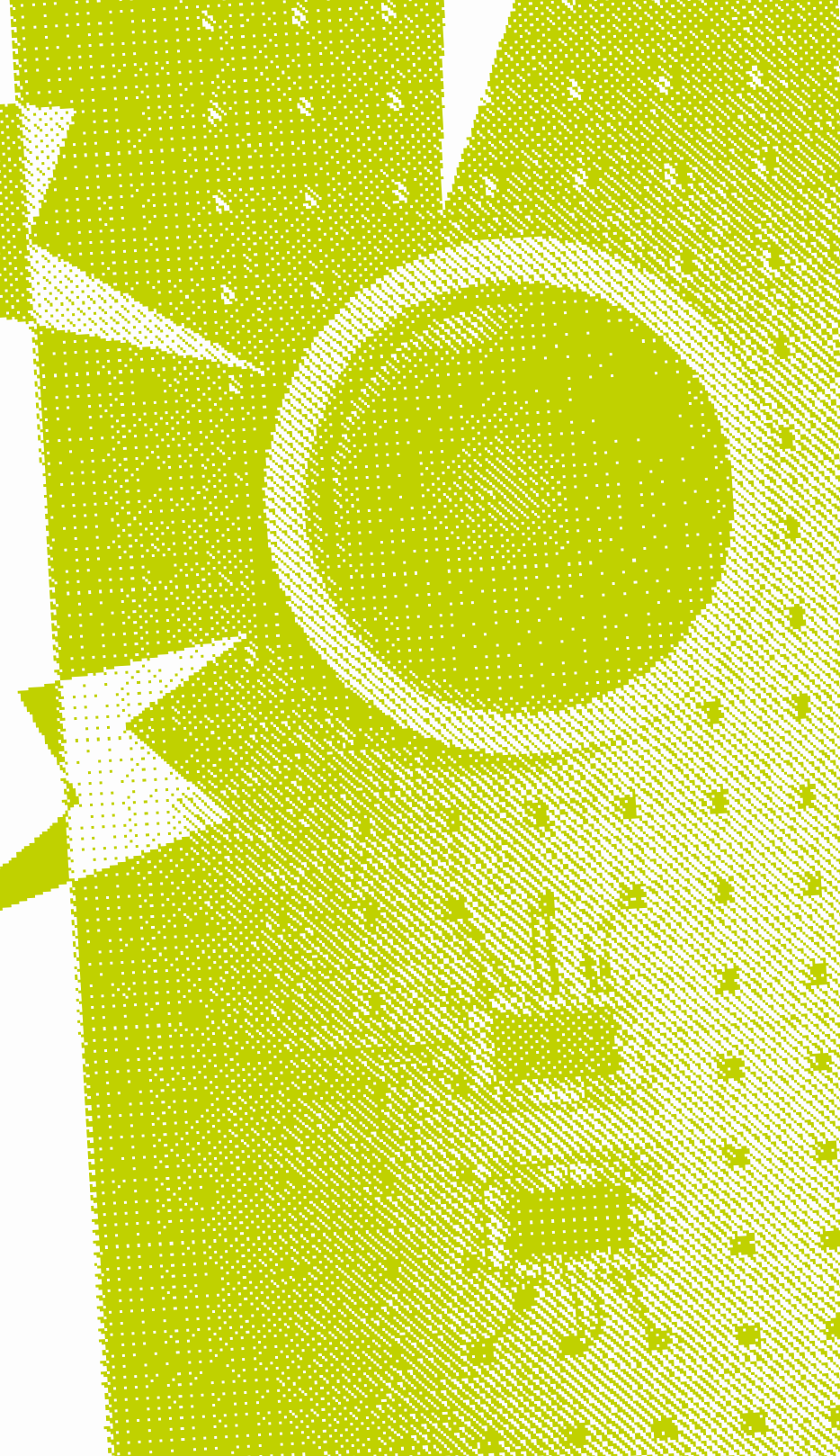
DEATH TONES!

WISE 25/26 RORY DOYLE

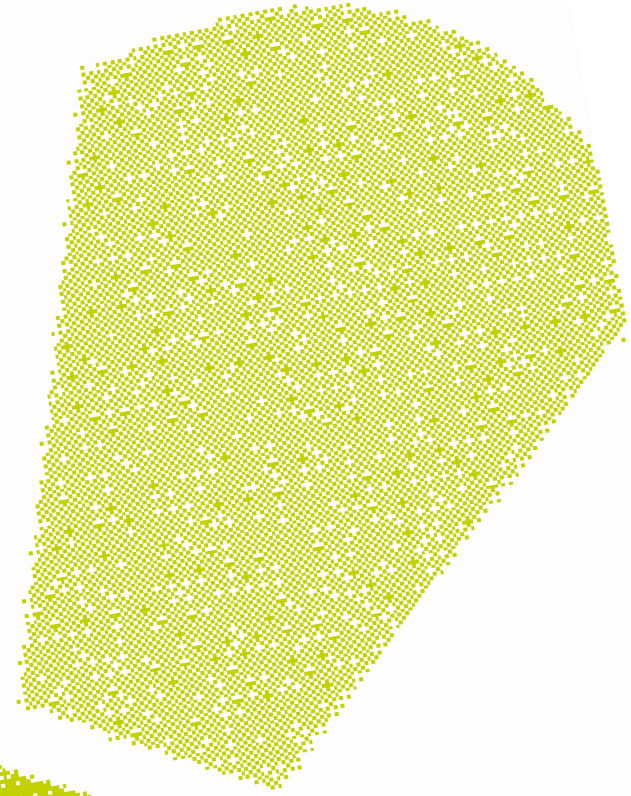


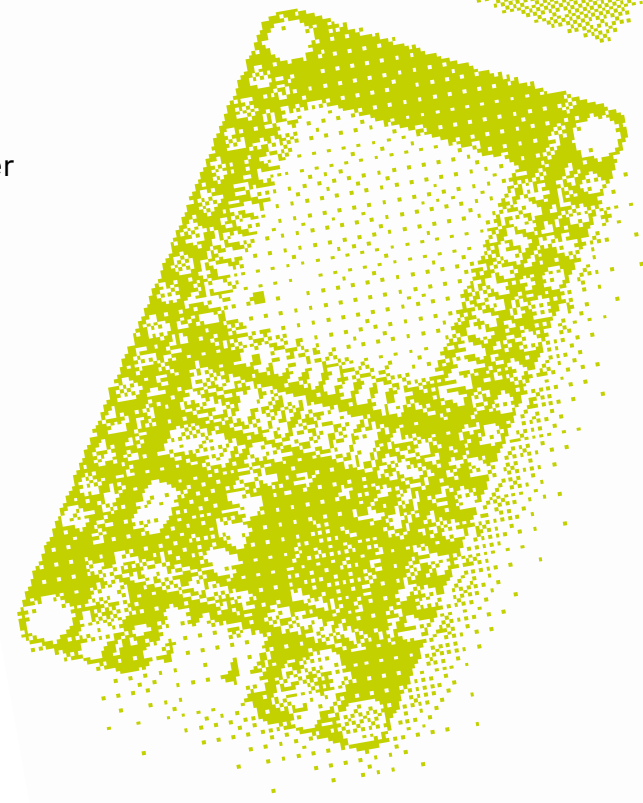
IDEA

This interactive installation draws from the Irish and Scottish folk phrase “someone has walked over your grave”, an expression traditionally used to explain a sudden, involuntary shiver. The saying imagines that a chill occurs when someone unknowingly steps across the site of your future burial, momentarily disturbing the boundary between presence and absence.

Installed at the dog’s grave of Kurwenal outside M18 , the work literalises this superstition. People who step onto or approach the grave activate a proximity sensor system. An ESP32 microcontroller transmits real-time distance data into the exhibition space, where it modulates the frequency of a continuous sine-wave soundscape.

Physical presence at a burial site, once associated with an unseen shiver, becomes measurable, transmitted, and audible. The work connects superstition, signal, and atmosphere, transforming a subtle bodily sensation into a spatial acoustic event. The grave becomes an interface, a presence becomes a signal, and the haunting is recoded as data.





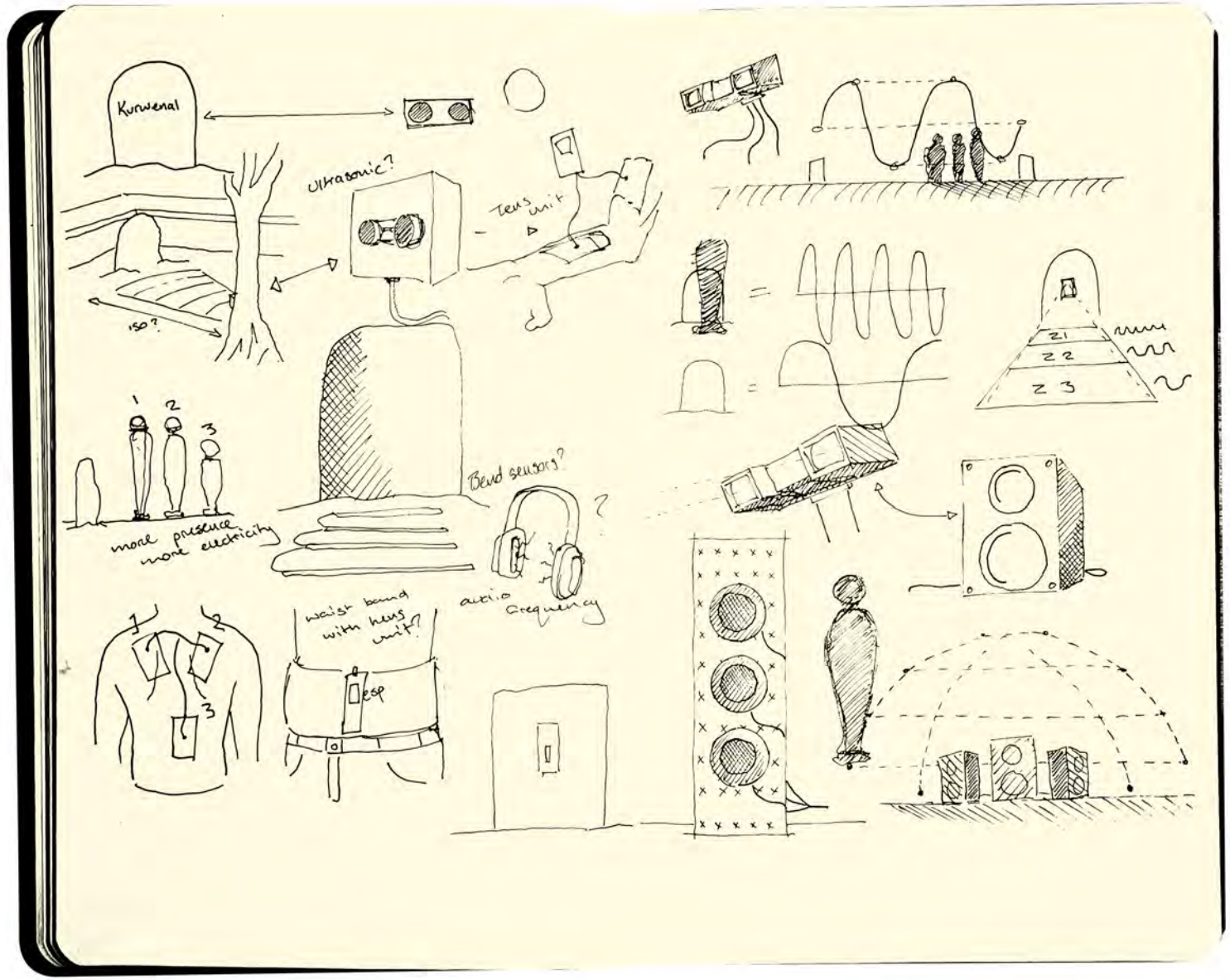
PROCESS

The project unfolded in three phases:

1. Site & Sensor
The infrared distance sensor was chosen for reliability outdoors. The sensor measures distance and transmits the data wirelessly via ESP32.

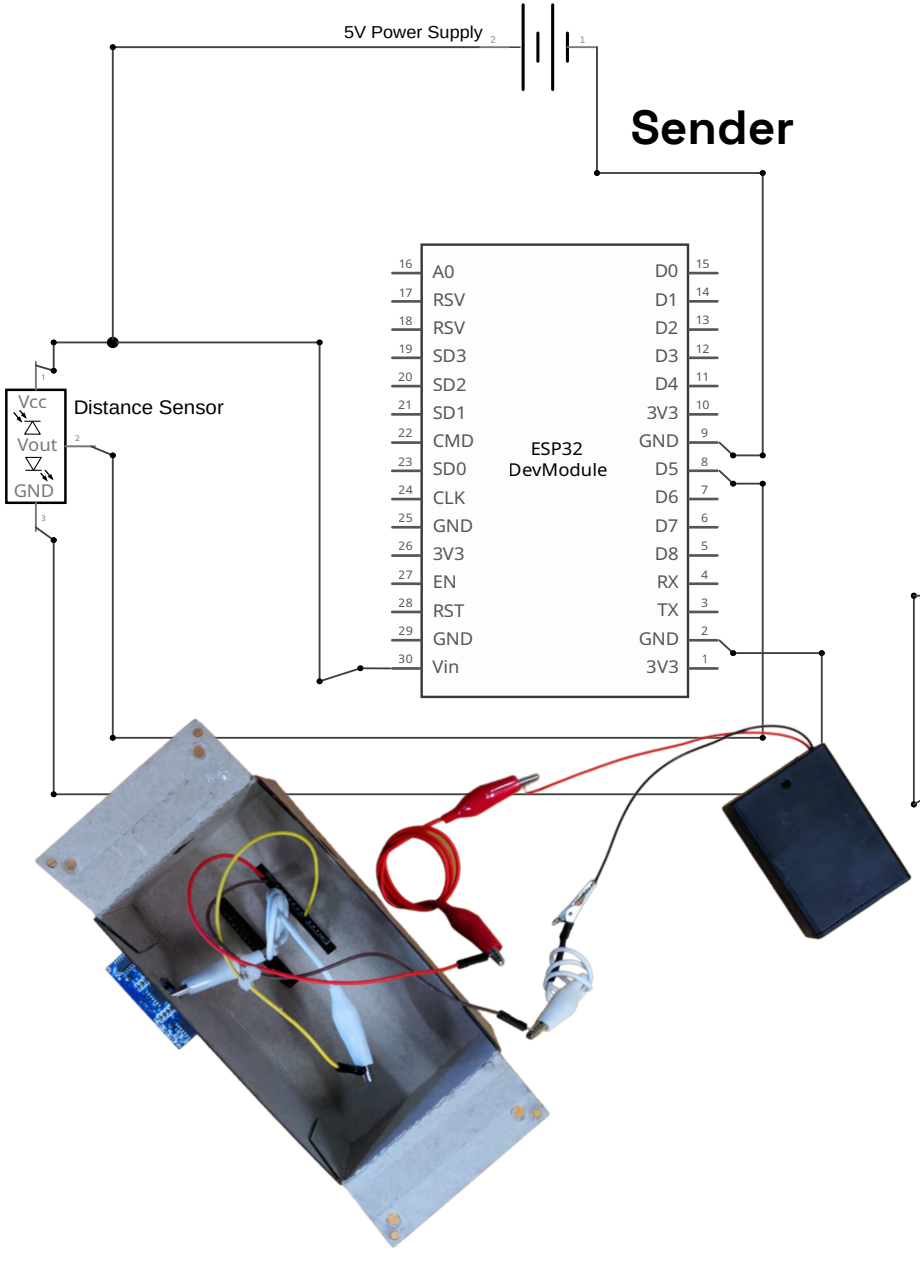
2. Mapping & Sound Design
Distance → Data → Frequency
The closer a body is to the grave, the higher the sine wave frequency. A smoothing algorithm prevent abrupt changes. The sine wave is minimal, the idea being that the medium shapes perception more than content. The shift in frequency is subtle, creating tension without the spectacle.

3. Transmission & Atmosphere
Visitors inside the exhibition experience the sound but cannot see what manipulates it's changing soundscape. This separation mirrors the original superstition: cause and effect are uncoupled. In media-theoretical terms, the installation makes visible the invisible infrastructure of signals and supernatural networks surrounding us.

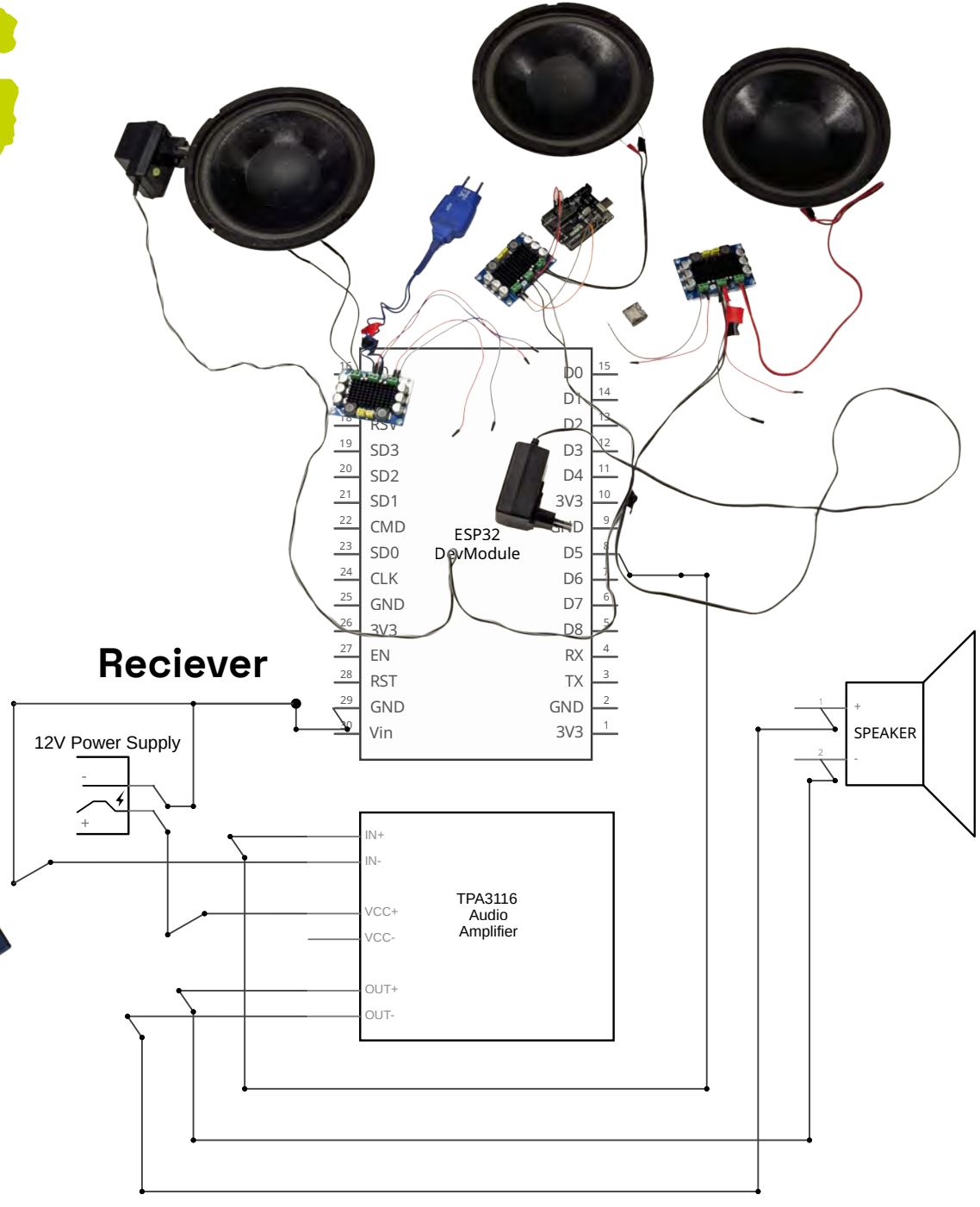


WiRiNG

Sender



Reciever



CODE

Reciever

```
#include <SensingTheCampusLib.h>
#include <WiFi.h>
#include <math.h>

//MQTT

String eduRoamUser = "hode8762@uni-weimar.de";
String eduRoamPassword = "Ramsey1/234567";

String studentName = "Rory";
String hiveMQUserName = "SensingTheCampus";
String hiveMQPassword = "SensingTheCampus2025.";
String hiveMQServerAddress =
  "5f2326c3de2044798fd58379a751ce5c.s1.eu.hivemq.cloud";

String zoneTopic = "esp/distanceZone";

//AUDIO

#define DAC_PIN_L 25
#define DAC_PIN_R 26

#define SAMPLE_RATE 8000

#define BASE_FREQ 95.0
#define DETUNE    1.6

float phase1 = 0.0;
float phase2 = 0.0;

float noiseLP = 0.0;
float breathPhase = 0.0;
float driftPhase  = 0.0;
float stereoPhase = 0.0;

//soundscape
float presence = 0.35;        //base sound
float targetPresence = 0.35;

//timings
unsigned long lastSampleMicros = 0;
const unsigned long samplePeriod = 1000000 / SAMPLE_RATE;


void setup() {
  Serial.begin(115200);

  //start audio immediately
  targetPresence = 0.35;
  presence = 0.35;

  connectToEduRoam(eduRoamUser, eduRoamPassword);

  connectMQTT(
    hiveMQServerAddress,
    hiveMQUserName,
    hiveMQPassword,
    studentName
  );
```

```
void loop() {
  loopMQTT();

  //slow memory/inetia
  presence += (targetPresence - presence) * 0.0015;
  presence = constrain(presence, 0.2, 1.0);

  unsigned long now = micros();
  if (now - lastSampleMicros >= samplePeriod) {
    lastSampleMicros += samplePeriod;
    generateAudioSample();
  }
}


void generateAudioSample() {

  breathPhase += 0.00035;
  driftPhase  += 0.00012;
  stereoPhase += 0.00005;

  float breath = 0.6 + 0.4 * sin(breathPhase);
  float drift  = sin(driftPhase) * 0.4;
  float width  = sin(stereoPhase) * 0.06 * presence;

  float f1 = BASE_FREQ + drift;
  float f2 = f1 + DETUNE + presence * 0.8;

  phase1 += TWO_PI * f1 / SAMPLE_RATE;
  phase2 += TWO_PI * f2 / SAMPLE_RATE;

  if (phase1 > TWO_PI) phase1 -= TWO_PI;
  if (phase2 > TWO_PI) phase2 -= TWO_PI;

  float sine =
    0.6 * sin(phase1) +
    0.4 * sin(phase2);

  float noise = random(-1000, 1000) / 1000.0;
  noiseLP += (noise - noiseLP) * 0.01;

  float signal =
    sine * breath +
    noiseLP * presence * 0.35;

  signal *= 0.45 + presence * 0.55;

  int dacL = 128 + signal * 85;
  int dacR = 128 + (signal + width) * 85;

  dacWrite(DAC_PIN_L, constrain(dacL, 0, 255));
  dacWrite(DAC_PIN_R, constrain(dacR, 0, 255));
}


void receiveProcedure(char *topic, byte *payload, unsigned int length) {
  int value = mqttPayloadToInt(payload, length);

  if (value >= 1 && value <= 130) {
    targetPresence =
      map(value, 130, 1, 20, 100) / 100.0;

    Serial.print("Presence → ");
    Serial.println(targetPresence);
  }
```

Sender

```
#include <SensingTheCampusLib.h>
#include <WiFi.h>

// eduroam wifi
String eduRoamUser = "hode8762@uni-weimar.de";
String eduRoamPassword = "Ramsey1/234567";

// MQTT
String studentName = "Rory";
String hiveMQUserName = "SensingTheCampus";
String hiveMQPassword = "SensingTheCampus2025.";
String hiveMQServerAddress = "5f2326c3de2044798fd58379a751ce5c.s1.eu.hivemq.cloud";

String zoneTopic = "esp/distanceZone";

//sensor settings

#define SENSOR_PIN 34
#define SAMPLES 10

//sensor functions

float readVoltage() {
  int sum = 0;
  for (int i = 0; i < SAMPLES; i++) {
    sum += analogRead(SENSOR_PIN);
    delayMicroseconds(200);
  }
  float avg = sum / (float)SAMPLES;
  return avg * (3.3 / 4095.0);
}

int getDistanceCM(float voltage) {
  //infrared empirical curve (20–150 cm)
  if (voltage < 0.4) return 200;   //out of range distance
  float distance = 61.573 * pow(voltage, -1.106);
  return (int)distance;
}

void setup() {
  Serial.begin(115200);

  // ADC config
  analogReadResolution(12);
  analogSetAttenuation(ADC_11db);

  // Connect WiFi
  connectToEduRoam(eduRoamUser, eduRoamPassword);

  // Connect MQTT
  connectMQTT(
    hiveMQServerAddress,
    hiveMQUserName,
    hiveMQPassword,
    studentName
  );
}
```

```
void loop() {
  // keep MQTT alive
  connectMQTT(
    hiveMQServerAddress,
    hiveMQUserName,
    hiveMQPassword,
    studentName
  );
  loopMQTT();

  // read sensor
  float voltage = readVoltage();
  int distance = getDistanceCM(voltage);

  //distance to value mapping
  // 20–150 cm → 1–130
  int mappedValue = 0;

  if (distance >= 20 && distance <= 150) {
    float normalized = (distance - 20) / 130.0;   //0.0 – 1.0
    mappedValue = 1 + round(normalized * 129);    //1 – 130
  }

  //serial monitor output
  Serial.print("Distance: ");
  if (distance < 20 || distance > 150) {
    Serial.print("out of range");
  } else {
    Serial.print(distance);
  }
  Serial.print(" cm | Value: ");
  Serial.println(mappedValue);

  //mqtt sender
  if (mappedValue > 0) {
    sendMessageMQTT(mappedValue, zoneTopic);
  }

  delay(1000);
}
```

SENDER

On the sender side, the ESP32 functions as a sensing unit that measures physical distance and translates it into a numerical value that can influence the sound system. Installed inside the Sensing the Campus box and powered by a 5V battery pack, it is connected to a Sharp GP2Y0A02YK0F infrared distance sensor on pin 34. During setup, the board configures its analog-to-digital converter to 12-bit resolution with the appropriate attenuation, allowing it to accurately read the sensor's analog voltage output across its working range.

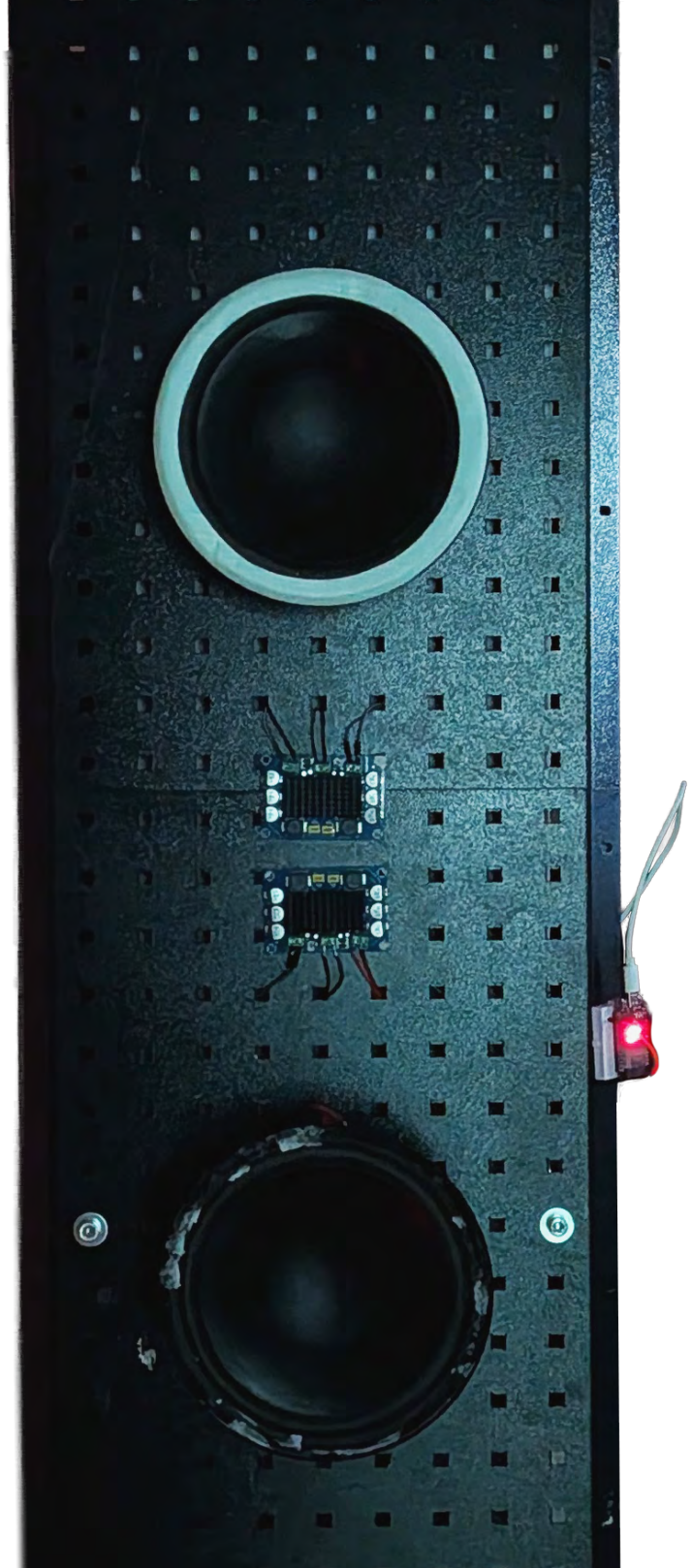
In the main loop, the ESP32 repeatedly samples the sensor multiple times in quick succession and averages the readings to reduce noise and instability. This averaged analog value is converted into a voltage between 0 and 3.3 volts, which is then translated into an estimated distance in centimeters using an empirical formula specific to the sensor. Because the sensor's response is nonlinear, this mathematical conversion is necessary to approximate real-world distance from the voltage output. If the voltage falls below a usable threshold, the system treats the reading as out of range.

Rather than sending raw distance data, the code maps the valid range of 20–150 cm onto a scaled value between 1 and 130. This normalisation compresses physical space into a numerical range that the receiver can interpret. Values outside the defined range are ignored, which prevents erratic behavior when nothing is detected. Once per second, the ESP32 sends the mapped value, creating a steady stream of data that reflects how close someone is to the sensor. Physical movement in front of the box is continuously translated into a structured digital signal that drives the evolving sound environment on the receiving side.

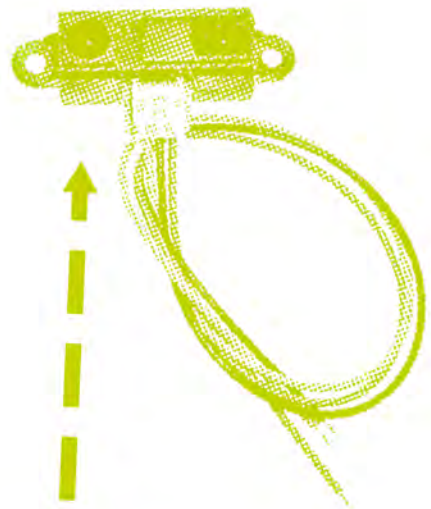
RECIEVER

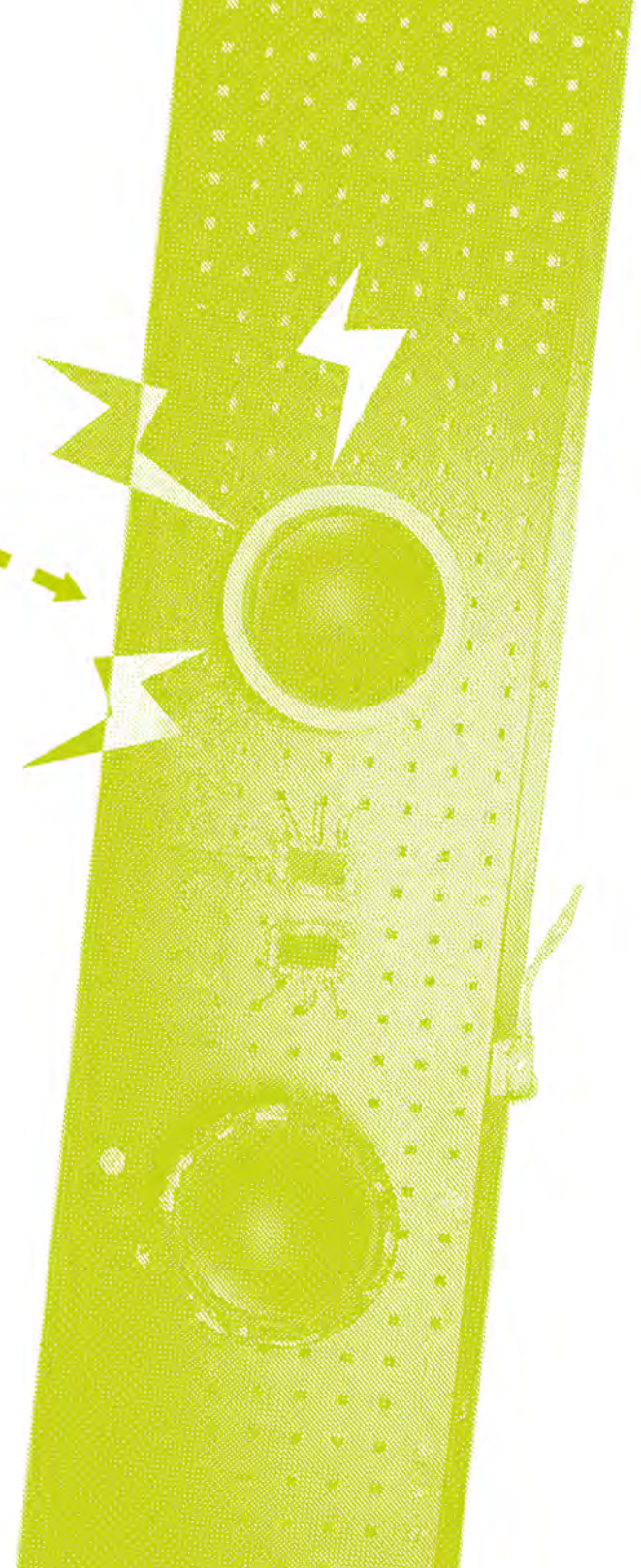
In this setup, the ESP32 acts as a small real-time sound synthesizer rather than a playback device. During initialisation, it connects to WiFi and subscribes to the MQTT topic so it can receive distance or presence data from the other ESP. At the same time, it prepares the audio system by defining a fixed sample rate and setting up timing control using micros(), which ensures that audio samples are generated at a steady interval. Instead of using PWM, the code writes directly to the ESP32's built-in DAC pins (GPIO 25 and 26), which output true analog voltages between 0 and 3.3V. These pins are connected to the input stages of two external TPA3116 amplifiers, each powered by its own 12V supply, and the amplifiers then drive the passive speakers.

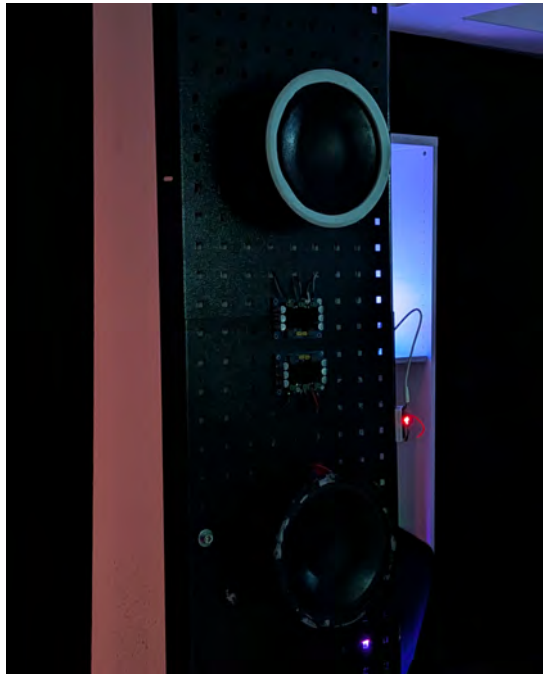
Once running, the main loop continuously maintains the MQTT connection while also checking whether it is time to generate the next audio sample. Each sample is calculated mathematically: two closely tuned sine waves form the base drone, slow modulation shapes pitch drift and amplitude "breathing," and filtered noise adds subtle texture. The signal is centered around the DAC midpoint (about 1.65V) so it oscillates cleanly as an audio waveform. For stereo output, the left and right channels are written separately to pins 25 and 26, with a slight slow variation between them to create spatial width. Incoming data does not immediately change the sound; instead, it is smoothed over time so the drone gradually shifts in intensity and complexity. This setup allows the installation to behave like a responsive, atmospheric sound organism rather than a simple tone generator.

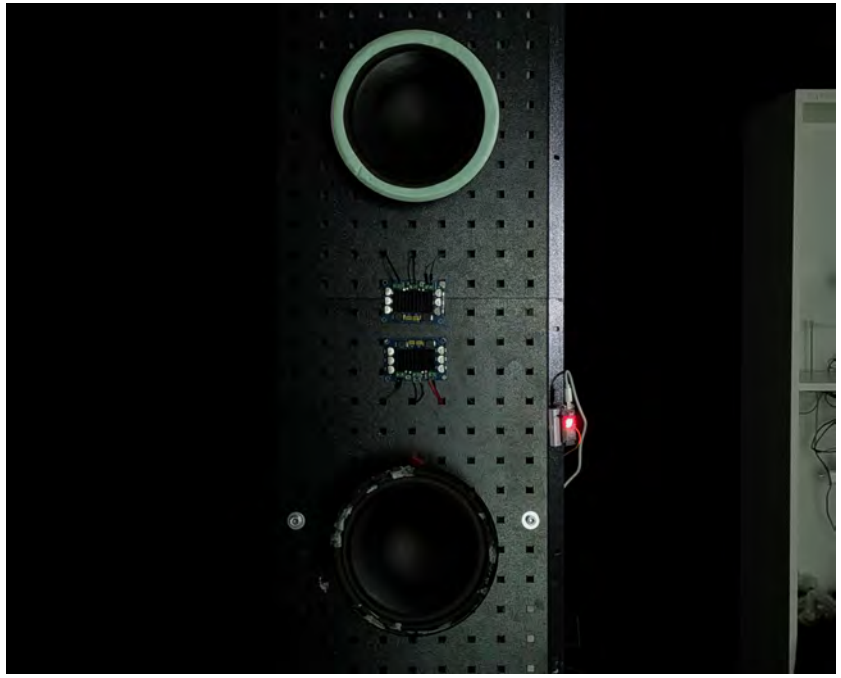


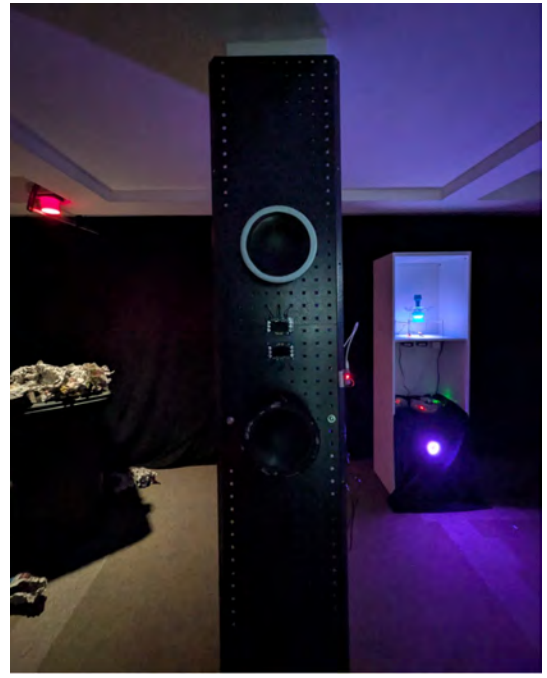
FINAL












Der klügste und edelste
aller Hunde,
Der weltberühmte


Der klügste und edelste
aller Hunde,
Der weltberühmte