UDC 004.415.2
DOI https://doi.org/10.32782/2663-5941/2023.4/14

Oleshchenko L.M.
National Technical University of Ukraine "Igor Sikorsky Kyiv polytechnic institute"

Moshenskyi A.O.
National University of Food Technologies

HARDWARE AND SOFTWARE SYSTEM OF ENVIRONMENTAL INDICATORS MONITORING AND ANALYSIS BASED ON ESP8266 CONTROLLER

Hardware and software systems for monitoring and analyzing environmental indicators play an important role in the collection, analysis and understanding of data on the quality of air, water, soil and other aspects of the environment. Monitoring systems make it possible to obtain information on pollution levels, chemical composition, meteorological conditions and other environmental indicators. This provides an opportunity to assess the quality of the environment, identify problem areas and take measures to reduce pollution and improve the state of the environment. Thanks to monitoring systems, dangerous levels of pollution or chemicals in the environment can be detected in time. This helps to prevent potential threats to human health and to take necessary safety measures, such as evacuation or the use of protective equipment. Hardware and software systems allow monitoring the implementation of environmental standards and regulations established by authorities. This may include requirements for pollution levels, emissions of harmful substances, and monitoring compliance with environmental licenses and permits. Hardware and software monitoring systems allow detection of emergency situations, such as chemical leaks, fires or other environmental accidents. This allows to react quickly, take the necessary measures to minimize damage and ensure the safety of the population.

The article describes the environmental data monitoring system and software methods for analyzing temperature and air humidity data, as well as the signal level of the router with which the controller interacts. The obtained data were analyzed and visualized using analytical tools of the Python programming language. A correlation was found between temperature and humidity parameters, time periods of emergency shutdowns of the monitoring system due to emergency power outages, as well as time periods when the signal level was reduced due to the presence of people in the room for a selected period of time. The Google Colab cloud environment was used for data analysis. The technical characteristics of the monitoring system installed in Cherkasy city based on the ESP8266+HTU21 controller are described.

Key words: *streaming data, Python, environmental monitoring, temperature and humidity sensor, Wi-Fi signal level sensor, Internet of Things (IoT), ESP8266+HTU21.*

Introduction. Problem Statement. The importance of systems for monitoring and analyzing environmental indicators lies in the fact that they provide objective information necessary for making decisions about environmental protection, ensuring safety and creating sustainable environmental conditions for future generations. Air quality monitoring helps assess the environmental impact of various sources of pollution such as industrial plants, transportation systems, energy, etc. This provides a basis for making decisions to improve efficiency, eradicate hazardous sources of pollution and promote sustainable development. Air quality monitoring plays a crucial role in evaluating the environmental impact of industrial plants, transportation systems and energy production. By monitoring air quality, decision-makers can gather data to assess the level of pollution, identify sources of contamination, and make informed decisions to enhance efficiency and promote sustainable development. This information serves as a foundation for implementing measures to eliminate hazardous pollution sources, improve air quality, and protect human health and the environment.

Related research. Researchers continuously improve sensor technologies for air quality monitoring, focusing on accuracy, precision, and reliability. They use monitoring data to create detailed air pollution maps, identify hotspots, and understand pollutant dispersion [1]. Integration with GIS and predictive models helps estimate pollution levels in unmonitored areas [2]. Studies on health effects analyze the relationship between pollutant concentrations and respiratory diseases, cardiovascular issues, and mortality rates, aiding

in policy development [3]. Source apportionment models identify pollution sources, supporting targeted emission control [4]. Citizen science engages communities in monitoring, validating data, raising awareness, and influencing policies for better air quality [5]. ESP8266 based mesh network for smart monitoring systems was described, but usualy physical LR radio on the L1(L2) for range extension and sensivity increase was not imployed [6]. NodeMCU and different IoT Cloud implementation in the authors articles cycle was described [7]. Platform choice, power consumption in the mash modes and timeings were optimized rarely.

The main goal of the article is software analysis of streaming data of the environmental indicators monitoring system, in particular, air temperature and humidity and wireless network data transmission signal level in conditions of emergency power outages.

Existing software solutions for sensor data analysis. ThingSpeak is an IoT platform for collecting, analyzing, and visualizing sensor data. It integrates with various IoT platforms, supports MATLAB for advanced analysis, and offers APIs for easy integration with other applications. It's open-source with an active user community. Analogues to ThingSpeak include InfluxDB, a time-series database for sensor data analysis, Grafana for visualization, AWS IoT Analytics for scalable data processing and analysis, and Microsoft Azure IoT Hub for managing sensor data and integration with Azure services.

Research results. The authors use Google Colab cloud computing and Python programming language technologies to analyze streaming data from online platforms to which data from sensors are sent. The sensor based on ESP8266+HTU21, which was installed in August 2022 in the Cherkasy city [8] (Fig. 1).

The sensor is designed to measure the temperature and humidity of the air, in addition, it records the strength of the Wi-Fi network signal and the voltage level of the battery [9].

Consider the technical characteristics of the IoT system based on ESP8266+HTU21 microcontroller (Fig. 2). The pinout of the ESP8266 microcontroller is shown in Fig. 3.

HTU21 temperature and air humidity sensor is connected to the ESP8266 microcontroller (Fig. 4).

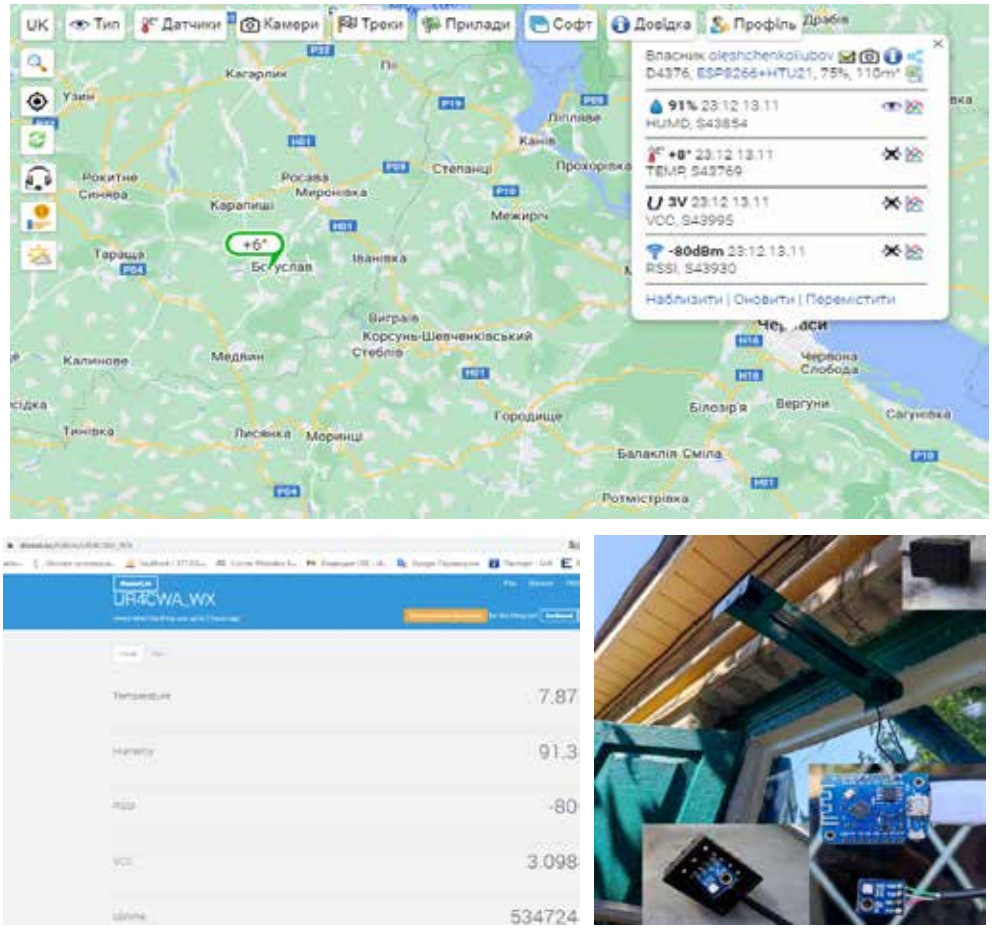

Fig. 1. Installation and connection to the Internet sensor based on ESP8266+HTU21 in the Cherkasy city [8, 9]

Fig. 2. ESP8266 controller for research

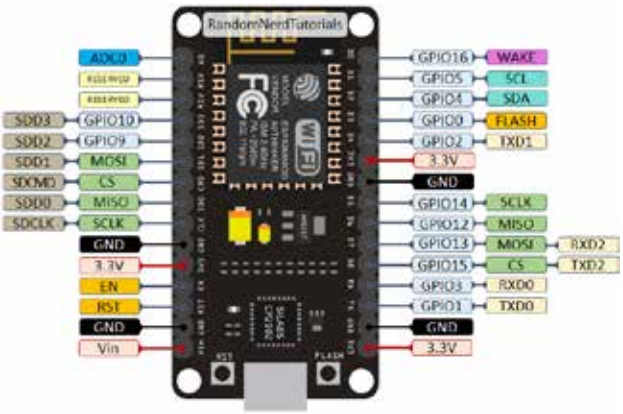

Fig. 3. Pinout of the ESP8266 controller

Fig. 4. HTU21 sensor

The connection diagram of the HTU21 sensor to the ESP8266 microcontroller is shown in Fig. 5.

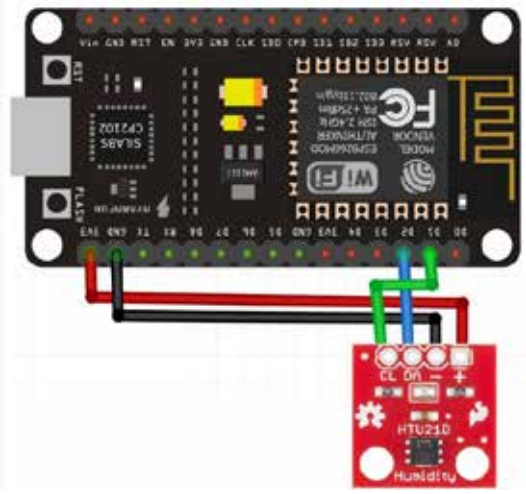

Fig. 5. Connection diagram of ESP8266 with HTU21

To analyze the sensor data, we use the data dataset D4376-20220826-20221114-1H.csv, obtained from this sensor in the period from August to mid-November 2022. Consider the volume and basic statistical information about the dataset (minimum, maximum, average value, etc.), visualize the data using the capabilities of the Python programming language. To perform this task, we use an IPython notebook in the Google Colab cloud computing environment.

Google Colab is a cloud-based tool that provides free access to computing power, memory, and GPUs to run Python software code without having to install hardware on computer. Using Colab, developers can work on CPU/GPU-intensive tasks such as machine learning and big data analytics. Colab users get free access to powerful graphics processing units (GPUs) and tensor processing units (TPUs) for their tasks. In addition, Colab includes a wide range of popular Python libraries such as NumPy, Pandas, Matplotlib, PyTorch, TensorFlow, Keras and allows to install other libraries as needed. Files can be stored on Google Drive, which allows for easy recovery from any computer and provides data backup.

Before uploading the provided dataset to Google Colab, we import the necessary Python libraries:

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
%matplotlib inline
```

We download the received dataset of indicators that were recorded by the temperature and humidity sensor. Since the file was received in CSV format, the appropriate function was used to load it into a dataframe. Since in the provided CSV file a semicolon was used as a separator instead of a comma, we specify the separator ";" in the import parameters:

```
pathToWeatherDataset = '/content/drive/MyDrive/data_colab/D4376-20220826-20221114-1H.csv'
weatherSensorDf = pd.read_csv(pathToWeatherDataset, sep=';')
```

We check the correctness of the import by displaying the first five records in the dataframe:

```
weatherSensorDf.head()
```

The content of the CSV file was correctly loaded into the dataframe. The table has seven columns: UNIXTIME, Date, Time, TEMP, HUMD, RSSI, VCC. The Date and Time columns are responsible for the date and time of taking the indicator, and duplicate the information from the UNIXTIME column, in which the date and time are presented in UNIX timestamp format. We add a new TIMESTAMP column to the dataframe, into which we copy the values from the UNIXTIME column and convert them to the Python datetime type by applying the appropriate function:

```
weatherSensorDf['TIMESTAMP'] = pd.to_datetime(weatherSensorDf['UNIXTIME'], unit='s', utc=True).dt.tz_convert('Europe/Kyiv')
weatherSensorDf
```

The new TIMESTAMP column in the dataframe now contains correct information about the date and time of recording the sensor indicators in a readable format. Now the UNIXTIME, Date, Time columns are not needed, as they duplicate the existing information in the TIMESTAMP column. Therefore, we remove these columns UNIXTIME, Date, Time from the dataframe:

```
weatherSensorDf.drop(['UNIXTIME', 'Date ', 'Time'], axis=1, inplace=True)

weatherSensorDf.head()
```

	TEMP	HUMD	RSSI	VCC	TIMESTAMP
0	20.94	69.70	-41.00	3.1	2022-08-27 22:00:00+03:00
1	21.51	62.06	-42.17	3.1	2022-08-27 23:00:00+03:00
2	20.45	68.52	-43.42	3.1	2022-08-28 00:00:00+03:00
3	20.15	69.38	-44.75	3.1	2022-08-28 01:00:00+03:00
4	19.68	70.85	-46.45	3.1	2022-08-28 02:00:00+03:00

So, five columns in the dataframe are obtained. The TEMP column indicates the outdoor temperature in degrees Celsius, the HUMD column indicates the relative humidity in percent. The RSSI column is an indicator of the power level of the Wi-Fi signal (through which the recorded data is sent) arriving at the sensor antenna, the VCC column displays the voltage of the batteries that power this sensor in volts, the TIMESTAMP column displays the date and time the sensor recorded the indicators. We find out how many records of sensor measurements are available in the data frame:

```
len(weatherSensorDf)
```

The dataset contains 1392 records. That is, the volume of the dataset is 1392 rows, and the number of columns is 5. We check the data types of each column in the dataset and make sure that there are no missing values in the rows:

```
weatherSensorDf.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1392 entries, 0 to 1391
Data columns (total 5 columns):
 # Column Non-Null Count Dtype
--- ------ -------------- -----
0 TEMP 1392 non-null float64
1 HUMD 1392 non-null float64
2 RSSI 1392 non-null float64
3 VCC 1392 non-null float64
4 TIMESTAMP 1392 non-null datetime64[ns, Europe/Kyiv]
dtypes: datetime64[ns, Europe/Kyiv](1), float64(4)
memory usage: 54.5 KB
```

Therefore, there are no missing values in the table. Therefore, there is no need to filter the dataframe to exclude empty rows. As we can see, the data type for the TIMESTAMP column is datetime, that is, the conversion was done correctly. We also learn basic statistical information: minimum, maximum, average value, etc:

```
weatherSensorDf.describe()
```

	TEMP	HUMD	RSSI	VCC
count	1392.000000	1392.000000	1392.000000	1392.000000
mean	13.461602	75.639325	-72.403247	3.097399
std	5.641691	17.412619	5.452205	0.004884
min	1.410000	23.540000	-87.830000	3.070000
25%	9.607500	64.250000	-75.250000	3.100000
50%	12.960000	79.600000	-72.500000	3.100000
75%	16.572500	90.172500	-70.080000	3.100000
max	37.370000	101.180000	-41.000000	3.100000

We see that the maximum value of the temperature is 37.37 °C, and the minimum is 1.41 °C. The average value is 13.462 °C. The same information can be viewed about other columns. In addition, in the above statistical information, we can see the value of the standard deviation, percentiles of the level of 25%, 50%, 75%. We visualize the sensor data using the capabilities of the Python language. To do this, we plot the dependence of all four measured values (TEMP, HUMD, RSSI, VCC) on the date and time of their fixation. To do this, we create a separate function that create a graph with the appropriate signatures:

```
def plot_weather_sensor_data(column_name: str, column_label: str, color: str):
    fig, ax = plt.subplots(figsize=(20, 5))
    ax.plot(weatherSensorDf["TIMESTAMP"], weatherSensorDf[column_name], f'{color}o-', markersize=3)
    plt.xticks(rotation=90)
    ax.xaxis.set_major_locator(mdates.DayLocator())
    ax.xaxis.set_minor_locator(mdates.HourLocator(interval=12))
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%d.%m'))
    plt.ylabel(column_label, fontsize=15)
    plt.xlabel('Timestamp', fontsize=15)
    plt.show()
```

We have the following four graphs that visualize the dependence temperature (°C), relative humidity (%), Wi-Fi signal power level (dBm), battery voltage (V) versus time, respectively (Fig. 6, 7, 8, 9):

```
plot_weather_sensor_data("TEMP", "Temperature, °C", "g")
plot_weather_sensor_data("HUMD", "Humidity, %", "r")
plot_weather_sensor_data("RSSI", "RSSI, dBm", "m")
plot_weather_sensor_data("VCC", "VCC, V", "b")
```

As we can see in the graphs above, there was one long period (around September 10–13, 2022) where there was no power, so the sensor was unable to capture the relevant readings at that time and send them over the Wi-Fi network.

We find the correlation between temperature and humidity parameters. To perform this task, we

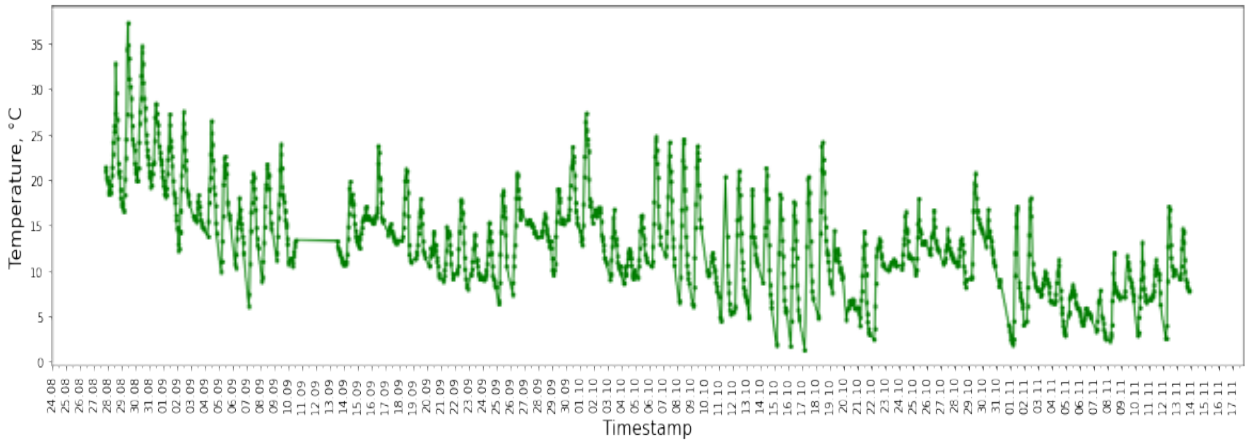

Fig. 6. Dependence of temperature (°C) on time

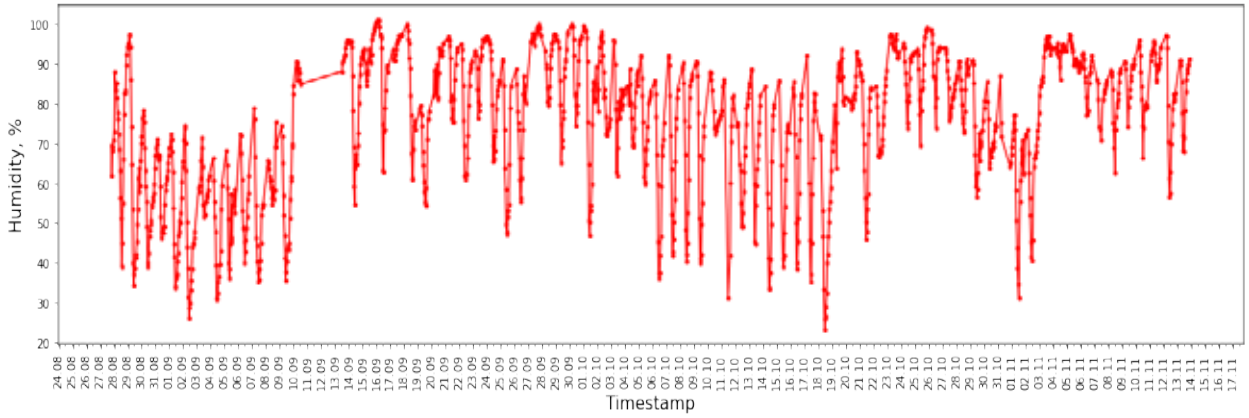

Fig. 7. Dependence of relative humidity (%) on time

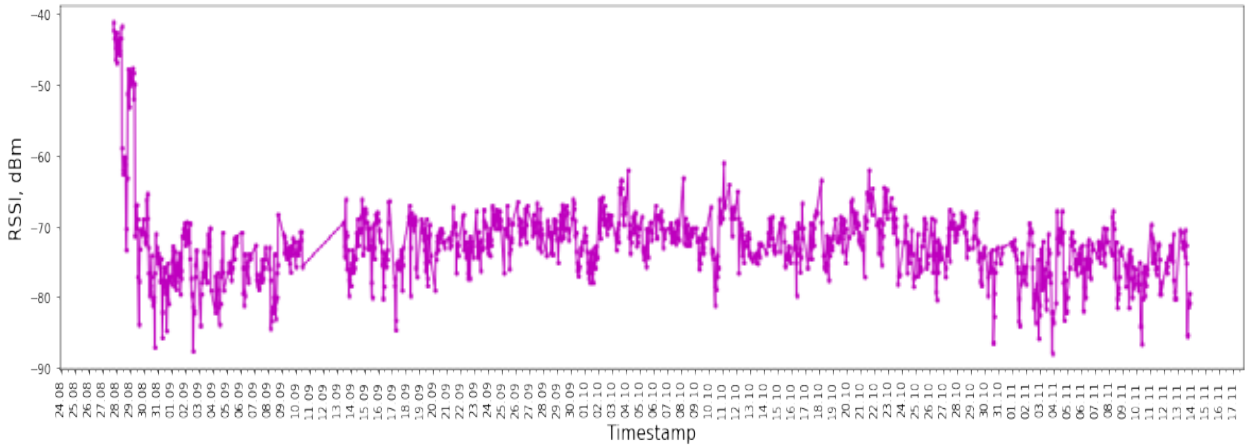

Fig. 8. Dependence of Wi-Fi signal power level (dBm) on time

again use an IPython notebook in the Google Colab environment. To build a heat map, we import the Seaborn library:

```
import seaborn as sns
```

We choose the period from 08/27/22 to 11/13/22. We build a correlation matrix for parameters: TEMP, HUMD, RSSI, VCC. To determine the correlation value, we use the Pearson method. Having received the correlation matrix, we build a heat map for it to see more clearly which variables are correlated (Fig. 10). The lighter the color, the more the two variables are correlated, and the darker the color, respectively, the opposite, that is, they are less correlated or not correlated at all:

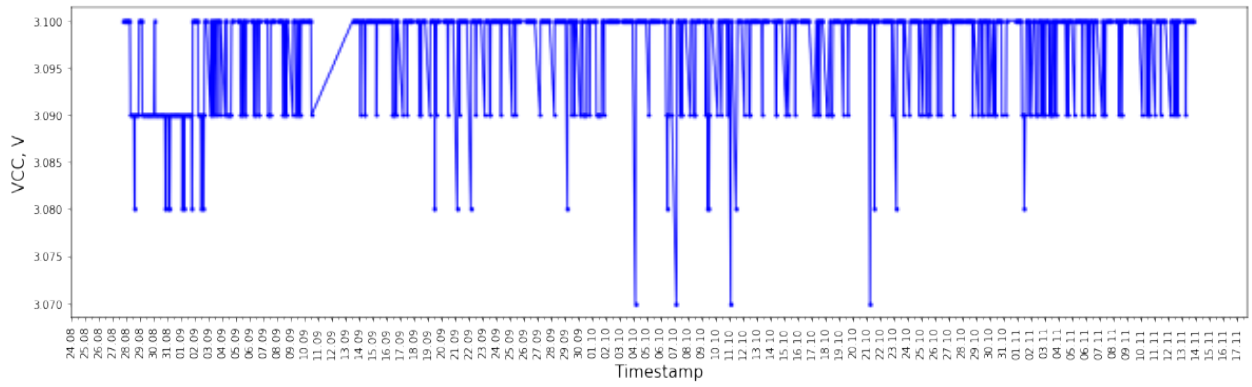

Fig. 9. Dependence of battery voltage (V) on time

	TEMP	HUMD	RSSI	VCC
TEMP	1.000000	-0.601915	0.073954	-0.432898
HUMD	-0.601915	1.000000	0.144917	0.364643
RSSI	0.073954	0.144917	1.000000	0.075421
VCC	-0.432898	0.364643	0.075421	1.000000

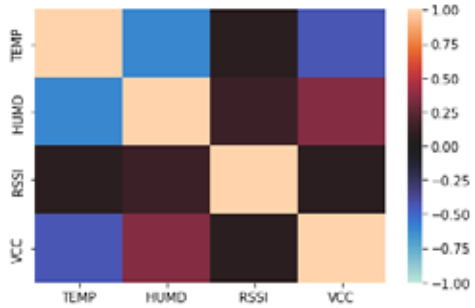

Fig. 10. Correlation matrix for parameters: TEMP, HUMD, RSSI, VCC

```
    weatherSensorDfCorr = weatherSensorDf.
corr(method='pearson')
    weatherSensorDfCorr
```

We are investigating the correlation between temperature and humidity parameters, so we consider these two parameters. It can be seen that the parameters of temperature and humidity have a correlation with a value of -0.602, that is, it is a moderately negative (negative) correlation. From this it follows that with an increase in temperature, the relative humidity of the air can sometimes decrease. There is a certain logic in this, because as the temperature increases, water evaporates more actively, so the air is drier. Also, the obtained moderate correlation can be explained by the fact that in autumn, when the temperature decreased, there was a lot of rain, as a result of which the humidity was higher. We construct a dot diagram of the dependence of relative air humidity on air temperature (Fig. 11):

```
    y = weatherSensorDf["HUMD"]
    x = weatherSensorDf["TEMP"]
    plt.figure(figsize=(20,10))
    plt.scatter(x, y)
    plt.ylabel('Humidity, %', fontsize=20)
    plt.xlabel('Temperature, °C', fontsize=20)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.show()
```

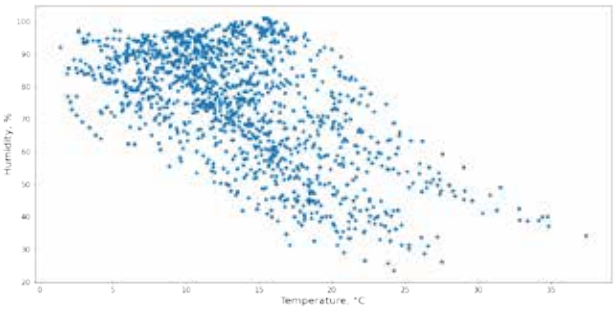

Fig. 11. The dependence of relative air humidity on air temperature

From the resulting diagram of the dependence of relative air humidity on air temperature, it can be seen that, indeed, at higher air temperatures, air humidity is often less important. We find out when emergency sensor shutdowns occurred. To perform this task, we again use the IPython notebook in the Google Colab environment. As we can see above, the sensor in normal mode sends readings once an hour. Therefore, in order to determine emergency shutdowns, it is necessary to find the largest gaps in time between fixing the sensor indicators. Since the recorded indicators of the sensor are stored in chronological order, it be enough to compare the

recording time in each pair of saved indicators in the data frame.

```
    startTimestamp = weatherSensorDf["TIMESTAMP"].min()
    endTimestamp = weatherSensorDf["TIMESTAMP"].max()
    countDays = (endTimestamp - startTimestamp).days + 1
    fig, ax = plt.subplots(figsize=(20, 5))
    n, bins, patches = ax.hist(weatherSensorDf["TIMESTAMP"],
countDays, rwidth=0.9)
    plt.xticks(rotation=90)
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.xaxis.set_major_locator(mdates.DayLocator())
    ax.xaxis.set_major_formatter(mdates.
DateFormatter('%d.%m'))
    ax.yaxis.grid()
    plt.ylabel('Number of measurements', fontsize=15)
    plt.xlabel('Day and month', fontsize=15)
    plt.ylim(0, 24)
    plt.xlim(startTimestamp, endTimestamp)
    plt.show()
```

For a visual presentation, we build a histogram, where for each day we display how many indicators were recorded.

If 24 indicators were recorded in one day, this means that the sensor worked around the clock and there were no emergency shutdowns (Fig. 12):

As we can see from the histogram above, only at the beginning of the measurements, in August, the sensor worked absolutely around the clock. Starting from September, we can see that there was no electricity for 4–5 hours on average. At the same time, we can see fairly large blackouts, there is even one the size of several days. We write a function that iterate through all recorded values in the dataframe and look for breaks in measurements of more than one hour. Next, we sort these interruptions in measurements by the largest interval in hours. We assume that emergency outages are those outages that last 10 hours or more:

```
    last_timestamp = None
    gaps = []
    for index, measure in weatherSensorDf.iterrows():
    current_timestamp = measure["TIMESTAMP"]
    if last_timestamp == None:
    last_timestamp = current_timestamp
    continue
    hours_diff = int((current_timestamp - last_timestamp).
total_seconds())//3600
    if hours_diff > 1:
    gaps.append((hours_diff, last_timestamp, current_
timestamp))
    last_timestamp = current_timestamp
    gaps.sort(key=lambda x: x[0], reverse=True)
    print("Electricity outages (>= 10 hours):")
    for row in gaps:
    if row[0] < 10:
    break
```

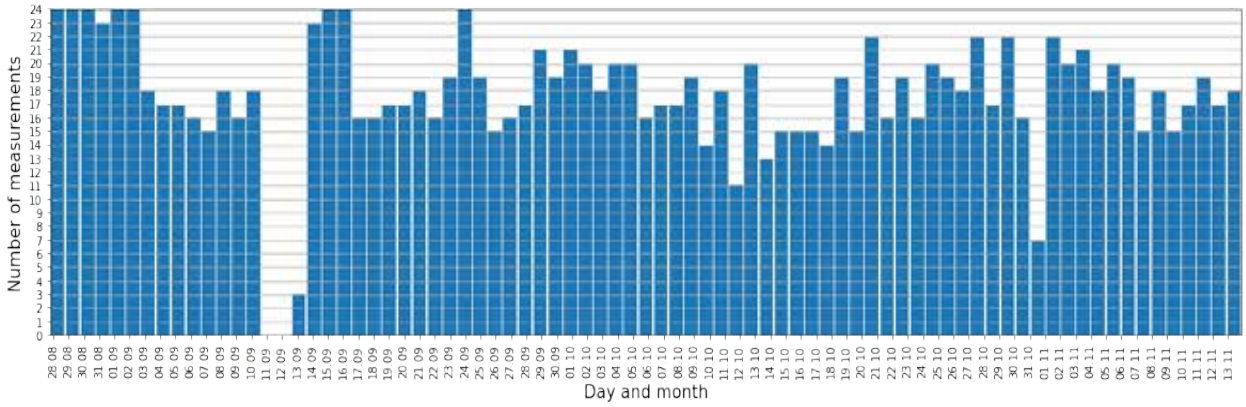

Fig. 12. Daily measurements

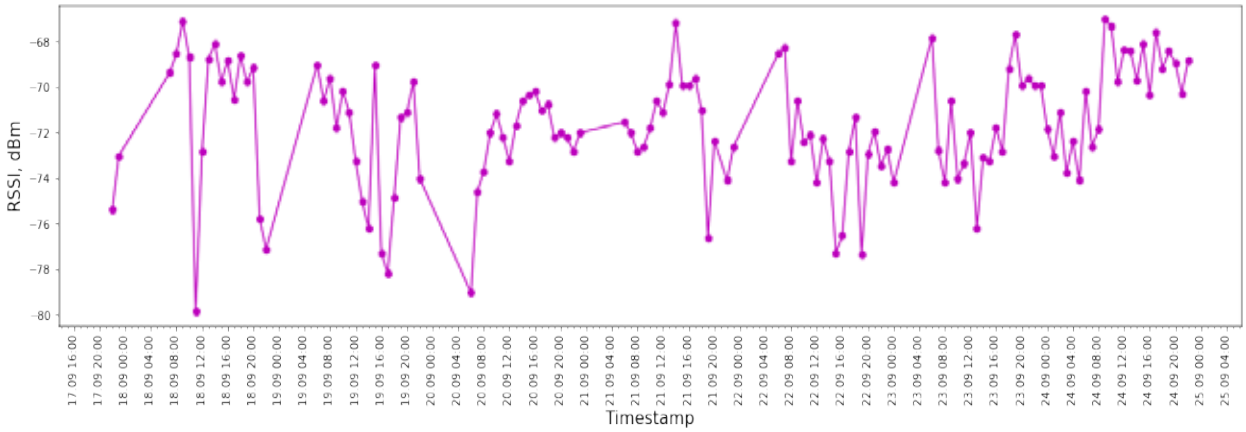

Fig. 13. The dependence of the Wi-Fi signal power level on time

```
    print(f"Duration:    {row[0]}\tFrom:    {row[1]:%d.%m
%H:%M}\tTill: {row[2]:%d.%m %H:%M}")
    Electricity outages (>= 10 hours):
    Duration: 72  From: 10.09 15:00Till: 13.09 15:00
    Duration: 15  From: 31.10 08:00Till: 31.10 23:00
    Duration: 10  From: 25.09 20:00Till: 26.09 06:00
    Duration: 10  From: 09.10 20:00Till: 10.10 06:00
```

There were a total of four emergency shutdowns that lasted 10 hours or more.

```
    72 hours without light: from 10.09 15:00 to 13.09 15:00
    15 hours without light: from 31.10 08:00 to 31.10 23:00
    10 hours without light: from 25.09 20:00 to 26.09 06:00
    10 hours without light: from 09.10 20:00 to 10.10 06:00
```

We find when there was a decrease in the signal level due to the presence of people in the room for a given period of time. To perform this task, we again use an IPython notebook in the Google Colab environment. We set the period of time in which the research should be conducted. We filter the dataset to get data for, for example, the fourth week of sensor operation:

```
    firstTimestamp = weatherSensorDf["TIMESTAMP"].min()
    week = 4
    startTimestamp = firstTimestamp +
timedelta(weeks=(week - 1))
    endTimestamp = firstTimestamp +
timedelta(weeks=week)
    weatherSensorDfFiltered = weatherSensorDf[
    (weatherSensorDf["TIMESTAMP"] >= startTimestamp) &
    (weatherSensorDf["TIMESTAMP"] <= endTimestamp)]
    weatherSensorDfFiltered
```

Therefore, 130 measurements were obtained during the fourth week of sensor operation. We visualize the filtered data from the data frame and we plot a graph of the dependence of the Wi-Fi signal power level on time in a given period of time (Fig. 13):

```
    fig, ax = plt.subplots(figsize=(20, 5))
    ax.plot(weatherSensorDfFiltered["TIMESTAMP"],
weatherSensorDfFiltered["RSSI"], 'mo-')
    plt.xticks(rotation=90)
    ax.xaxis.set_major_locator(mdates.
HourLocator(interval=4))
    ax.xaxis.set_minor_locator(mdates.HourLocator())
    ax.xaxis.set_major_formatter(mdates.
DateFormatter('%d.%m %H:%M', tz=pytz.timezone('Europe/
Kyiv')))
    plt.ylabel('RSSI, dBm', fontsize=15)
    plt.xlabel('Timestamp', fontsize=15)
    plt.show()
```

We create a function that go through all the recorded values in the data frame and look for those values of the Wi-Fi signal strength level that be less than or equal to -75 dBm. We assume that due to people in the room, the Wi-Fi signal should drop to a level of -75 dBm or below:

```
    low_signals = [] for index, measure in
weatherSensorDfFiltered.iterrows():
    current_rssi = measure["RSSI"]
    if current_rssi <= -75:
    current_timestamp = measure["TIMESTAMP"]
    next_timestamp = current_timestamp + timedelta(hours=1)
    current_row = [[current_rssi], current_timestamp, next_timestamp]
    len_low_signals = len(low_signals)
    if len_low_signals > 0 and low_signals[len_low_signals -
1][2] == current_timestamp:
    low_signals[len(low_signals) - 1][0].append(current_rssi)
    low_signals[len(low_signals) - 1][2] = next_timestamp
    else:
    low_signals.append(current_row)
    print("Lowering the Wi-Fi signal strength (RSSI <= -75):")
    for row in low_signals: rssi_mean = pd.DataFrame(row[0]).
mean()[0]
     print(f"RSSI: {rssi_mean:.2f}\tFrom: {row[1]:%d.%m
%H:%M}\tTill: {row[2]:%d.%m %H:%M}")
    Lowering the Wi-Fi signal strength (RSSI <= -75):
    RSSI: -75.33   From: 17.09 22:00  Till: 17.09 23:00
    RSSI: -79.83   From: 18.09 11:00  Till: 18.09 12:00
    RSSI: -76.42   From: 18.09 21:00  Till: 18.09 23:00
    RSSI: -75.59   From: 19.09 13:00  Till: 19.09 15:00
    RSSI: -77.71   From: 19.09 16:00  Till: 19.09 18:00
    RSSI: -79.00   From: 20.09 06:00  Till: 20.09 07:00
    RSSI: -76.58   From: 21.09 19:00  Till: 21.09 20:00
    RSSI: -76.88   From: 22.09 15:00  Till: 22.09 17:00
    RSSI: -77.33   From: 22.09 19:00  Till: 22.09 20:00
    RSSI: -76.17   From: 23.09 13:00  Till: 23.09 14:00
```

We combine the intervals in which the break in time is one or two hours, because we consider the reason for the separation of such intervals to be the measurement error. So, there were only eight people present in the room during the 4th week of operation of the device (from 09/17 22:00 to 09/24 22:00), because the Wi-Fi signal strength level dropped to -75 dBm or lower:

```
    From: 17.09 22:00 to: 17.09 23:00; average RSSI
value: -75.33 dBm
    From: 18.09 11:00 to: 18.09 12:00; average RSSI
value: -79.83 dBm
    From: 18.09 21:00 to: 18.09 23:00; average RSSI
value: -76.42 dBm
    From: 19.09 13:00 to: 19.09 18:00; average RSSI
value: -76.65 dBm
    From: 20.09 06:00 to: 20.09 07:00; average RSSI
value: -79.00 dBm
    From: 21.09 19:00 to: 21.09 20:00; average RSSI
value: -76.58 dBm
    From: 22.09 15:00 to: 22.09 20:00; average RSSI
value: -77.11 dBm
    From: 23.09 13:00 to: 23.09 14:00; average RSSI
value: -76.17 dBm
```

Conclusions and future work. The article describes the hardware and software system based on the ESP8266 controller. Software methods are used to analyze data on temperature and air humidity, as well as the signal level of the router with which the controller interacts. The obtained data were analyzed using analytical tools of the Python programming language. A correlation was found between temperature and humidity parameters, time periods of emergency shutdowns of the monitoring system due to emergency power outages, as well as time periods when there was a decrease in the signal level to account for the presence of people in the case of selected time intervals.

In the future, the authors plan to optimize the runtime when connecting and polling sensor sets to enable long battery life in offline mode. The cycle is scheduled as follows: wakeup, connect, fast connect

with settings without DHCP, DNS and other timers, check sensors, request, send data, ACK, properly prepare for sleep with port modes and stop the main core, deep sleep. The equipment is planned to be designed without interface converters, LDOs, only MCU, sensors, BMS, battery, charger, solar charger.

Bibliography:

1. Rahman, M.H., Agarwal, S., Sharma, S. et al. High-Resolution Mapping of Air Pollution in Delhi Using Detrended Kriging Model. Environ Model Assess 28, 39–54 (2023). https://doi.org/10.1007/s10666-022-09842-5
2. Ramos, R.V., Blanco, A.C. Integrated GIS and air dispersion modeling approach for roadside pollutant mapping: a case study in Baguio City, Philippines. Spat. Inf. Res. 30, 371–383 (2022). https://doi.org/10.1007/s41324-022-00438-5
3. Keswani A., Akselrod H., and. Anenberg S. C. (2022) Health and Clinical Impacts of Air Pollution and Linkages with Climate Change. NEJM Evid 2022; 1(7). DOI: 10.1056/EVIDra2200068
4. Morantes, G., González, J.C. & Rincón, G. Characterisation of particulate matter and identification of emission sources in Greater Caracas, Venezuela. Air Qual Atmos Health (2021). https://doi.org/10.1007/s11869-021-01070-2
5. Kaginalkar A., Kumar S., Gargava P., Kharkar N. and Niyogi D. (2022) SmartAirQ: A Big Data Governance Framework for Urban Air Quality Management in Smart Cities. Front. Environ. Sci. 10:785129. doi: 10.3389/fenvs.2022.785129
6. Syifaul Fuada., Hendriyana Hendriyana. UPISmartHome V.2.0 – A Consumer Product of Smart Home System with an ESP8266 as the Basis June 2022 Journal of Communications 17(7):541-552 DOI: 10.12720/jcm.17.7.541-552
7. R. Sarwansah, U. Jaelani, A. Hasad, S. Supratno, and Sugeng, "Aplikasi NodeMCU ESP8266 Untuk Monitoring Kelembaban Tanah Berbasis Internet of Things," Journal of Students Research in Computer Science, Vol. 3, No. 1, pp. 63-72, May 2022, doi: 10.31599/jsrcsv3i1.1174.
8. Public monitoring project https://eu.narodmon.com/4376
9. Dweet.io https://dweet.io/follow/UR4CWA_WX
10. UT5UUV Callsign Lookup https://www.qrz.com/db/UT5UUV

Олещенко Л.М., Мошенський А.О. АПАРАТНО-ПРОГРАМНА СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ ПОКАЗНИКІВ НАВКОЛИШНЬОГО СЕРЕДОВИЩА НА ОСНОВІ КОНТРОЛЕРА ESP8266

Апаратно-програмні системи моніторингу та аналізу показників навколишнього середовища відіграють важливу роль у зборі, аналізі даних про якість повітря, води, ґрунту та інших аспектів навколишнього середовища. Такі системи моніторингу дозволяють отримувати відомості про рівні забруднення, хімічний склад, метеорологічні умови та інші показники навколишнього середовища. Це надає можливість оцінити якість середовища, виявити проблемні зони та вжити заходів для зменшення забруднення та покращення стану довкілля. Завдяки системам моніторингу можна вчасно виявляти небезпечні рівні забруднення або хімічних речовин у середовищі. Це допомагає запобігти потенційним загрозам для здоров'я людей та прийняти такі заходи безпеки, як евакуація або застосування захисного обладнання. Апаратно-програмні системи дозволяють контролювати виконання екологічних стандартів та нормативів, встановлених органами влади, зокрема, вимоги до рівнів забруднення, викидів шкідливих речовин, а також моніторинг дотримання екологічних ліцензій та дозволів. Апаратно-програмні системи моніторингу дозволяють виявляти надзвичайні ситуації, такі як витоки хімічних речовин, пожежі або інші екологічні аварії. Це дозволяє швидко реагувати, вживати необхідних заходів для мінімізації збитків та забезпечення безпеки населення.

У статті описано побудовану авторами апаратно-програмну систему моніторингу даних навколишнього середовища та програмні методи аналізу даних температури та вологості повітря, а також рівень сигналу маршрутизатора, з яким взаємодіє контролер. Проаналізовано та візуалізовано отримані дані за допомогою аналітичних інструментів мови програмування Python. Знайдено кореляцію між параметрами температури та вологості, періоди часу аварійних відключень моніторингової системи внаслідок екстрених відключень електроенергії, а також періоди часу, коли відбувалось зниження рівня сигналу за рахунок присутності людей в приміщенні за обраний проміжок часу. Для аналізу даних використано хмарне середовище Google Colab. Описано технічні характеристики встановленої у м. Черкаси моніторингової системи на базі контролера ESP8266+HTU21.

Ключові слова: *потокові дані, Python, моніторинг стану навколишнього середовища, датчик температури та вологості повітря, датчик рівня сигналу Wi-Fi, Інтернет речей, ESP8266+HTU21.*