**Metodología para un Sistema de Adquisición de Datos de Humedad con el Sensor 3001-TH (RS-485, Versión 1) y Python en Windows**

En la actualidad, el monitoreo eficiente de variables ambientales como la humedad del suelo es esencial para optimizar el uso de recursos en agricultura y jardinería. Las mediciones in situ, es decir, realizadas de forma local, representan una valiosa fuente de información que puede aprovecharse para estudios y la creación de modelos de estimación basados en inteligencia artificial, como se propone en este trabajo de tesis.

A continuación, se describe la implementación de un sistema de adquisición de datos utilizando el sensor de humedad de suelo 3001-TH, diseñado específicamente para mediciones precisas y duraderas. Sensores robustos como este se perfilan como herramientas clave en proyectos que promueven la sostenibilidad y la optimización de recursos hídricos.

El sistema propuesto permite la recopilación y almacenamiento periódico de datos de humedad del suelo, los cuales pueden ser utilizados para análisis y automatización, como en sistemas de riego inteligente. La combinación del sensor 3001-TH con tecnologías de la información ofrece una solución modular, fácil de implementar y escalable, con la posibilidad de integrarse en plataformas de almacenamiento en la nube o sistemas de alerta en tiempo real.

Esta solución no solo respalda prácticas agrícolas sostenibles y mejora el cuidado de cultivos, jardines e invernaderos, sino que también contribuye al desarrollo y validación de modelos de estimación basados en datos climatológicos, como los propuestos en este proyecto. Al promover el uso eficiente del agua y la gestión responsable de los recursos naturales, esta propuesta se alinea con los objetivos de sostenibilidad y eficiencia en la agricultura moderna.

1. Selección e Instalación de Herramientas y Recursos

1.1. Hardware

* Sensor 3001-TH (Versión 1): Sensor capacitivo que mide la humedad del suelo mediante RS-485.
* Adaptador RS-485 a USB: Necesario para conectar el sensor a la computadora.
* Cables de conexión: necesarios para conectar el sensor al adaptador
* Computadora con Windows

1.2. Software

1. Python: en este caso mediante el ide Visual Studio Code
2. Controladores del adaptador RS-485 a USB: Actualizar o descargar según sea el caso para tener la comunicación adecuada.
3. Bibliotecas necesarias

2. Conexión del Hardware

2.1. Cableado del Sensor

1. Identificación de pines del sensor 3001-TH:

Diagrama

Descripción generada automáticamente 

* + Marrón = Fuente de alimentación +
  + Negro = Fuente de alimentación -
  + Amarillo (gris) = 485-A
  + Azul = 485-B

2.2 Conexión al adaptador RS-485 a USB:

1. Conexión al PC:
   * Insertar el adaptador en un puerto USB disponible y asegúrese de que sea reconocido por el sistema operativo.

2.2. Verificación de la Conexión

* Identificar el puerto COM asignado al adaptador mediante el Administrador de dispositivos de Windows.
* Es recomendable el uso de una herramienta como PuTTY o RealTerm para verificar la comunicación con el sensor.

3. Desarrollo del Sistema

3.1. Configuración Inicial en Python

1. Instalación de bibliotecas:

Mediante código en el bash, debemos preparar el ide con las bibliotecas necesarias para que el sensor y el sistema funcionen de manera adecuada, las bibliotecas utilizadas son las siguientes:

* pymodbus: Biblioteca para implementar comunicación Modbus (RTU, TCP) en Python.
* time: Módulo estándar para manejar funciones relacionadas con el tiempo (relojes, pausas, etc.).
* sqlite3: Módulo para interactuar con bases de datos SQLite en Python.
* Flask: Framework ligero para desarrollar aplicaciones web.
* Dash: Framework para construir aplicaciones web interactivas de visualización de datos.
* Plotly: Biblioteca para crear gráficos interactivos y visualizaciones avanzadas.
* Pandas: Herramienta poderosa para análisis y manipulación de datos estructurados (tablas y series).

Todas estas bibliotecas se instalan mediante el comando “pip3 install” más el nombre de la biblioteca.

1. Configuración de parámetros de comunicación:

* Velocidad de transmisión: 9600
* Parámetros RS-485 estándar: 8 bits de datos, sin paridad, 1 bit de parada

client = ModbusSerialClient(

    port='COM7',

    baudrate=9600,

    parity='N',

    stopbits=1,

    bytesize=8,

    timeout=1)

1. Lectura del sensor

El código a continuación es una función para leer datos del sensor que utiliza el protocolo Modbus RTU sobre RS-485. Este protocolo es común en sensores industriales debido a su robustez y fiabilidad en entornos ruidosos.

* Dirección de registro inicial: Lo primero a definir en la función es la dirección donde está el registro de lectura de sensor dado que en el protocolo Modbus, los datos están organizados en registros de 16 bits, cada uno identificado por una dirección única.
* Cantidad de registros a leer: Por la configuración del sensor, los datos se almacenan en dos registros los cuales se deben de combinar para obtener el valor completo de lectura.
* Solicitud al cliente modbus: El cliente modbus envia una solicitud al sensor para leer los registros especificados, este método devuelve una respuesta que contiene los datos de lectura del sensor de forma nativa o un error si la solicitud falla.
* Manejo de errores: Se verifica si la respuesta contiene un error para garantizar que los datos obtenidos son válidos.
* Extracción de registros: Los valores obtenidos se almacenan en dos registros Reg\_1 y Reg\_2 los cuales se combinan mediante una operación de bits, la operación “<<” desplaza los bits del primer registro 16 posiciones a la izquierda, formando la parte alta del valor final de 32 bits y usando el operador “|” se combinan ambos registros para formar el numero completo de la lectura de forma nativa, es decir, en hexadecimal.
* Conversión a decimal: Afortunadamente en el lenguaje Python no es necesario realizar operaciones para la conversión, ya que especificando el formato “int” automáticamente realiza la conversión, lo que es necesario realizar extra, es un ajuste de valor ya que, según el fabricante, el valor de lectura es un valor escalado, por lo que es necesario dividir entre 10 para convertir en un valor de porcentaje de humedad con un decimal de precisión.

Es común que en este tipo de casos se use un método conocido como CRC (cyclic reduncandy check) de forma manual dentro del código para validad errores cuando se usan sistemas de comunicación y almacenamiento de datos para validar la integridad de la información, sin embargo, mediante el uso de la biblioteca Modbus automáticamente incluye el CRC en los frames de solicitud y valida las respuestas, esto ayuda mucho a reducir la complejidad del código y reduciendo el riesgo de errores en la implementación del protocolo.

Hasta este punto, ya tenemos el sensor trabajando, capaz de obtener lecturas y mostrarlas de forma directa, lo siguiente en la metodología aquí propuesta, es generar una base de datos para almacenar los datos arrojados por el sensor, para este caso, la base de datos no tiene que ser muy robusta, solo es necesario la capacidad de almacenar nuestros datos, por lo tanto, se optó por “SQLite” ya que es ligero y compatible con Python.

* Crear o conectar la base de datos: el comando “conexion” dentro de nuestra función “Crear base de datos”, se revisa si existe una base de datos con el nombre especificado, si no es así, la crea automáticamente.
* Crear un cursor: El cursor es necesario para ejecutar los comandos SQL en la base de datos y poder interactuar con los datos.
* Crear tabla: En este caso, crearemos una tabla llamada “lecturas”, mediante el comando “CREATE TABLE IF NOT EXISTS;” para prevenir errores futuros al ejecutarse múltiples veces, pasándole argumentos a la tabla los cuales son los mismos que nos arroja la parte del sensor, como en este caso (id,fecha\_hora y humedad) de la siguiente forma (id INTEGER PRIMARY KEY AUTOINCREMENT, fecha\_hora TEXT NOT NULL, humedad REAL NOT NULL). Esto para asegurar que no tengamos valores nulos y en el caso del id sea una llave primaria única para cada fila
* Aplicar los cambios y cerrar: Mediante los comandos “commit” y “close” respectivamente, esto es una buena practica y asegura que las modificaciones son guardadas en la configuración de la base de datos, además de cerrar para la liberación de recursos.

En la función “guardar\_en\_base\_de\_datos(humedad)”, se registra la medición de la humedad en la base de datos SQlite previamente configurada.

* Conectar con base de datos: Primero que nada, se debe de establecer una conexión con la base de datos mediante “sqite3.conect” para acceder a la anteriormente creada la cual se nombró “lecturas\_sensor.bd”, si la base de datos no existe, el comando la creara automáticamente, así que se debe de asegurar de escribir correctamente el nombre de la base de datos que se quiera acceder.
* Creación de cursos: La finalidad de este comando es interactuar con la base de datos y poder ejecutar comandos SQL.
* Obtener fecha y hora: “fecha\_hora = time.strftime("%Y-%m-%d %H:%M:%S")”, esta lina de código obtiene la fecha y hora actual en el formato YYYY-MM-DD HH:MM:SS, formato ampliamente utilizado y compatible con muchas herramientas de análisis de datos
* Insertar datos en tabla: “cursor.execute("INSERT INTO lecturas (fecha\_hora, humedad) VALUES (?, ?)", (fecha\_hora, humedad))”, esta línea de código, agrega una nueva fila en la tabla “lecturas”, afectando a las columnas “fecha\_hora” y “humedad,” ya que la columna id genera automáticamente un numero nuevo
* Confirmar cambios y cerrar: Como anteriormente se mencionó, se logra con los comandos commit y close.

Conectar con el cliente modbus

* El método client.connect() intenta establecer una conexión con el dispositivo Modbus configurado previamente, perteneciente a la biblioteca pymodbus, devuelve “True” si la conexión es exitosa o un “False” si la conexión falla.
* Para tener respuestas de forma visual, se imprimen mensajes del caso, “Error, no se pudo conectar al sensor” en caso de Error y “Conectado al sensor” en caso de éxito.

Código para leer los datos del sensor y almacenarlos en una base de datos

Este código tiene una fase de prueba para pausar cada 20 lecturas y preguntar si continua con las lecturas, esto es meramente con la finalidad de prueba y experimento, una versión final se esperaría que corriera y obtuviera datos de forma continua.

Mediciones controladas para pruebas experimentales

Generamos un bucle que nos da la oportunidad de detener o continuar después de un numero especifico de lecturas, esto con la finalidad de monitorear de forma controlada el experimento, verificar datos y acciones en los demás componentes del sistema de adquisición.

* Iniciar contador: definimos un contador para rastrear el número de lecturas realizadas en cada ciclo y dar la oportunidad al usuario si desea continuar o parar el experimento.
* Mientras el bucle este activo, manda leer los datos del sensor utilizando la función “leer\_registros ()” asignándola a una variable nombrada “humedad” para este caso.
* Verificamos con un “if” que la variable no este vacía, si no es así, mediante la función “guardar\_en\_base\_datos(humedad)” mandamos guardar los datos del sensor en la base de datos
* Incrementamos el contador en 1 y esperamos 1 segundo antes de la siguiente lectura.
* Verificamos mediante otro “if” la condición del número máximo de lecturas establecido, si no se ha cumplido, se repite el proceso de lectura y almacenamiento, de lo contrario, emitirá un mensaje que permita al usuario decidir si desea continuar tomando lecturas o no, de ser así, el programa reinicia el contador y nos da otras lecturas hasta llegar de nuevo al número establecido, de lo contrario el programa se cerrará, terminando el experimento.

Para poder acceder a los datos almacenados en nuestra base de datos de forma remota, se puede hacer de forma sencilla mediante una API rest utilizando Flask para exponer los datos.

* Flask: es un Framework web en Python que facilita la creación de aplicaciones web, mediante “jsonify” se convierten las estructuras de datos de Python en respuestas tipo “JSON”, el cual es un formato estándar para intercambiar datos en aplicaciones web.
* Crear la aplicación Flask: para crear la app necesitamos pasarle el argumento del nombre del modulo actual mediante “\_name\_” para que pueda acceder a los recursos.
* Definir la ruta: @app.route define la url del endpoint y los métodos HTTP permitidos
* Obtener lecturas: Esta función se activa cuando un cliente realiza una solicitud GET a la URL espeficicada
* Acceso a la base de datos: para acceder a la base de datos, se realizan los paso tal cual los códigos pasados, mediante la conexión “sqlite3”, los datos se obtienen y se almacenan temporalmente en la variable “datos” la cual convertiremos en respuesta JSON mediante “jsonify”.
* Configuración del servidor: configurando el “**host='0.0.0.0'”** para permitir que el servidor sea accesible desde cualquier dispositivo en la red local, y configurando el puerto “**port=5000**” ya que el puerto 5000 es el predeterminado para Flask.

Al corer el programa, nos generara un mensaje como el mostrado a continuacion, donde muestra el estado del servidor y la dirección web donde podremos acceder a la lista.

\* Serving Flask app 'api\_sensor'

\* Debug mode: off

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

\* Running on all addresses (0.0.0.0)

\* Running on http://127.0.0.1:5000

\* Running on http://10.0.0.133:5000

Press CTRL+C to quit

Texto

Descripción generada automáticamente con confianza media

Como extra, generamos una ayuda visual con “Dash” y “Ploty” para obtener una grafica en linea de los valores lectura de la base de datos.

* Importar librerías:
  + Dash: Framework para construir aplicaciones web interactivas en Python.
  + dcc (Dash Core Components): Ofrece componentes interactivos como gráficos y controles deslizantes.
  + html: Proporciona componentes HTML para la estructura del dashboard
  + plotly.express: Simplifica la creación de gráficos interactivos.
  + sqlite3: Permite conectar y consultar la base de datos SQLite., pandas: Se usa para manejar y analizar datos en formato tabular.
* Obtener datos: Mediante una función la cual incluye la conexión a la base de datos mediante “sqlite3” obtendremos las lecturas del sensor de la base de datos al ejecutar la consulta SQL y convertir los resultados en un dataframe de pandas para facilitar su manipulación.
* Creación del grafico: Para este caso se genera un grafico simple, solo de línea, ya que nuestros datos son en dos dimensiones, los cuales constan del tiempo, el cual se representa en el eje X, y las mediciones del sensor, los cuales se reflejan en el eje Y.
* Diseño del Dashboard: mediante “app.layout” se define la estructura visual de la aplicación y con “html.Div” obtendremos nuestro contenedor principal para agrupar los elementos HTML, por último,”dcc.Graph” nos muestra el grafico interactivo que se genero previamente.

Al correr el programa, este nos arroja un mensaje como el siguiente:

Dash is running on http://127.0.0.1:8050/

\* Serving Flask app 'dashboard'

\* Debug mode: on

Donde nos especifica la dirección web donde podremos acceder al grafico interactivo generado.

Tabla

Descripción generada automáticamente

* import time
* import sqlite3
* from pymodbus.client import ModbusSerialClient
* # Configuración el cliente Modbus
* client = ModbusSerialClient(
* port='COM7',
* baudrate=9600,
* parity='N',
* stopbits=1,
* bytesize=8,
* timeout=1
* )
* # Función para leer registros del sensor
* def leer\_registros():
* start\_address = 0x0001  # Dirección de registro inicial
* length = 2
* try:
* # Lectura de registros
* response = client.read\_holding\_registers(start\_address, length)
* if response.isError():
* print(f"Error en la respuesta: {response}")
* return None
* else:
* # Obtener los valores de los registros
* reg\_1 = response.registers[0]
* reg\_2 = response.registers[1]
* # Combinar registros para obtener el valor completo (32 bits)
* valor\_hex = (reg\_1 << 16) | reg\_2
* #print(f"Valor hexadecimal combinado: 0x{valor\_hex:08X}")
* # Convertir a decimal
* valor\_decimal = int(valor\_hex)
* #print(f"Valor decimal de humedad: {valor\_decimal}")
* # Ajustar el valor de escala a porcentaje
* humedad = valor\_decimal / 10.0  # Ajusta esto según el sensor
* print(f"Humedad: {humedad:.1f}%\n")
* return humedad
* except Exception as e:
* print(f"Error durante la lectura: {e}")
* return None
* # Configuración de la base de datos SQLite
* def crear\_base\_datos():
* conexion = sqlite3.connect("lecturas\_sensor.db")
* cursor = conexion.cursor()
* cursor.execute('''
* CREATE TABLE IF NOT EXISTS lecturas (
* id INTEGER PRIMARY KEY AUTOINCREMENT,
* fecha\_hora TEXT NOT NULL,
* humedad REAL NOT NULL
* )
* ''')
* conexion.commit()
* conexion.close()
* # Función para almacenar una lectura en la base de datos
* def guardar\_en\_base\_datos(humedad):
* conexion = sqlite3.connect("lecturas\_sensor.db")
* cursor = conexion.cursor()
* fecha\_hora = time.strftime("%Y-%m-%d %H:%M:%S")
* cursor.execute("INSERT INTO lecturas (fecha\_hora, humedad) VALUES (?, ?)", (fecha\_hora, humedad))
* conexion.commit()
* conexion.close()
* # Crear la base de datos y la tabla
* crear\_base\_datos()
* # Conectar al cliente Modbus
* if not client.connect():
* print("Error: No se pudo conectar al sensor.")
* else:
* print("Conectado al sensor.")
* # Inicializar un contador de lecturas
* contador\_lecturas = 0
* # Bucle para tomar lecturas, almacenar en la base de datos y preguntar al usuario después de 20 lecturas
* try:
* while True:
* humedad = leer\_registros()  # Leer datos del sensor
* if humedad is not None:
* guardar\_en\_base\_datos(humedad)  # Guardar la lectura en la base de datos
* print("Lectura almacenada en la base de datos.")
* contador\_lecturas += 1
* time.sleep(1)  # Esperar 1 segundo antes de la siguiente lectura
* # Preguntar al usuario si desea continuar después de 20 lecturas
* if contador\_lecturas == 20:
* user\_input = input("¿Deseas continuar tomando lecturas? (s/n): ")
* if user\_input.lower() != 's':
* print("Cerrando el programa...")
* break  # Rompe el bucle y termina el programa
* else:
* contador\_lecturas = 0  # Reiniciar el contador después de la pregunta
* except KeyboardInterrupt:
* print("Lectura detenida por el usuario.")
* # Cerrar la conexión Modbus
* client.close()

Código para acceder y mostrar las lecturas de la base de datos

import sqlite3

# Función para leer y mostrar las lecturas almacenadas en la base de datos

def leer\_lecturas():

    # Conectar a la base de datos

    conexion = sqlite3.connect("lecturas\_sensor.db")

    cursor = conexion.cursor()

    try:

        # Ejecutar consulta para obtener todas las lecturas

        cursor.execute("SELECT \* FROM lecturas")

        filas = cursor.fetchall()

        # Mostrar las lecturas

        print(f"\n{'ID':<5} {'Fecha y Hora':<20} {'Humedad (%)':<10}")

        print("-" \* 35)

        for fila in filas:

            print(f"{fila[0]:<5} {fila[1]:<20} {fila[2]:<10.1f}")

    except Exception as e:

        print(f"Error al leer la base de datos: {e}")

    finally:

        # Cerrar la conexión

        conexion.close()

# Llamar a la función para leer y mostrar las lecturas

leer\_lecturas()

Código para crear una API y compartir los datos

from flask import Flask, jsonify

import sqlite3

app = Flask(\_\_name\_\_)

@app.route('/lecturas', methods=['GET'])

def obtener\_lecturas():

    conexion = sqlite3.connect("lecturas\_sensor.db")

    cursor = conexion.cursor()

    cursor.execute("SELECT \* FROM lecturas")

    datos = cursor.fetchall()

    conexion.close()

    return jsonify(datos)

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(host='0.0.0.0', port=5000)

Código para generar un Dash grafico

from dash import Dash, dcc, html

import plotly.express as px

import sqlite3

import pandas as pd

app = Dash(\_\_name\_\_)

def obtener\_datos():

    conexion = sqlite3.connect("lecturas\_sensor.db")

    df = pd.read\_sql\_query("SELECT \* FROM lecturas", conexion)

    conexion.close()

    return df

df = obtener\_datos()

fig = px.line(

    df,

    x="fecha\_hora",

    y="humedad",

    title="Mediciones de Humedad del Sensor 3001-TH",

    labels={

        "fecha\_hora": "Tiempo",

        "humedad": "Humedad (%)"

    })

app.layout = html.Div([

    html.H1("Dashboard de Humedad"),

    dcc.Graph(figure=fig)

])

if \_\_name\_\_ == "\_\_main\_\_":

    app.run\_server(debug=True)