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](#_Toc208149099)

[Información General 3](#_Toc208149100)

[Descripción General 3](#_Toc208149101)

[CARACTERISTICAS TECNICAS 4](#_Toc208149102)

[Arquitectura del Software 4](#_Toc208149103)

[**Componentes Principales** 4](#_Toc208149104)

[Funcionalidades Especificas 4](#_Toc208149105)

[HARDWARE COMPATIBLE 5](#_Toc208149106)

[Microcontrolador Principal 5](#_Toc208149107)

[Sensor Requerido 5](#_Toc208149108)

[Diagrama de Conexiones 5](#_Toc208149109)

[REQUERIMIENTOS DE INSTALACIÓN Y CONFIGURACIÓN 6](#_Toc208149110)

[Requisitos del Sistema 6](#_Toc208149111)

[Dependencias de Software 6](#_Toc208149112)

[Proceso de Instalación 6](#_Toc208149113)

[INICIO RÁPIDO 7](#_Toc208149114)

[ALGORITMOS IMPLEMENTADOS 8](#_Toc208149115)

[APLICACIONES EDUCATIVAS 11](#_Toc208149116)

[Niveles Académicos 11](#_Toc208149117)

[Conceptos Didácticos Cubiertos 11](#_Toc208149118)

[APÉNDICES 12](#_Toc208149119)

# INTRODUCCIÓN

## Información General

**Nombre del Software:**SENSORA\_ANGLE\_ARM

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

## Descripción General

El módulo SENSORA\_ANGLE\_ARM es un sistema avanzado de monitoreo angular y detección de proximidad que utiliza tres potenciómetros independientes para medición de ángulos y un sensor capacitivo para detección de presencia. El sistema implementa una arquitectura cliente-servidor basada en TCP que permite la comunicación entre un microcontrolador ESP32 y una aplicación de escritorio desarrollada en PySide6, proporcionando medición multicanal en tiempo real con capacidades de calibración independiente por canal.

# CARACTERISTICAS TECNICAS

## Arquitectura del Software

Patrón Cliente-Servidor TCP Multi-Canal: Comunicación simultánea de 4 sensores independientes

Threading Asíncrono: QThread para operaciones de red sin bloqueo de interfaz

Calibración Multi-Canal: Sistema independiente de calibración lineal por potenciómetro

Visualización Dual: Gráficas simultáneas de valores AD y ángulos calibrados

Persistencia JSON: Almacenamiento independiente de configuraciones por canal

**Componentes Principales**

**AngleArmThread**: Hilo de comunicación TCP para recepción de datos multi-sensor

**AngleArmLogic**: Controlador principal con gestión de calibración por canal

**Sistema de Calibración**: Regresión lineal independiente para cada potenciómetro

**Exportador Avanzado**: Generación de reportes Excel con metadatos y gráficas

**Visualización Matplotlib**: Gráficas integradas con ejes duales para análisis comparativo

## Funcionalidades Especificas

Medición simultánea de 3 ángulos independientes (-135° a +135°)

Detección digital de proximidad capacitiva (ON/OFF)

Calibración asistida independiente por canal con múltiples puntos

Visualización en tiempo real con ventana deslizante de 100 muestras

Exportación completa con puntos de calibración y análisis de error

# HARDWARE COMPATIBLE

## Microcontrolador Principal

**Modelo**: ESP32 DevKit V1

**Conectividad**: WiFi integrado para comunicación TCP

**ADC Multi-Canal**: 3 canales analógicos independientes (D32, D33, D34)

**GPIO Digital**: Pin D25 para sensor capacitivo

**Alimentación**: 3.3V regulado para potenciómetros y lógica

## Sensor Requerido

**Potenciómetros (3 unidades):**

**Tipo**: Potenciómetros lineales 10kΩ

**Rango Angular**: Típicamente 270° de rotación mecánica

**Señal de Salida**: Analógica 0-3.3V proporcional al ángulo

**Resolución**: 12 bits ADC (4096 valores discretos)

**Sensor Capacitivo:**

**Tipo**: Sensor de proximidad capacitivo digital

**Salida**: Digital (HIGH/LOW) compatible con lógica 3.3V

**Rango de Detección**: Variable según sensor específico

**Aplicación**: Detección de presencia de objetos o manos

## Diagrama de Conexiones

┌─────────────────────────────────┐

│ ESP32 DevKit V1 │

│ 3V3 ○ ←── Potenciómetros (+) │

│ GND ○ ←── Potenciómetros (-) │

│ D32 ○ ←── Potenciómetro 1 (S) │

│ D33 ○ ←── Potenciómetro 2 (S) │

│ D34 ○ ←── Potenciómetro 3 (S) │

│ D25 ○ ←── Sensor Capacitivo │

│ LED integrado: GPIO 2 │

└─────────────────────────────────┘

3 potenciómetros 10kΩ:

• Pin (+): Alimentación 3.3V (todos)

• Pin (S): Señales analógicas:

• Pin (-): Tierra (GND) (todos)

# 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**:
   * Confirmar todas las conexiones según diagrama
   * Verificar alimentación 3.3V a todos los potenciómetros
   * Asegurar que ESP32 esté energizado y conectado a WiFi
   * Probar movimiento completo de cada potenciómetro
2. **Configuración de Comunicación**:
   * Introducir IP del ESP32 en campo correspondiente
   * Verificar conectividad de red antes de iniciar
   * Confirmar que puerto 8080 esté disponible
3. **Inicio de Monitoreo**:
   * Hacer clic en "Iniciar Monitoreo"
   * Observar cambio de estado del botón a "Pausar"
   * Confirmar llegada de datos de los 4 sensores

# ALGORITMOS IMPLEMENTADOS

Comunicación TCP Multi-Sensor

*# Protocolo de comunicación para múltiples sensores*

def run(self):

self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.sock.connect((self.esp32\_ip, self.port))

self.sock.sendall(b'MODO:BRAZO\_ANGULO')

*# Verificación de handshake específico*

resp = self.sock.recv(64).decode('utf-8', errors='ignore').strip()

if 'BRAZO\_ANGULO\_OK' not in resp:

self.connection\_status.emit("Error: ESP32 no acepto modo BRAZO\_ANGULO")

return

*# Loop de recepción de datos multi-sensor*

while self.running:

chunk = self.sock.recv(256)

buffer += chunk.decode('utf-8', errors='ignore')

while '\n' in buffer:

line, buffer = buffer.split('\n', 1)

*# Parseo: POT1:x,ANG1:y,POT2:x,ANG2:y,POT3:x,ANG3:y,SENSOR:True/False*

try:

parts = line.split(',')

kv = {}

for p in parts:

if ':' in p:

k, v = p.split(':', 1)

kv[k.strip()] = v.strip()

l1 = int(kv.get('POT1', '0'))

a1 = int(kv.get('ANG1', '0'))

l2 = int(kv.get('POT2', '0'))

a2 = int(kv.get('ANG2', '0'))

l3 = int(kv.get('POT3', '0'))

a3 = int(kv.get('ANG3', '0'))

sensor = kv.get('SENSOR', 'False').lower() in ('true', '1', 'on')

self.data\_received.emit(l1, a1, l2, a2, l3, a3, sensor)

except Exception:

*# Ignorar líneas malformadas*

continue

Calibración Independiente por Canal

*# Sistema de calibración lineal independiente*

def calibrar(self):

*# Calibrar cada canal secuencialmente*

for idx in range(3): *# POT1, POT2, POT3*

proceed = QMessageBox.question(self, "Calibración",

f"¿Calibrar POT{idx+1} ahora?")

if proceed != QMessageBox.Yes:

continue

puntos = []

for i in range(1, num\_points + 1):

*# Pedir ángulo de referencia*

angle, ok = QInputDialog.getDouble(

self, f"POT{idx+1} - Punto {i}/{num\_points}",

"Fija el potenciómetro en el ángulo deseado y escribe el valor:",

0.0, -135.0, 135.0, 1)

if ok:

lectura = int(self.last\_l[idx])

puntos.append((lectura, float(angle)))

*# Regresión lineal por mínimos cuadrados*

if len(puntos) >= 2:

xs = [p[0] for p in puntos] *# Lecturas AD*

ys = [p[1] for p in puntos] *# Ángulos de referencia*

*# Cálculo de pendiente y ordenada*

n = len(xs)

mean\_x = sum(xs) / n

mean\_y = sum(ys) / n

numerador = sum((xs[i] - mean\_x) \* (ys[i] - mean\_y) for i in range(n))

denominador = sum((xs[i] - mean\_x) \*\* 2 for i in range(n))

if denominador != 0:

m = numerador / denominador

b = mean\_y - m \* mean\_x

*# Guardar calibración por canal*

self.cal[idx]['m'] = float(m)

self.cal[idx]['b'] = float(b)

self.cal[idx]['ok'] = True

Aplicación de Calibración

*# Aplicación de calibración con límites físicos*

def \_apply\_cal(self, idx: int, lectura: int, raw\_angle: int) -> float:

if not self.cal[idx]['ok']:

return float(raw\_angle if raw\_angle is not None else 0)

*# Aplicar ecuación de calibración*

angle\_calibrado = self.cal[idx]['m'] \* lectura + self.cal[idx]['b']

*# Clamp a límites físicos realistas*

return max(-135.0, min(135.0, angle\_calibrado))

*# Procesamiento de datos multi-sensor en tiempo real*

def \_on\_data(self, l1, a1, l2, a2, l3, a3, sensor):

*# Calcular ángulos calibrados*

ac1 = self.\_apply\_cal(0, l1, a1)

ac2 = self.\_apply\_cal(1, l2, a2)

ac3 = self.\_apply\_cal(2, l3, a3)

*# Actualizar ventana deslizante*

self.time\_idx.append(self.time\_idx[-1] + 0.1 if self.time\_idx else 0.0)

for i, (l\_val, a\_cal) in enumerate([(l1,ac1), (l2,ac2), (l3,ac3)]):

self.ad\_series[i].append(l\_val)

self.ang\_series[i].append(a\_cal)

*# Mantener ventana de 100 muestras*

if len(self.ad\_series[i]) > 100:

self.ad\_series[i].pop(0)

self.ang\_series[i].pop(0)

if len(self.time\_idx) > 100:

self.time\_idx.pop(0)

*# Actualizar gráfica dual*

for i in range(3):

self.lines\_ad[i].set\_data(self.time\_idx, self.ad\_series[i])

self.lines\_ang[i].set\_data(self.time\_idx, self.ang\_series[i])

# APLICACIONES EDUCATIVAS

## Niveles Académicos

* **Educación Media:** Conceptos básicos de sensores
* **Técnico Superior:** Sistemas de adquisición de datos
* **Universitario:** Instrumentación y control
* **Posgrado:** Procesamiento avanzado de señales

## Conceptos Didácticos Cubiertos

* **Electrónica Analógica:** Potenciómetros y divisores de voltaje
* **Conversión A/D:** Cuantización y resolución
* **Calibración:** Regresión lineal y estadística
* **Programación:** Interfaces gráficas y threads
* **Comunicaciones:** Protocolos serie y WiFi

# 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>