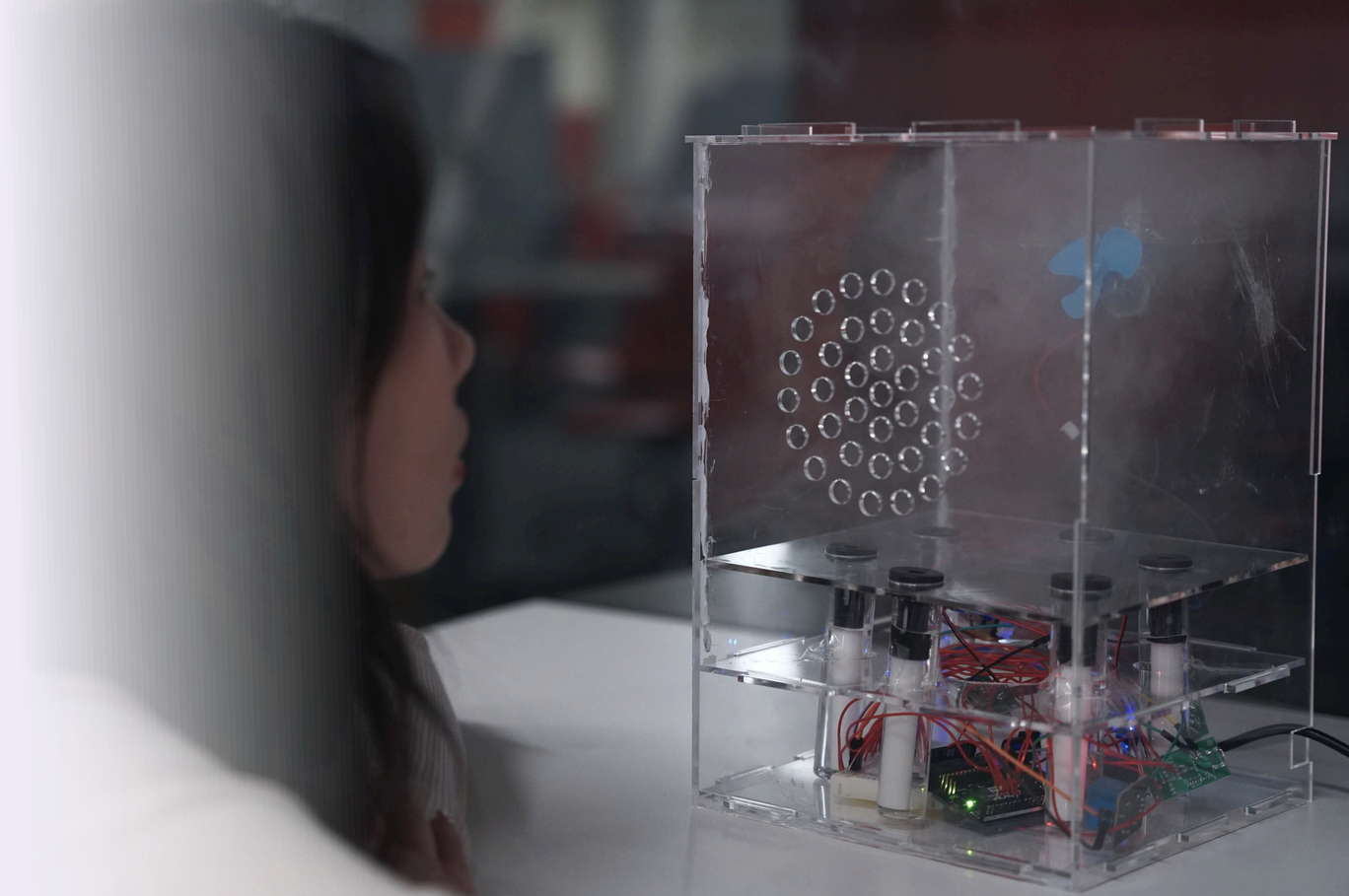
SMELL THE SOUND

Interaction Design & Installation Art &
Audio Experience Design & UX Design

Smell the Sound is an interactive installation that transforms spoken voice into corresponding scents, simulating auditory-olfactory synesthesia. By analyzing speech characteristics such as frequency, envelope, pause, and harmonics, the system maps voice features to specific scents, allowing users to "smell" their own voice. The installation integrates Max/MSP for real-time sound analysis, Arduino for scent diffusion control, and Python for data communication, creating a unique sensory experience.

Personal Project: 2025.02 - 2025.03

My Role : MAX/MSP, Python, Integration



Content

Overview	1
Research & Design Aims	2
Design & Concept	3
Build Design	4
MAX/MSP Patch	5
Python	9
Arduino	10
Components	10
Assembly	10
Display	11
Workflow	12
User Test & Feedback	12
Conclusion	13
Future Development	14
Reference	15

Huiqu Lei

Audio and Smell Mapping
MAX/MSP Harmonic Analysis
Enclosure design
User testing

Jie Wang

MAX/MSP Envelope Analysis
Arduino
Python (to Arduino)
MAX/MSP Visualisation (attempted)

Yushan Xu

MAX/MSP Frequency Analysis
MAX/MSP Pause Analysis
Python (from MAX/MSP)
Max/MSP Integration

Can You Smell Your Voice?

Smell the Sound is an interactive installation that transforms spoken voice into corresponding scents, simulating auditory-olfactory synesthesia. By analyzing speech characteristics such as frequency, envelope, pause, and harmonics, the system maps voice features to specific scents, allowing users to "smell" their own voice. The installation integrates Max/MSP for real-time sound analysis, Arduino for scent diffusion control, and Python for data communication, creating a unique sensory experience.

RESEARCH

What is Synesthesia?

Synesthesia is a neurological condition in which stimulation of one sense automatically triggers a perception in another. For example, some individuals may see colors when hearing music or associate specific scents with sounds[1] .

Why Simulate Auditory-Olfactory Synesthesia?

Auditory-olfactory synesthesia, where sounds consistently evoke specific smells, is extremely rare—reported in only about 1.58% of synesthetes[8]. This rarity makes it difficult for non-synesthetes to understand or experience such cross-sensory perception naturally.

"The sound of stepping on branches in the forest smells like cinnamon and French toast."[2]

"I can smell the music most of the time, but a lot of the smells are either very strong and citrusy or like nature. When I'm in band class, all of us smell salty like the ocean."[2]

Auditory and Olfactory Cross-modality

The senses of smell and hearing share some of the neural processing mechanisms in the brain.
About 19% of neurons in the olfactory bulb nucleus respond significantly to tonal stimuli, suggesting that these neurons not only respond to odours but also process sounds.
About 29% of neurons show either supra-additive enhancement or inhibitory effects when odours and tones are present simultaneously, suggesting that auditory information modulates the perception of odours[3] .

DESIGN AIMS

Enabling Non-Synesthetes to Experience "Synesthesia"

The proportion of synesthetes in the global population is approximately 4% to 5%, and auditory-olfactory synesthesia is extremely rare—reported in only about 1.58% of synesthetes[8].
By analyzing voice characteristics (e.g., pitch, loudness, speech rate, HNR) and mapping them to corresponding scents, the system enables users to smell their own voice, offering a unique cross-sensory perception.

Redefining the Perception of Sound

Sound is typically an auditory stimulus, but this device transforms it into an olfactory experience, expanding the way we understand sound.
Using scent as a feedback channel, rather than the usual visual or tactile methods, creates a non-traditional sensory mapping.

Enabling Users to "Smell" Their Own Voice

Human voices carry unique characteristics (tone, emotion, speed), and this device translates those features into personalized scent feedback. This installation can help users reflect on their voice in a novel way, experiencing self-expression through scent.

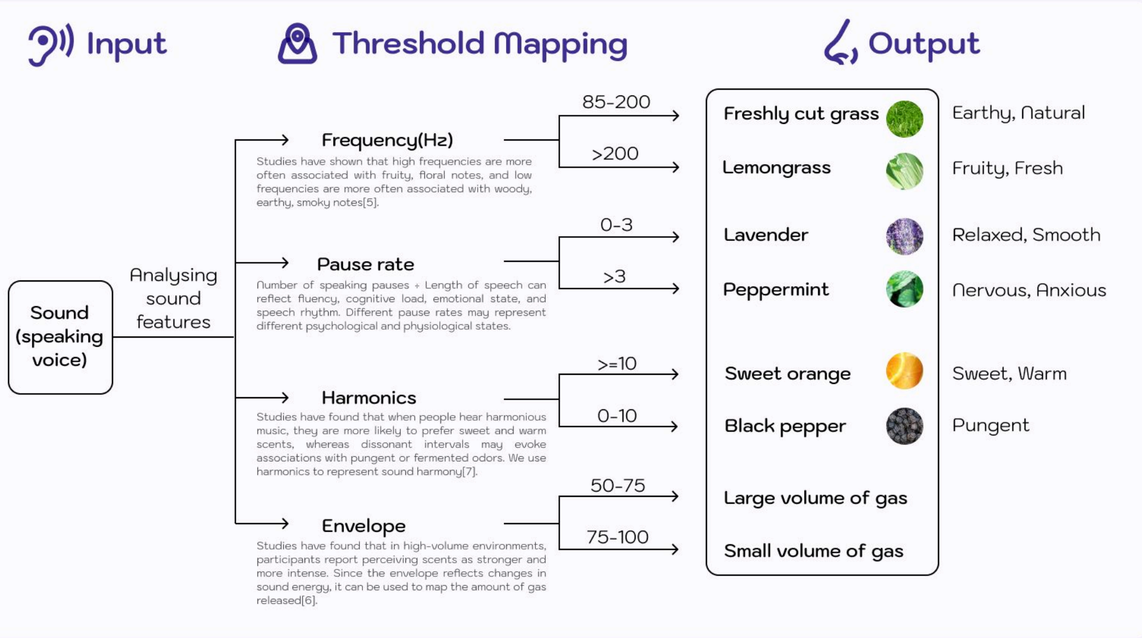
Enhancing Multisensory Interaction

Modern interactive design emphasizes multisensory experiences, but most human-computer interactions are still vision and sound-based.
Olfaction is often overlooked, yet it plays a crucial role in emotions, memory, and immersion.
Our device fuses sound and scent, offering new possibilities for gaming, art, therapy, and education.

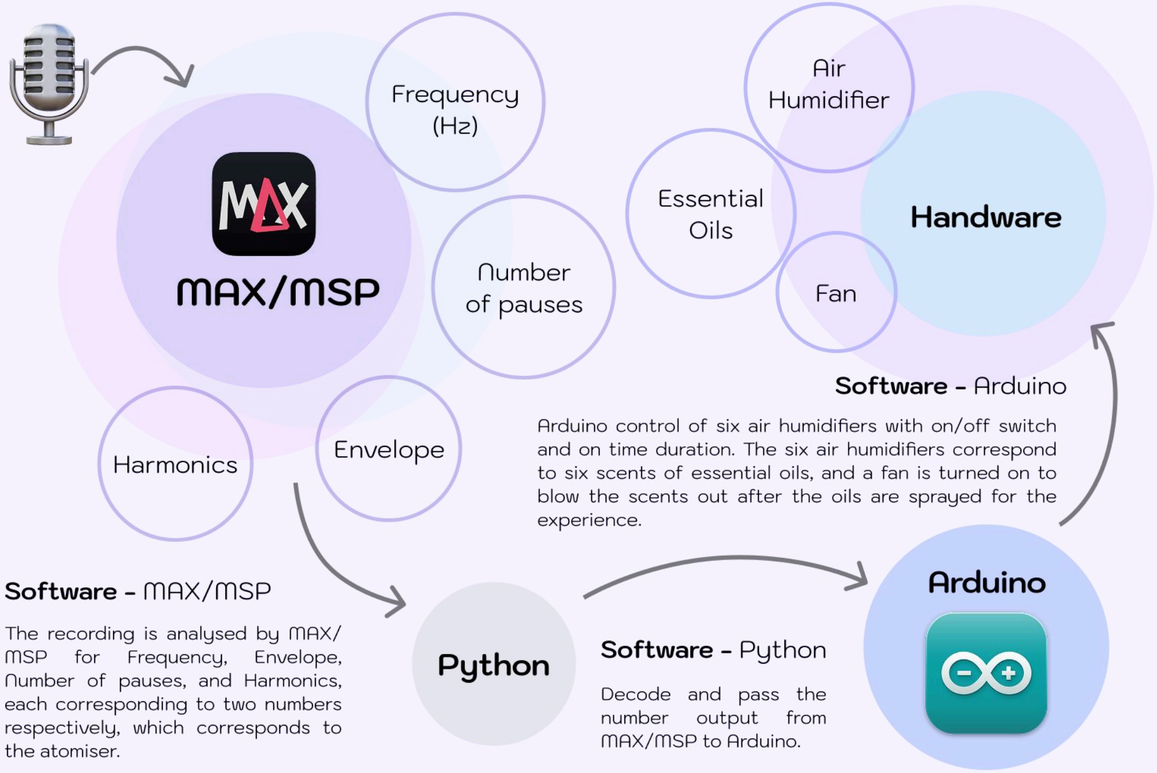
DESIGN & CONCEPT

How Does the Installation Create This Experience?

- The system analyzes voice characteristics (e.g., frequency, pause, harmonics, envelope).
- It then maps these auditory features to specific scents, generating a corresponding olfactory output.
- By allowing users to hear their voice while simultaneously experiencing its scent representation, the device provides a simulated synesthetic experience, offering a new way to perceive sound through smell.

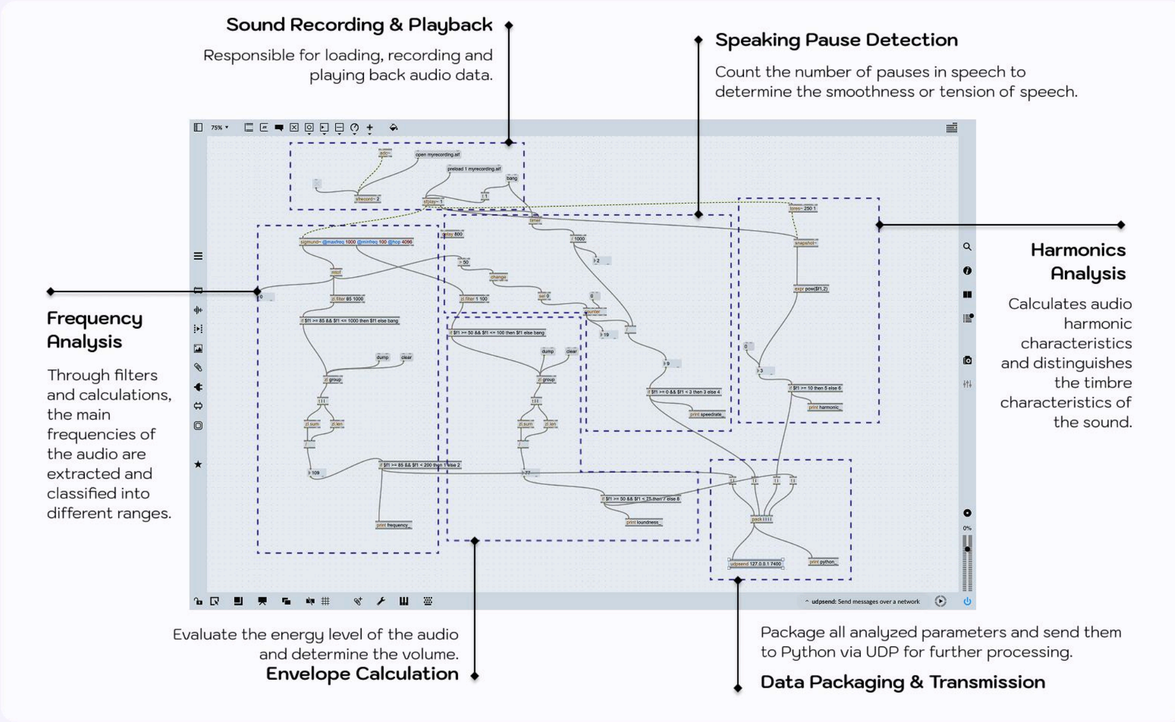


BUILD DESIGN



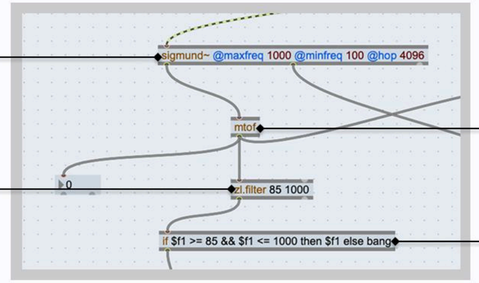
MAX/MSP PATCH

A patch was made to analyze voice characteristics in Max, extracting frequency, speech rate, harmonics, and loudness. The processed signals were then sent to Python for final compilation before being transmitted to Arduino, which controlled the atomizers and fans to generate corresponding scents.



MAX/MSP PATCH Frequency+Speaking Pause

Frequency



sigmund~
extracts the fundamental frequency of the incoming voice signal, analyzing its pitch within a specified range (100 Hz - 1000 Hz).

zl.filter 85 1000
ensures that only frequencies between 85 Hz and 1000 Hz are considered, filtering out irrelevant data.

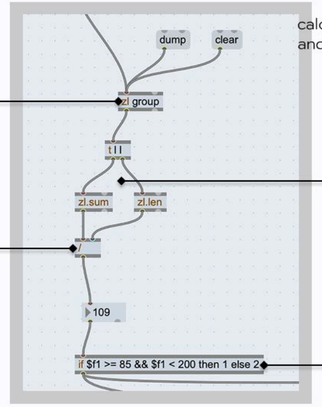
converts the MIDI note values from the frequency analysis into standard Hz values.

mtof

checks whether the frequency falls within the valid range and either outputs the value or triggers a bang if it is outside the range.

if statement

Averaging & Classification



zl.group
collects incoming values into a list, allowing batch processing of multiple data points.

division (/)
computes the average value of the grouped data.

calculate the sum and length of the stored values, preparing for averaging.

zl.sum & zl.len

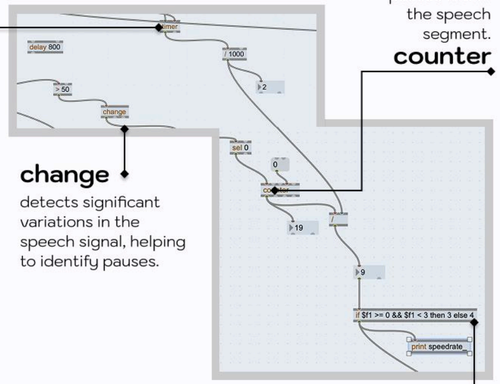
checks if the computed average falls within the 85–200 Hz range. If so, it assigns classification "1"; otherwise, it assigns "2".

if statement

Speaking Pause

measures the time interval between detected speech events to determine pause durations.

timer



keeps track of the total number of pauses within the speech segment.

counter

change
detects significant variations in the speech signal, helping to identify pauses.

classifies the speech rate based on the number of pauses, determining whether the speaker's pace is normal or slow.

if statement

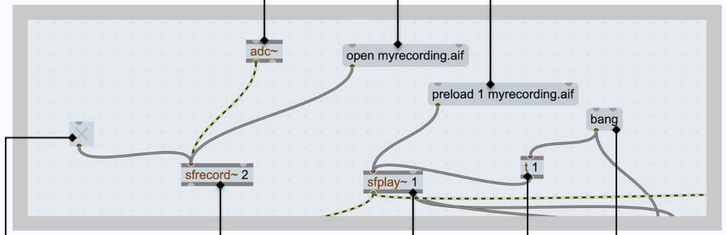
MAX/MSP PATCH Input+Data Transmission

Sound Input & Playback

preload 1 myrecording.aif
preloads the file into memory, ensuring smooth playback.

open myrecording.aif
loads the recorded file for playback.

adc~
captures real-time microphone input, allowing the system to record the user's voice.



records the incoming audio signal into a file for later processing and playback.

sfrecord~ 2

plays back the recorded file through the audio output.

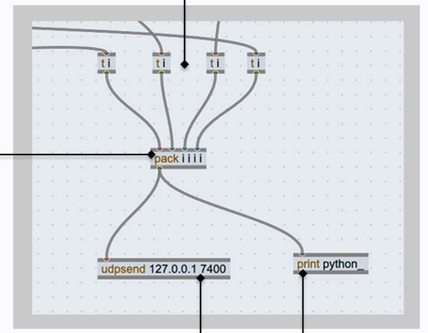
sfplay~ 1

Data Transmission to Python

trigger (t i i i i)
ensures that the data is structured and sent in the correct order.

combines multiple integer values into a single data packet for transmission.

pack i i i i



sends the packed data to Python via UDP at port 7400 on the local machine.

udpsend 127.0.0.1 7400

displays the sent data for debugging purposes, ensuring the values are correctly formatted.

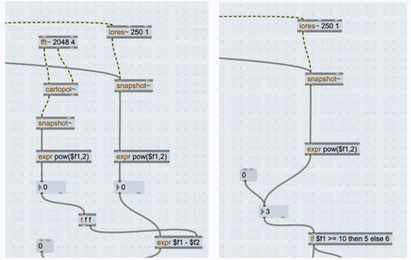
print python

bang & trigger (t 1)
ensures the playback starts properly by sending a trigger signal.

MAX/MSP PATCH

Iteration

Harmonic



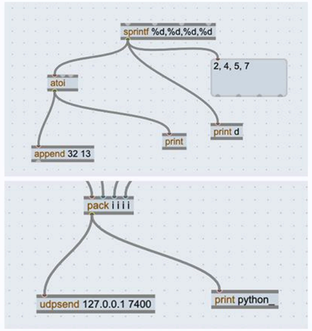
First Version: Harmonic-to-Noise Ratio (HNR) Calculation

Initially, HNR was calculated by comparing harmonic and noise power using FFT (fft~ 2048 4), cartopol~, and lores~ 250 1. snapshot~ captured real-time values, and expr pow($f1,2) computed squared amplitudes. The difference expr $f1 - $f2 estimated harmonic strength. However, the complexity made real-time processing impractical, leading to a simplified method.

Second Version: Simplified Harmonic Energy Calculation

Initially, HNR was calculated by comparing harmonic and noise power using FFT (fft~ 2048 4), cartopol~, and lores~ 250 1. snapshot~ captured real-time values, and expr pow($f1,2) computed squared amplitudes. The difference expr $f1 - $f2 estimated harmonic strength. However, the complexity made real-time processing impractical, leading to a simplified method.

Data Transmission



First Version: Direct Serial Transmission from Max to Arduino

Initially, Max sent data directly to Arduino via serial communication (Serial), formatting numerical values using atoi and sprintf before sending them. However, Arduino was unable to correctly interpret the incoming data, resulting in garbled text and communication failures. This issue arose because Max and Arduino handle serial data differently—Max sent data as raw ASCII characters, while Arduino expected properly structured serial messages. Without clear delimiters or correct encoding, the received data was unreadable.

Second Version: Adding Python as an Intermediary

To resolve the formatting issue, Python was introduced as an intermediary to reformat and forward the data. Instead of sending data directly to Arduino, Max now transmits formatted values via udpsend to Python at 127.0.0.1:7400. Python receives the data, processes it, and repackages it into a structured format that Arduino can interpret before forwarding it via serial communication. This ensures that the data format remains consistent and eliminates transmission errors, allowing Arduino to correctly process incoming values.

PYTHON

Data from MAXMSP

Import required libraries

Set up UDP socket for Max/MSP communication

Set up serial communication with Arduino

Initialize variables

Define wait time before resending data

Receive and process UDP data from Max/MSP

Handle exceptions

Send formatted data to Arduino

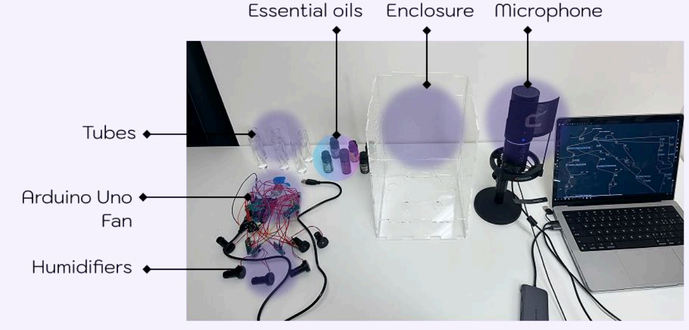
```python
import socket
import struct
import serial
import time

UDP_IP = "127.0.0.1"
UDP_PORT = 7400

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.settimeout(0.5)

sock.bind((UDP_IP, UDP_PORT))

print(f"🎧 Listening for UDP data from Max/MSP on {UDP_IP}:{UDP_PORT}...")

ARDUINO_PORT = "/dev/cu.usbmodem21201"
BAUD_RATE = 9600

try:
    arduino = serial.Serial(ARDUINO_PORT, BAUD_RATE, timeout=1)
    print(f"✅ Connected to Arduino on {ARDUINO_PORT} at {BAUD_RATE} baud.")
except serial.SerialException:
    print("⚠ Error: Could not connect to Arduino. Check your port and try again.")
    arduino = None

last_valid_numbers = None
last_received_time = 0.0

WAIT_TIME = 3.0

while True:
    try:
        data, addr = sock.recvfrom(1024)

        if len(data) >= 16:
            numbers = struct.unpack("!4i", data[-16:])
            print(f"📡 Received from Max/MSP (stored): {' '.join(map(str, numbers))}")

            last_valid_numbers = numbers
            last_received_time = time.time()

    except socket.timeout:
        pass
    except Exception as e:
        print(f"⚠ Error receiving data: {e}")

    if last_valid_numbers is not None:
        elapsed = time.time() - last_received_time
        if elapsed > WAIT_TIME:
            if arduino:
                formatted_message = " ".join(map(str, last_valid_numbers)) + "\n"
                arduino.write(formatted_message.encode("utf-8"))
                print(f"✅ Sent to Arduino (no new data for {WAIT_TIME}s): {formatted_message.strip()}")

            last_valid_numbers = None
```

ARDUINO

```ino
#define NUM_MISTERS 6
const int misterPins[NUM_MISTERS] = {2, 3, 4, 5, 6, 7}; // 6 misting units control pins
const int fanRelayPin = 8;  // Relay pin controlling the fan
const int fanRunTime = 15000;  // Fan run time (6 seconds)

int selectedMisters[3] = {0, 0, 0}; // Stores 3 mister indexes
int sprayTime = 3000;  // Default spray time (3 seconds)
int receivedCount = 0;

void setup() {
    Serial.begin(9600);
    for (int i = 0; i < NUM_MISTERS; i++) {
        pinMode(misterPins[i], OUTPUT);
        digitalWrite(misterPins[i], HIGH); // Initial state not pressed
    }

    pinMode(fanRelayPin, OUTPUT);
    digitalWrite(fanRelayPin, LOW); // Default fan off
    delay(500); // Wait for relay to stabilize
}

void loop() {
    if (Serial.available()) {
        int command = Serial.parseInt();

        // Only store the first three mister indexes
        if (receivedCount < 3 && command >= 1 && command <= 6) {
            selectedMisters[receivedCount] = command - 1; // Store index (array indexing from 0)
            receivedCount++;
        }
        // Fourth number determines spray time
        else if (receivedCount == 3 && (command == 7 || command == 8)) {
            sprayTime = (command == 7) ? 3000 : 5000; // 7=3s, 8=5s
            receivedCount++;

            // All 4 data have been received, start spraying
            activateSelectedMisters();
        }
    }
}

// Activate the 3 selected misters in sequence
void activateSelectedMisters() {
    Serial.println("Starting selected misters:");
    for (int i = 0; i < 3; i++) {
        Serial.print("Starting Mister ");
        Serial.println(selectedMisters[i] + 1);
        pressButton(misterPins[selectedMisters[i]]); // Activate mister
        delay(sprayTime);  // Spray for 3s / 5s
        stopButton(misterPins[selectedMisters[i]]); // Stop the mister
    }

    // After all misters finish spraying, start the fan
    startFan();

    receivedCount = 0; // Reset count, wait for next input
}

// Simulate pressing the button
void pressButton(int pin) {
    Serial.println("Activating Device");
    digitalWrite(pin, LOW); // Simulate button press
    delay(500); // Hold for 0.5 seconds
    digitalWrite(pin, HIGH); // Release button
}

// Ensure the mister is fully off (press twice to ensure off)
void stopButton(int pin) {
    Serial.println("Stopping Device...");
    delay(1000); // Wait 1 second to prevent mode switching

    digitalWrite(pin, LOW); // Second press (switch mode)
    delay(500);
    digitalWrite(pin, HIGH);
    delay(1000); // Wait 1 second to prevent mode freeze

    digitalWrite(pin, LOW); // third press (turn off)
    delay(500);
    digitalWrite(pin, HIGH);
}

// Start fan, automatically shut off after 6s
void startFan() {
    Serial.println("Starting fan...");
    digitalWrite(fanRelayPin, HIGH); // Start fan
    delay(fanRunTime);  // Run for 6 seconds
    Serial.println("Stopping fan...");
    digitalWrite(fanRelayPin, LOW);  // Stop fan
}
```

COMPONENTS

Components



Assembly Process


1

Soldering the circuit board in the atomiser


2

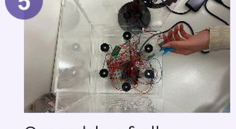
Laser cut acrylic sheet


3

Assembling an Arduino uno and atomiser


4

Assembling Shells

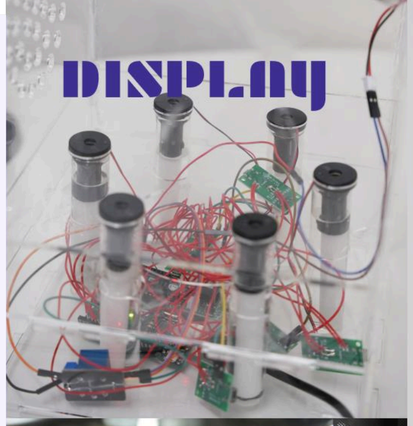

5

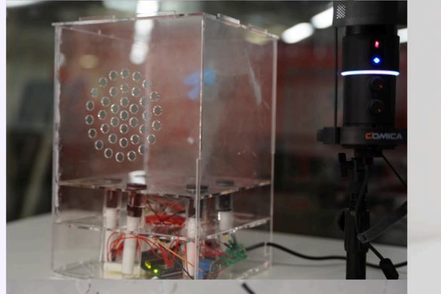
Assembly of all components

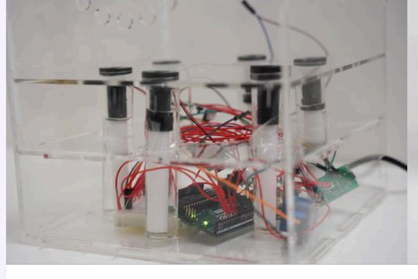

6

Assembly of all components

DISPLAY







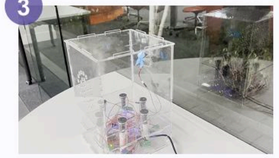
WORKFLOW

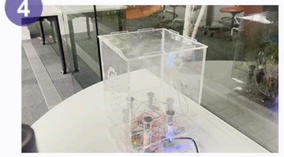
1

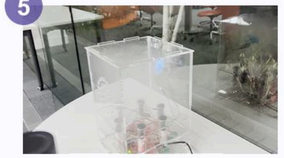
The user speaks or sings into the microphone

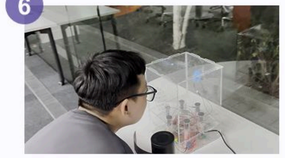
2

Computer analyses audio

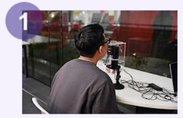
3

PC drives arduino to control the atomiser

4

Gases of different odours are emitted in sequence

5

The fan blows the gas mixture out

6

The user smells the generated odour

USER TEST & FEEDBACK

1


Computer Output Value: 1457
Scent Smelled: Herbal
Self-Perception of Voice: typically gay, mid-range pitch, tends to have a rhythmic "bouncing" quality, generally speaks softly.
Does the Scent Match Your Voice? Yes, it feels harmonious and not too intense.
Experience Feedback: Very interesting, the generated scent matches my voice.
Suggestion: It would be great to have real-time feedback on captured voice data.

2


Computer Output Value: 1357
Scent Smelled: Woody
Self-Perception of Voice: Typical female voice.
Does the Scent Match Your Voice? Yes, my voice feels harmonious.
Experience Feedback: Feels very novel and intriguing.
Suggestion: Increase the playability and make the scent stronger.

CONCLUSION

The Smell the Sound installation successfully transformed auditory inputs into olfactory outputs, offering participants a unique sensory experience that simulates synesthesia. By analyzing speech characteristics, the system mapped voice features to specific scents, allowing users to "smell" their own voice. This innovative approach expanded the perception of sound beyond traditional auditory and visual modalities, reinforcing the potential of multisensory interaction in design.

The integration of Max/MSP, Arduino, and atomisers enabled real-time voice analysis and scent diffusion, ensuring a seamless interaction between the user and the system. The structured mapping of sound features to distinct fragrances allowed participants to gain a deeper awareness of their vocal expressions through scent-based feedback.

Through user engagement and real-time feedback, the installation provided a compelling and interactive experience that bridged the gap between hearing and smell. It encouraged participants to reflect on their own voices in a novel way, creating a deeper emotional connection between sound and personal expression.