ENVIRONMENTAL MONITORING USING INTERNET OF THINGS

1. PROJECT DESCRIPTION

Internet of Things is a platform where every day devices surrounding us become smarter, every day collecting the data and processing becomes intelligent, and every day communication becomes more useful for people. The Internet of Things (IoT) has been defined in Recommendation ITU-T Y.2060 (ITU-T Study, 2015) as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies". While the Internet of Things is still evolving, its effects have already stared in making incredible changes in many areas as a universal solution media for providing services in completely different ways. Gartner (Rivera, Rob, 2014) reported that 25 billion devices will be connected to the internet by 2020 and those connections will facilitate the used data to analyze, preplan, manage, and make intelligent decisions autonomously. The US National Intelligence Council (NIC) has embarked IoT as one of the six''Disruptive Civil Technologies" (Council National Intelligence, 2008). In this context, we can see that service several sectors, such as: smart city, e-health,e-governance, transportation, e-education, retail, agriculture, process automation, industrial manufacturing, logistics and business management etc., are already getting benefited from various forms of IoT paradigm and technology. First wave of IoT systems in smart city domain emphasized on connecting sensor using lightweight protocols such as CoAP and XMPP (old fashioned publish/subscribe communication model). The Smart-Object devices with domain specific smart automatic rules are rapidly replacing first wave of IoT devices (Kortuem et al., 2010). Although these devices do not utilize semantic technologies (Park et al., 2014), they provide higher-level of sensor-to-application communication than just plain transfer of raw sensor data.Sensor data representation formats used in a sensor networks is various, and most of them cannot understand the value's meaning in other systems. Therefore, values and information from the systems must be prepared,shared and provided additional metadata information for other applications.The second challenge is develop knowledge discovery techniques and services for big smart city data storing, transfer and provide useful analytics.

Most of the current IoT services (figure 1) require IoT applications (BIG IoT API) to have knowledge of IoT middleware (communication software) and sensors or sensor networks for accessing IoT resources. Heterogeneous IoT middleware are not easy to be accessed by different applications since each IoT gateways has no open and standardized Application Programming Interface (APIs). Various organizations such as the OpenIoT Alliance, AllSeen Alliance, and IPSO Alliance are working on standardization of „Internet of Everything" communication protocols to provide interoperability betweenvarious solutions.

2. PROJECT IMPLEMENTATION

The implementation of the IoT-based Environmental Monitoring System involves a systematic integration of hardware, software, and communication protocols to achieve real-time data collection, analysis, and presentation.

In the hardware layer, an array of environmental sensors, including air quality sensors, temperature and humidity sensors, water quality sensors, and soil moisture sensors, are strategically deployed across the target area. Microcontrollers, such as Arduino or Raspberry Pi, serve as the brains of the operation, interfacing with the sensors, collecting raw data, and performing initial processing. These microcontrollers are equipped with wireless communication modules, such as Wi-Fi or LoRa, ensuring seamless transmission of data to the central monitoring system.

The communication layer employs protocols like MQTT or HTTP to facilitate the efficient exchange of data between the sensors and the central hub. Robust encryption mechanisms are implemented to safeguard data integrity during transmission, ensuring the privacy and security of the collected environmental information.

Data processing and storage occur in the cloud-based layer, where advanced algorithms convert raw sensor data into meaningful insights. A secure cloud database is utilized for storing processed data, enabling historical analysis and trend identification. This cloud-based architecture not only ensures scalability but also allows users to access environmental data remotely through a user-friendly web interface or mobile application.

The web interface or application layer is designed to provide an intuitive and interactive platform for users to access real-time and historical environmental data. Visualizations, such as graphs and charts, enhance data interpretation, empowering users to make informed decisions based on the presented information.

An intelligent alert system is implemented to notify stakeholders of any abnormal environmental conditions. Dynamic threshold values for each parameter trigger alerts, which are sent through multiple channels, such as email or SMS, ensuring that timely actions can be taken in response to potential environmental threats.

To optimize power consumption and promote sustainability, the power management layer incorporates energy-efficient measures. This includes implementing sleep modes for sensors during periods of inactivity and harnessing renewable energy sources, such as solar panels, to power the monitoring system.

Throughout the implementation, scalability is a key consideration, allowing the system to easily accommodate additional sensors or expand the monitoring network. Regular maintenance checks and procedures for software updates and hardware upgrades are established to ensure the reliability and longevity of the Environmental Monitoring System.

3. CIRCUIT DIAGRAM

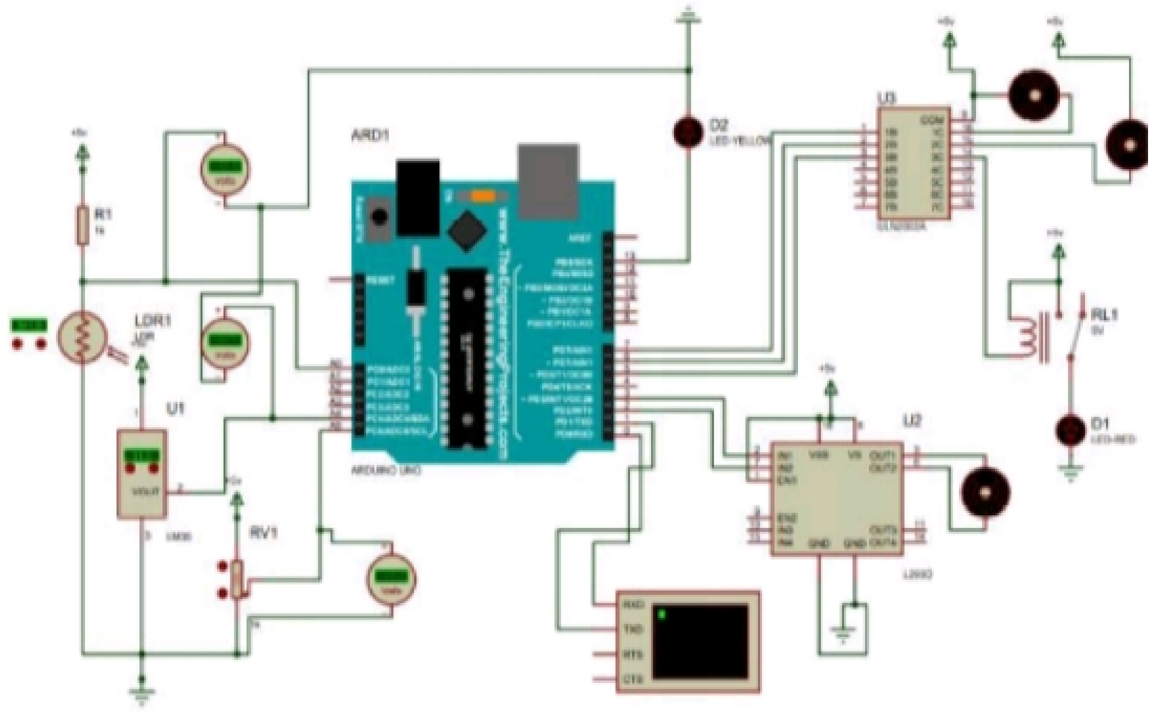

Fig.Circuit diagram

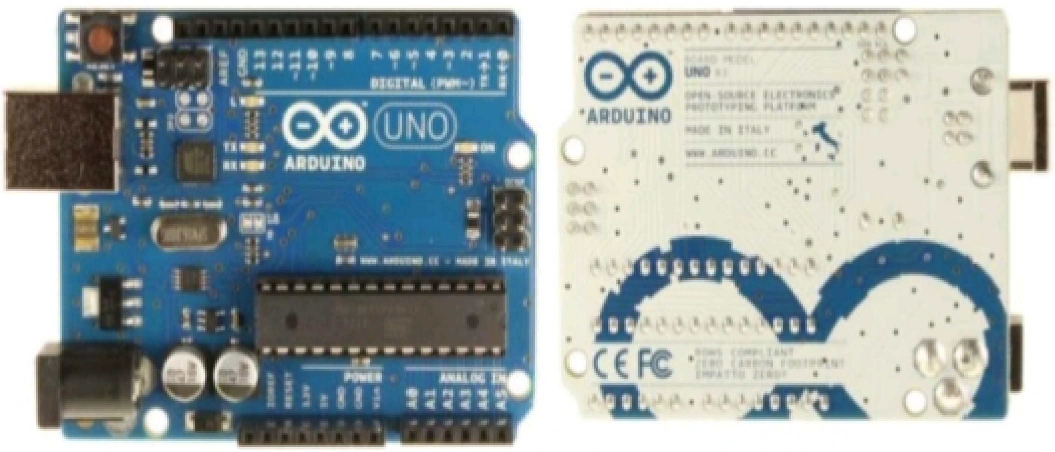
Fig. Top and Back view of Arduino UNO

4. SOURCE CODE

4.1 FRONT END(Python)

```
from nnfs import datasets
import numpy as np
from nnfs.datasets import spiral_data
import nnfs
from numpy import core
nnfs.init()
import matplotlib.pyplot as plt
# Importance of transposing
inp = np.array([[1, 2, 3, 2.5],
               [2, 5, -1, 2],
               [-1.5, 2.7, 3.3, -0.8]])
w = np.array([[0.2, 0.8, -0.5, 1],
             [0.5, -0.91, 0.26, -0.5],
             [-0.26, -0.27, 0.17, 0.87]])
b = [2,3,0.5]
w2 = np.array([[0.1, -0.14, 0.5],
              [-0.5, 0.12, -0.33],
              [-0.44, 0.73, -0.13]])
b2 = [-1, 2, -0.5]
out1 = np.dot(inp,w.T) + b
out2 = np.dot(out1, w2.T) + b2
# Introducing non linear data
X, y = spiral_data(samples = 100, classes = 3)
print(X.shape, y.shape)
# plt.scatter(X[:,0], X[:,1], c= y, cmap = 'brg')
# plt.show()
# print(X.shape)
# Defining dense layers
class Layer_Dense:
    def __init__(self, n_inputs, n_neurons):            # n_inputs are inputs, n_output are
output neurons
```

```
        self.weights =  0.01*np.random.randn(n_neurons, n_inputs)
        self.biases = np.zeros((1, n_neurons))
    def forward(self, inputs):
        self.inputs = inputs                              # to remember for backpropagation
        self.output = np.dot(inputs, self.weights.T) + self.biases
    def backward(self, dvalues):
        self.dweights = np.dot(dvalues.T, self.inputs)
        self.dbiases = np.sum(dvalues, axis = 0, keepdims = True)
        self.inputs = np.dot(dvalues, self.weights)
class activation_Relu:
    def forward(self, inputs):
        self.inputs = inputs
        self.output = np.maximum(0,inputs)
    def backward(self, dvalues):
        self.dinputs  = dvalues.copy()   #np.copy(dvalues)
        self.dinputs[self.inputs <= 0] = 0
class activation_softmax:
    def forward(self, inputs):
        self.inputs = inputs
        exp_values = np.exp(inputs-np.max(inputs, axis =  1, keepdims = True))
         self.output  = exp_values / np.sum(exp_values, axis =1 , keepdims =True)        #
Probabilties
        def backward(self,dvalues):
        self.dinputs = np.empty_like(dvalues)
                for index, (single_output, single_dvalues) in enumerate (zip(self.output,
dvalues)):
            single_output  = single_output.reshape(-1,1)
                jacobian_matrix  =  np.diagflat(single_output)  -  np.dot(single_output,
single_output.T)
            self.dinputs[index] = np.dot(jacobian_matrix, single_dvalues)
class Loss:
    def calculate(self, output, y):
        sample_losses = self.forward(output, y)
        data_loss = np.mean(sample_losses)
        return data_loss
class Loss_categoricalcrossentropy(Loss):
    def forward(self, y_pred, y_true): # suppose if we have predicted output and ground
truth labels
            y_pred_clipped = np.clip(y_pred, np.exp(-7), 1-np.exp(-7)) # Excluding the
values less than e-7 and higher that 1=e-7
     def backward(self, dvalues, y_true):
        labels = len(dvalues[0])
        if len(y_true.shape) == 1:
            y_true = np.eye(labels)[y_true]

        self.dinputs = -y_true/dvalues
        self.dinputs = self.dinputs/len(dvalues)
class activation_softmax_loss_categoricalentropy():
    def __init__(self):
        self.activation = activation_softmax()
        self.loss = Loss_categoricalcrossentropy()

    def forward(self, inputs, y_true):
        self.activation.forward(inputs)
        self.output = self.activation.output
        return self.loss.calculate(self.output, y_true)
    def backward(self, dvalues, y_true):
        if len(y_true.shape) == 2:
            y_true = np.argmax(y_true, axis = 1)
        self.dinputs = dvalues.copy()
        self.dinputs[range(len(dvalues)), y_true] -=1
        self.dinputs = self.dinputs /len(dvalues)
class optimizer_SGD():
    def __init__(self, learning_rate = 1.0):
        self.learning_rate = learning_rate
    def update_params(self, layer):
        layer.weights += -self.learning_rate*layer.dweights
        layer.biases += -self.learning_rate*layer.dbiases
```

```
# Creating a object class of Layer_Dense()
layer1 = Layer_Dense(2, 64)
# create a object of class Relu
activation1 = activation_Relu()
# second layer
layer2 = Layer_Dense(64,3)
# pass through activation function
activation1.forward(layer1.output)
# pass through second layer
layer2.forward(activation1.output)
# pass through second activation layer
activation2.forward(layer2.output)
# computing loss for the forward pass
loss = loss_function.calculate(activation2.output, y)
#loss = loss_activation.forward(layer2.output,y)
# Accuracy calculation
predictions = np.argmax(activation2.output, axis = 1)
if len(y.shape) == 2:
    y = np.argmax(y, axis = 1)
accuracy = np.mean(predictions == y)
print(loss)
```

4.2 BACK END(JSON)

```
        // Import necessary modules
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

// Connect to MongoDB (replace 'your-database-url' with your actual MongoDB connection string)
mongoose.connect('your-database-url', { useNewUrlParser: true, useUnifiedTopology: true });
```

```
// Create a MongoDB schema and model for the environmental data
const environmentalDataSchema = new mongoose.Schema({
  pm25: Number,
  temperature: Number,
  humidity: Number,
  timestamp: { type: Date, default: Date.now }
});

const EnvironmentalData = mongoose.model('EnvironmentalData',
environmentalDataSchema);

// Create the Express application
const app = express();

// Middleware to parse JSON in the request body
app.use(bodyParser.json());

// Endpoint to receive environmental data from IoT devices
app.post('/api/environmental-data', (req, res) => {
  // Extract data from the JSON request body
  const { pm25, temperature, humidity } = req.body;

  // Create a new EnvironmentalData instance
  const newEnvironmentalData = new EnvironmentalData({
    pm25,
    temperature,
    humidity
  });
  // Save the data to the MongoDB database
  newEnvironmentalData.save((err) => {
    if (err) {
      console.error(err);
      res.status(500).send('Internal Server Error');
    } else {
      res.status(201).send('Data received and saved successfully');
    }
  });
});

// Start the server (replace 3000 with your desired port)
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

SAMPLE OUTPUT

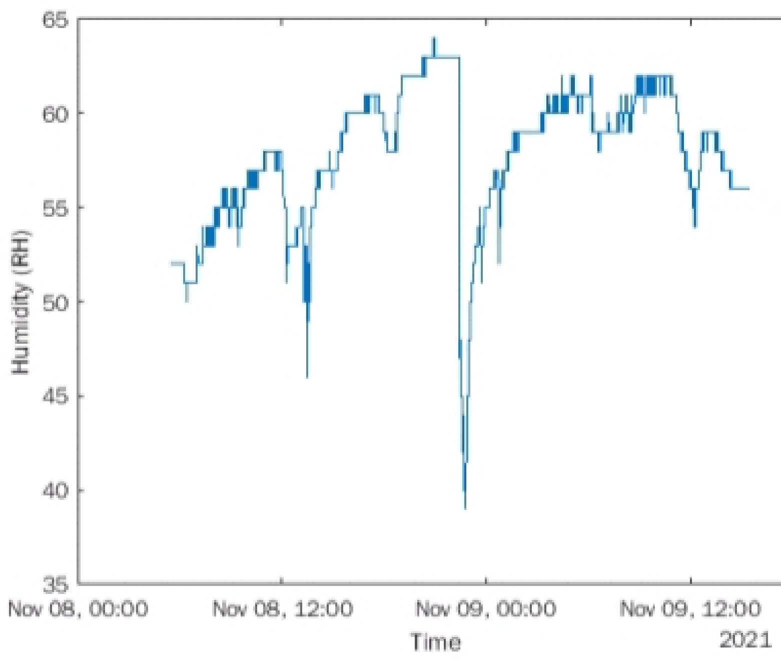

Fig 4.1.1: **Past_humidity_reading (1)**

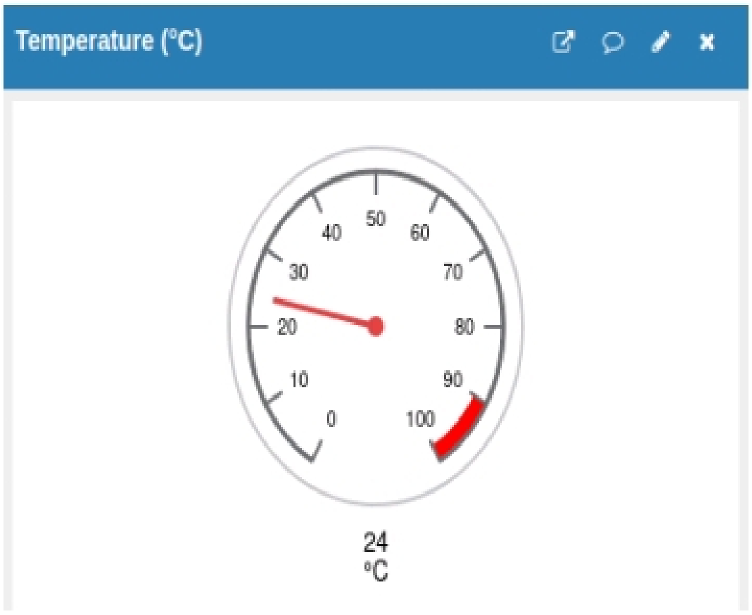
Temperature (°C)
0
10
20
30
40
50
60
70
80
90
100
24
°C

Fig4.1.2: Sample_temperature_reading

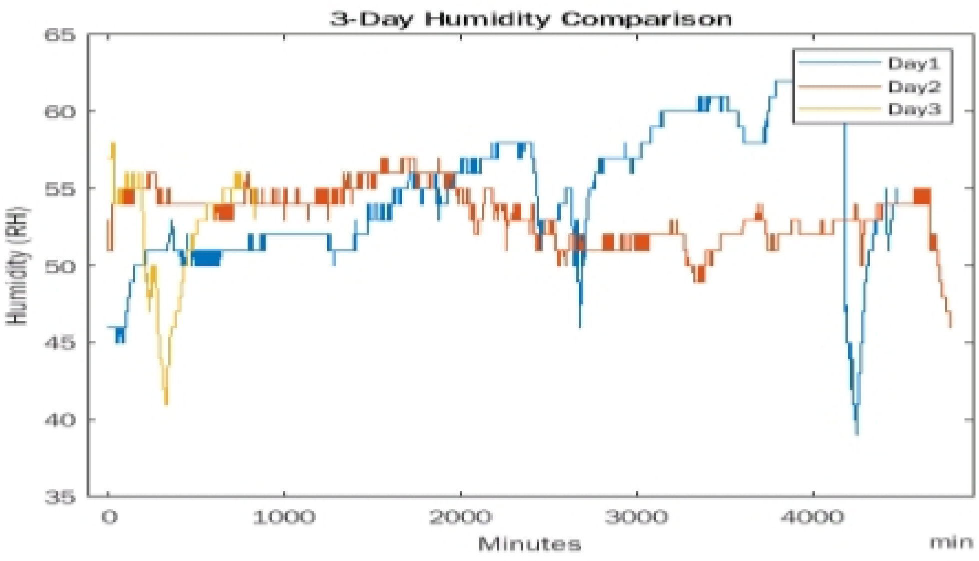

Fig4.2.1: Three_day__humidity_data