SYSTEM MODEL AND METHODS

Hardware components used in proposed model

We utilize one micro-controller, one bread board, one noise detecting sensor for environmental noise detection, and an Ethernet networking device to monitor the noise strength. We also used an Android cellphone to provide updates to the user. The Table 2 shows a brief functionality of all the hardware component used in our project. It described the breadboard, LM393, ESP8266, and LCD1602 module. And, the Figure 1 shows the physical view of these hardware components.

Table 2. List of Hardware components used in our Proposed System

Hardware component	Functionality
Breadboard	The board will interconnect all necessary hardware components to produce a comprehensive prototype of our proposed system.
Microphone sound detection sensor module LM393	It will assess the level of noise in the environment. The sensor output will be than converted into decibels unit.
ESP8266 NODEMCU-12E	It will work as the primary micro-controller unit. Using this MCU, we send sensor data to the cloud which can be utilized to make noise intensity predictions.
LCD1602 I2C display	It will show the noise intensity level and the status on its screen.

Explaining hardware in details

Microphone sound detection sensor module

It has a fantastic dynamic microphone that measures the sound level from the source. The sound sensor which showed in Figure 1(a) is a compact circuit board with a microphone (50Hz-10kHz) and circuit connections that convert sound waves to electrical pulses. The LM393, a comparator IC, receives this electrical pulse. The signal is digitized by the LM393 IC, which is coupled to the OUT pin. A variable resistor on this noise sensor controls the intensity of the OUT output. OUT, VCC, and GND are the three pins on it.

ESP8266 NODEMCU-12E

The NodeMCU ESP8266 which showed in Figure 1(b) comes with the ESP-12E module, which has the ESP8266 chip and a Tensilica Xtensa 32-bit LX106 RISC microprocessor. This CPU supports RTOS and runs at a clock frequency of 80MHz to 160MHz. The NodeMCU has 128 KB of RAM and 4 MB of Flash memory to store data and programs. Because of its high computing power and built-in Wi-Fi or Bluetooth, it's ideal for IoT projects. The primary component of this system is the NodeMCU ESP8266 12E, which maintains the Sensor Module LM393 and LCD display by connecting the pins of all of these components together and keeping the system procedure smooth and maintainable. The ESP8266 is a fantastic component that allows you to keep the entire process on a single breadboard, logically connected with the relevant pins via wired connections.

LCD1602 I2C display

LCD1602 I2C Display which showed in Figure 1(c) is a 16x2 LCD display screen with I2C interface. The characters in this component are white on a blue background and may display 16x2 characters on two lines. The wire soldering and connection is also somewhat difficult.

(a) (b) (c)

Figure 1. Hardware components used in the system (a) microphone sound detection sensor module, (b) ESP8266 NODEMCU-12E, and (c) LC1602 I2C display

IMPLEMENTATION OF THE SYSTEM

Our system was built using the LM393 sound sensor module and the ESP8266. To finish the connection, the GND of the Sensor Module is linked to the GND of the ESP8266. In addition, the VCC and OUT pins of the sensor module are connected to 3V3 and A0 on the ESP8266, respectively. The LCD1602 I2C Display GND pin, on the other hand, is connected to the ESP8266's GND pin. The LCD's VCC is also connected to the ESP8266's VIN. The LCD display's SDA and SCL pins are connected to the ESP8266's D2 and D1 pins, correspondingly. The block diagram is shown in the Figure 2. The pin connections between the three hardware components, ESP8266 NODEMCU-12E, microphone sound detection sensor module LM393 and LCD1602 I2C display respectively are shown in Table 3 as follows.

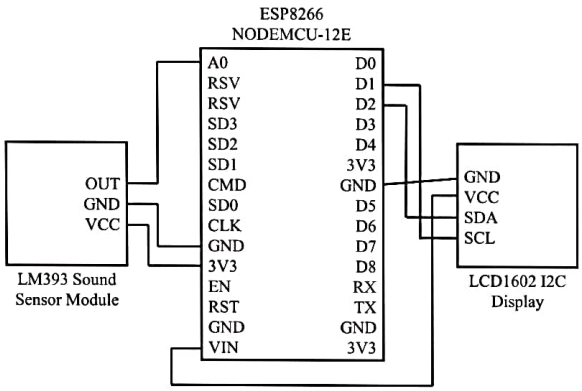


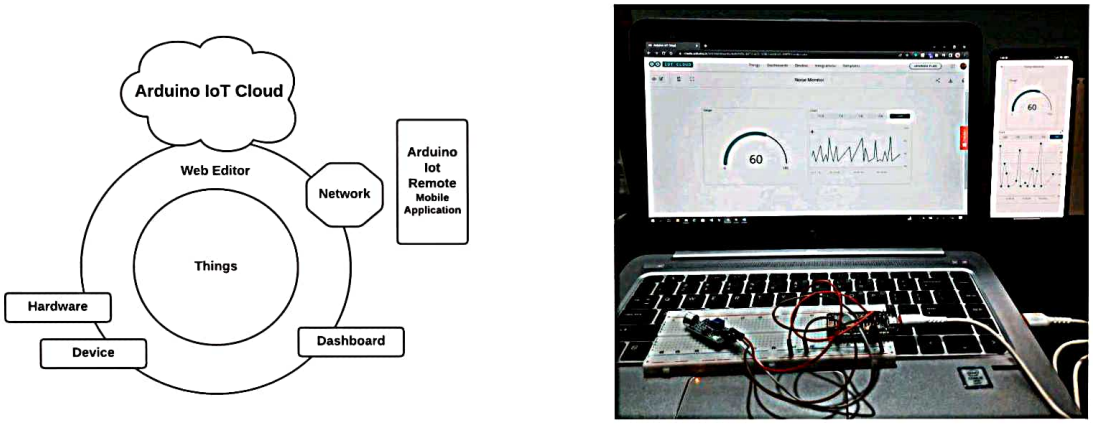
Figure 2. Block diagram of the proposed system

Table 3. Pin connection of NodeMCU-12E and LM393

ESP8266 NODEMCU-12E pin	Microphone sound detection sensor module LM939 Pin	LCD1602 I2C display pin
3V3	VCC	-
GND	GND	GND
A0	OUT	-
VIN	-	VCC
D1	-	SCL
D2	-	SDA

The Arduino IoT cloud must be connected to the hardware system. We must finish all of the settings in accordance with our system policy, such as setting the variable to decibel and collecting the ethernet to that cloud, after successfully logging into the software. Finally, we must set the board's NodeMCU 1.0 (ESP-12E module) and port configurations. Before connecting our hardware configuration, we must additionally launch the Arduino agent from our PC. We must make alterations to the in the flavors area, immediately beside the boards, by selecting V2 IPV6 higher bandwidth and all flash content for further work. We must code and send our work to the program and the board after we have completed all of the steps. After we finish uploading, we will get the real-time data we need from the environment.

On the other hand, for the mobile application, go to the Google Play Store and download the Arduino IoT remote app. We must first log into the application using the provided information, or we can use Google to do it. We must configure the program after logging in by connecting to the server, which occurs automatically. When a user successfully logs into the application, data from that environment is retrieved, and the user's dashboard displays the noise intensity level in that environment. Figure 3 depicts the whole architectural layout and setup of our proposed system where Figure 3(a) is the architectural diagram and Figure 3(b) is the hardware setup of the system.



(a) (b)

Figure 3. The design of the proposed system (a) architectural diagram and (b) the setup of the system

The suggested system's pseudo code is also shown as follows. To begin, we set up all of the variables we will require throughout the program using code. To get the single at the right spot, we set the signal maximum value to 0 and the signal minimum value to 1024. In order to receive the data, this will be reversed. The while loop was then utilized to evaluate the real signal's state.

The analog reading is set to 0 by default. The first if condition checks whether the sample in our input signal is smaller than 1024, if it is, it will go inside the loop and check the other condition, comparing the value sample to the maximum value, if it is, the value will be stored in the signal maximum variable, otherwise it will check the other condition, eventually storing the value in the signal minimum variable. After confirming all of the conditions, we will calculate the difference between the maximum and minimum signal. We save it in the *peakToPeak* variable. Then use calibration to transfer the value to db that is in decibel, the real variable.

```
peakToPeak = 0, signalMax = 0, signalMin = 1024
while (millis()- millis() < sampleWindow)
    sample = analogRead(0)
    if (sample < 1024)
        if (sample > signalMax) signalMax = sample
        else if (sample < signalMin) signalMin = sample
peakToPeak = signalMax - signalMin
db = map(peakToPeak, 20, 900, 49.5, 90)
```

To show the noise intensity level on the LCD screen, we used LiquidCrystal_I2C library for the I2C LCD screen module. We show the decibel value of the noise intensity in the first row of the LCD screen. And, in the second row of the LCD screen, we show the noise intensity status. The LCD works then proceed by initializing the values after doing so in code. As stated in the code, the cursor is set to (0,0). Then a print statement is given, which will print the output display when the system starts working according to the conditions set. After that, it will check the condition and display a *QUITE* message in the LCD display if the decibel value is less than or equal to 60 decibels. Again, if the decibel condition is more than or equal to 85 decibels, the output in display will be *HIGH*.

However, if both conditions are not met, it will output *MODERATE*, which is in the midst of the two above values. The moderate condition is defined as a decibel level of louder than 60 decibels but less than 85 decibels. Only if both conditions are true and met will it enter the loop. Otherwise, it will check the other condition. This status is showed in the right alignment of the second row of the screen. After delaying data collection for 1,500 milliseconds, new data is being collected.

```
lcd.setCursor(0, 0)
lcd.print("Noise: " + db + " dB")
if (db <= 60) lcd.print("QUITE")
else if (db > 60 && db < 85) lcd.print("MODERATE")
else if (db <= 85) lcd.print("HIGH")
delay(1500)
```

To show the noise intensity level on the LCD screen, we used *LiquidCrystalLiquidCrystal_I2C* library for the I2C LCD screen module. We show the decibel value of the noise intensity in the first row of the LCD screen. And, in the second row of the LCD screen, we show the noise intensity status. The LCD works then proceed by setting the cursor to (0,0). Then a print statement is given, which will print the output display when the system starts working according to the conditions set. After that, it will check the condition and display a *QUITE* message in the LCD display if the decibel value is less than or equal to 60 decibels. Again, if the decibel condition is more than or equal to 85 decibels, the output in display will be *HIGH*. However, if both conditions are not met, it will output *MODERATE*, which is in the midst of the two above values. The moderate condition is defined as a decibel level of louder than 60 decibels but less than 85 decibels. After delaying data collection for 1,500 milliseconds, new data is being collected.

Web data parsing

Our proposed system also works extremely well for web applications. To do so, a user must first register for the Arduino IoT Cloud software, or if they already have an account, they must login. After successfully

logging into the program, we must complete all of the settings in accordance with our system policy, such as setting the variable to decibel and collecting the ethernet to that cloud. Finally, we must configure the board to NodeMCU 1.0 (ESP-12E Module) and the port. We must also launch the Arduino agent from our PC before connecting our hardware configuration. In the flavors section, immediately beside the boards, we must make adjustments to the by selecting V2 IPV6 higher bandwidth and all flash content for further use. After completing all of the procedures, we must code and send our work to the software and the board. We will acquire our desired real-time data from the environment after we finish uploading.

Mobile data parsing

The technology collects data from the environment and displays it on a mobile app available on the Google Play Store. We must first log into the application using the specified credentials, or we can do it using Google. After logging in, we must configure the application by connecting to the server, which happens automatically. When a user successfully logs into the application, data from that environment begins to be retrieved, and the noise intensity level in that environment is displayed on the user's dashboard.

RESULTS AND ANALYSIS

For the result and analysis section, we did our utmost to make the system run as smoothly as possible. Our goal was to achieve the highest possible level of success with our proposed project. We have tested our system in a variety of ways in order to preserve this. For instance, we might make a loud noise to see if our system is working properly. As a result, it produces the ideal result. On the other hand, by reducing noise intensity in a certain location, we were able to determine whether the sound level gap is acceptable to the system. The system also displays the expected outcome on this base station.

The output of the sensor reading shows in the LCD display and Android IoT Cloud dashboard; the output shows in the Figure 4. Regarding better results and visualizations, we utilized an LCD display to indicate the noise in a certain area, as well as the amount of noise mode, to make things more exact and understandable. For instance, the image of the LCD display in Figure 4(a) demonstrates the noise level in decibels and a sound level that is $QUITE$ for that particular location.

To display the result in the dashboard, we have used a graph chart in relation to the data verses time after presenting the result in the gauge meter. The chart will primarily show the noise intensity level for various types of environments. Additionally, we used a gauge meter and labeled it with the variable decibel in Figure 4(b). So that the decibel output of the displayed noise intensity equals decibel.

In the Figure 4(c), the mobile app also displays the fine result of the noise intensity level, and the dashboard is set up as a gauge meter as well as a graph to make it easy to grasp. Because everyone has a smartphone in this highly technological age, having a cloud-based application that provides information about noise levels in a given area could be beneficial to the general public.

(a) (b) (c)

Figure 4. Output of the sensor reading (a) sensor reading and intensity status on the LCD module, (b) graph and gauge meter, and (c) mobile dashboard

We gathered real-time data for a week, from Sunday to Saturday. And, as shown in Figure 5, the average success rate of data collecting according to the day of the week. We've just calculated the average of the entire day's data to see if the system is operating as expected. The values ranged from 49 decibels to 86 decibels, with 86 decibels being the highest. On Saturday and Sunday, the noise level in the house was very low. On working days, which were Monday to Thursday, the values ranged from 61 to 72 decibels, respectively. We went out on Friday in multiple places, many of which were crowded, and because it was a holiday in our country, the average range was 86 decibels. All of this information was gathered at random in order to get a sense of the system's performance.

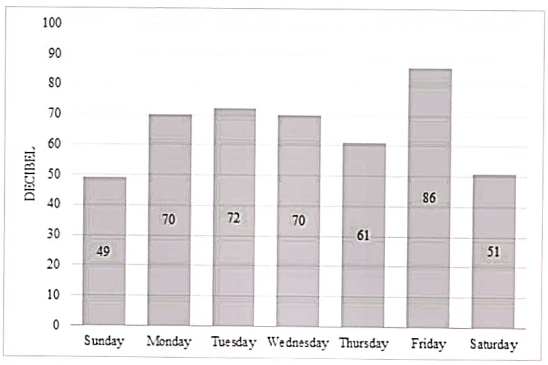


Figure 5. Graph of sound intensity levels averaged for each day of the week