Home High-Volume Sound Detector

Overview

In this project, I developed an IoT system that collects, stores, and displays sound data. At home, I used a Pico W microcontroller along with a sound sensor to capture the sound levels. A red LED is also included to indicate when the surrounding sound volume is high. The system connects to Wi-Fi to send the data to Adafruit, where it is stored and can be monitored. The entire project takes about 2 to 4 hours to complete.

Objective

The idea of collecting sound volume data interests me because in Sweden and other European countries, there are regulations concerning noise levels in residential areas, especially for rented apartments. These laws are designed to ensure that tenants do not disturb their neighbours with excessive noise. This IoT system can be utilized by property management companies to provide evidence of sound levels without infringing on privacy, as it only records the volume, not the content of the sounds.

Additionally, the system can be connected to devices like TVs to manage sound levels automatically. For example, if the TV's volume is too high, the IoT system can automatically lower it and trigger a red LED to alert the occupants of the apartment that the noise level is excessive. This can be especially useful for families with children, helping to protect their hearing and maintain a peaceful environment, while also ensuring they don't disturb the neighbourhood.

for the. People who lives in houses to control the sound volume for their children and. Make sure that they don't hurt their ears from high sounds at home. Beside the they don't bother the neighbourhood

Material

Component	Price in SEK	Pics
Raspberry Pi Pico W	109	

Sound sensor KY-038	36	
Red LED	5	
Resistors	<10	
Jumper wires	<34	

Computer Setup

For the computer setup, you'll need the Visual Studio Code (VS Code) IDE, along with the MicroPython and CircuitPython libraries. To get started with CircuitPython, you'll need to download the CircuitPython firmware file and copy it to your Pico W microcontroller. Once copied, the Pico W will automatically generate the necessary code framework. You can then write your code in the main Python file. CircuitPython is particularly useful for connecting the Pico W to Wi-Fi, which is the method used in this project (no LoRa is needed).

Additionally, you'll need an Adafruit account to store and retrieve data. JavaScript, along with charting libraries, can be used to present the data visually. CircuitPython also allows you to store sensitive information in a file called settings.toml. There is also a lib folder where you can add essential libraries, especially those required for Adafruit.

Putting Everything Together

There was an issue with the sound sensor where it required 5 volts from the Pico W's VBUS, but the Pico W can only handle a maximum of 3.3 volts. To address this, I needed to use resistors to reduce the voltage going to the Pico W. The KY-038 sound sensor has two outputs: one digital and one analog. The digital output can be controlled without resistors and can be connected to any general-purpose (GP) pin. However, for the analog output, resistors are necessary to lower the voltage to a safe level for the Pico W.

The Pico W has three GP pins (GP28, GP27, and GP26) that support analog input as ADC (Analog-to-Digital Converter) inputs. The KY-038 sensor also includes an LED that lights up when sound is detected, and its sensitivity can be adjusted to respond to specific sound volumes.

To visualize the circuit using Wokwi, I couldn't find an exact match for the sound sensor, so I used a similar component just to demonstrate how the circuit should look.
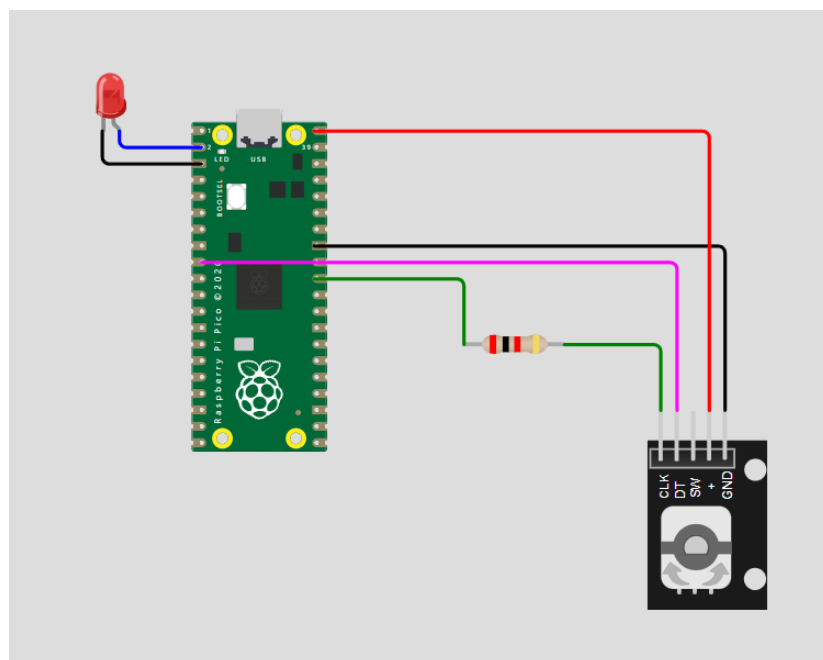
Platform

This project utilizes Wi-Fi, which is widely available in most Swedish homes. While the connection speed might be low, Wi-Fi is generally sufficient for this application. The system can also be further developed to support LoRa (Long Range) communication for areas with limited Wi-Fi access.

The Pico W microcontroller sends the data to Adafruit, where it is stored. Companies can then retrieve this data using JavaScript, along with various charting libraries, to present the information in a clear and effective manner. CircuitPython simplifies the process of connecting the Pico W to Wi-Fi and Adafruit, making the entire setup more user-friendly.

The code

To use CircuitPython for this project, follow these steps:

1. Download the .UF2 File: Obtain the CircuitPython .UF2 file for the Pico W.

2. Transfer the File: Plug the Pico W into your computer and copy the .UF2 file onto the Pico W. This process will automatically generate the necessary framework, including several files and folders.

3. Setup the Environment:

   I. lib Folder: Add the required libraries here for your project. For this project, I included the following libraries:
   - adafruit_bus_device
   - adafruit_io
   - adafruit_minimqtt
   - adafruit_register
   - adafruit_ahtx0.mpy
   - adafruit_connection_manager.mpy
   - adafruit_requests.mpy
   - adafruit_ticks.mpy

4. Configure Sensitive Information: Use the settings.toml file to store sensitive information such as Wi-Fi passwords and Adafruit API keys.

5. Write Your Code: Implement your project logic in the code.py file:

- **Connect to Wi-Fi**

```
import ssl
import wifi
import os

wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'), os.getenv('CIRCUITPY_WIFI_PASSWORD'))
```

- **Connect to Adafruit**

```
import adafruit_requests
import adafruit_dht
from adafruit_io.adafruit_io import IO_HTTP, AdafruitIO_RequestError
import socketpool

aio_username = os.getenv('aio_username')
aio_key = os.getenv('aio_key')

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())

io = IO_HTTP(aio_username, aio_key, requests)
print("Connected to Adafruit IO")


try:
    soundAnalog_feed = io.get_feed("soundanalog")
    soundDigital_feed = io.get_feed("sounddigital")
except AdafruitIO_RequestError:
    soundAnalog_feed = io.create_new_feed("soundanalog")
    soundDigital_feed = io.create_new_feed("sounddigital")


feed_names = [soundAnalog_feed, soundDigital_feed]
print("Feeds created")
```

- **Define sensor input from the sound sensor**

```
import analogio
import digitalio
import board

sound_analog = analogio.AnalogIn(board.GP26)
sound_digital = digitalio.DigitalInOut(board.GP6)
sound_digital.direction = digitalio.Direction.INPUT
```

- **The main code of getting the data and send it.**

```python
import time
import microcontroller


clock = 10

def read_analog(pin):
    return (pin.value * 3.3) / 65536

while True:
    try:
        if clock > 10:

            sound_analog_value = read_analog(sound_analog)
            sound_digital_value = sound_digital.value

            if sound_analog_value is not None and sound_digital_value is not None:
                data = [ sound_analog_value, sound_digital_value]
                for z in range(4):
                    io.send_data(feed_names[z]["key"], data[z])
                    print(f"Sent {data[z]:0.2f} to feed '{feed_names[z]['name']}'")
                    time.sleep(1)

                print(f"Sound Analog: {sound_analog_value:0.2f} V")
                print(f"Sound Digital: {sound_digital_value}")
                print()
            else:
                print("Failed to retrieve data from sensor")

            clock = 0
        else:
            clock += 1
    except Exception as e:
        print("Error:\n", str(e))
        print("Resetting microcontroller in 10 seconds")
        time.sleep(10)
        microcontroller.reset()

    time.sleep(1)
    print(clock)
```
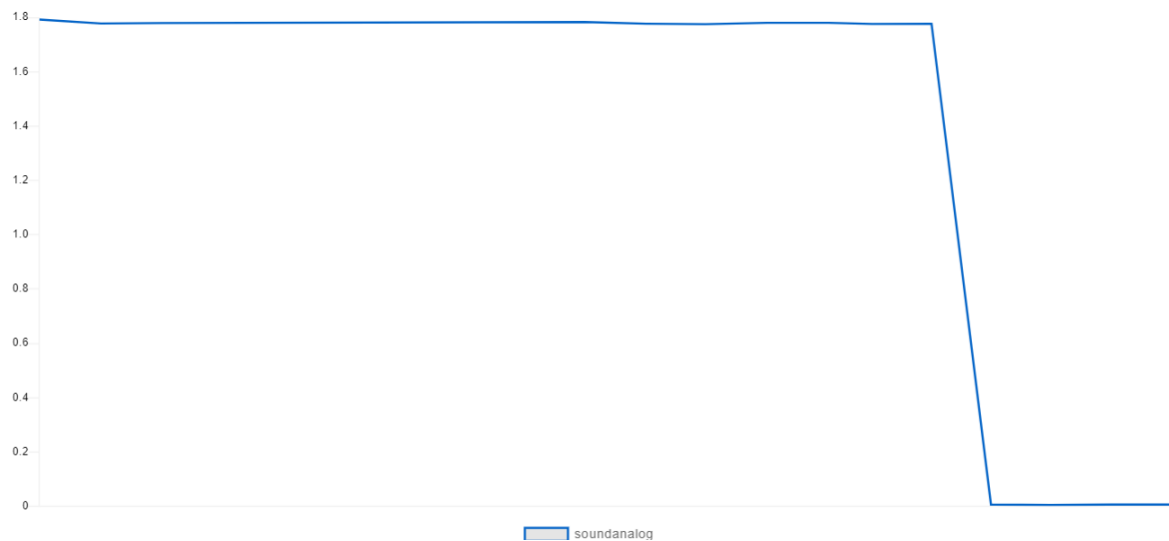
Presenting The Data

Adafruit offers excellent data presentation tools, including graphs and data visualizations. Although I initially planned to develop a JavaScript application to present the data from Adafruit, time constraints prevented me from completing this step. However, it is straightforward for companies interested in using this data to integrate it into their own applications. Adafruit provides an API that delivers data in JSON format, making it easy to connect this data to JavaScript web applications for seamless integration and visualization.

Finalizing the design

The design can be significantly improved by enclosing the electronic components and cables in a compact box. Similar to existing devices used for monitoring humidity and temperature and controlling heating automatically, this system could also be housed in such enclosures. Integrating the sound detection system into these existing boxes would streamline its appearance and functionality, making it more practical and aesthetically pleasing for home use.