Light Dependent Resistor)
* Rango Resistivo: 1kΩ-10MΩ (según iluminación)
* Tiempo de Respuesta: < 100ms para cambios graduales
* Conexión: Divisor de voltaje con resistencia fija

Diagrama de Conexiones

┌─────────────────────────────────┐

│ ESP32 DevKit V1 │

│ │

│ 3V3 ○ ←── VCC (+) │

│ D34 ○ ←── Señal (LDR + R) │

│ GND ○ ←── GND (-) │

│ │

│ Circuito Divisor: │

│ 3V3 ── LDR ── D34 ── R ── GND │

│ │

└─────────────────────────────────┘

# REQUERIMIENTOS DE INSTALACIÓN Y CONFIGURACIÓN

## Requisitos del Sistema

* **Sistema Operativo:** Windows 10/11
* **Python:** Versión 3.8 o superior
* **RAM:** Mínimo 2GB, recomendado 4GB

## Dependencias de Software

*# requirements.txt*

PyQt5>=5.15.0

matplotlib>=3.5.0

numpy>=1.21.0

scipy>=1.7.0

openpyxl>=3.0.9

pyserial>=3.5

pandas>=1.3.0

## Proceso de Instalación

1. **Clonar Repositorio:** Descargar código fuente
2. **Instalar Dependencias:** pip install -r requirements.txt
3. **Configurar ESP32:** Flashear firmware compatible
4. **Ejecutar Software:** python main.py

# INICIO RÁPIDO

**Inicio del Sistema**

1. **Verificación de Hardware**:
   * Conectar ESP32 al puerto USB
   * Verificar conexiones de LDR en configuración divisor de voltaje
   * Ejecutar aplicación SensoraCore
   * Seleccionar módulo "Monitoreo de Brillo"
2. **Configuración de Comunicación**:
   * Hacer clic en "Iniciar Monitoreo"
   * Verificar conexión TCP con ESP32
   * Observar actualización en tiempo real de:
     + Display LCD mostrando porcentaje de luminosidad
     + Barra de progreso visual
     + Valores ADC crudos
   * Probar variaciones de luz (cubrir/iluminar sensor)

# ALGORITMOS IMPLEMENTADOS

Conversión ADC a porcentaje

def \_on\_sample(self, adc: int, volt: float, ts: float):

"""

Convierte lectura ADC a porcentaje de luminosidad:

- adc: valor 0-4095 del conversor ADC

- porcentaje = (adc \* 100.0 / 4095.0)

- rango resultante: 0-100%

"""

pct = int(round(max(0.0, min(100.0, (adc \* 100.0 / 4095.0)))))

Gestión de Estados Visuales

def \_run(self):

"""

Bucle principal de comunicación:

- Envía comando MODO:BRIGHTNESS

- Configura intervalo con BR\_START:<ms>

- Procesa flujo de datos formato SENSOR:LDR

- Extrae ADC y VOLT de respuesta estructurada

"""

Actualización de Displays Múltiples

def \_on\_sample(self, adc: int, volt: float, ts: float):

"""

Sincroniza actualización de todos los displays:

- LCD: muestra porcentaje calculado

- ProgressBar: establece valor 0-100

- Label: muestra valor ADC crudo

"""

# APLICACIONES EDUCATIVAS

**Niveles Académicos**

* **Educación Media:** Conceptos básicos de sensores ópticos
* **Técnico Superior:** Sistemas de adquisición de datos analógicos
* **Universitario:** Instrumentación y caracterización de sensores
* **Posgrado:** Procesamiento de señales y calibración

**Conceptos Didácticos Cubiertos**

* **Sensores Ópticos:** Fotorresistencias y su comportamiento
* **Conversión A/D:** Cuantización y resolución de ADC
* **Divisores de Voltaje:** Circuitos de acondicionamiento de señal
* **Comunicaciones:** Protocolos TCP y streaming de datos
* **Visualización:** Múltiples formas de representar datos

# 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>