

## ARTICLE

# Interactive audio toolkit: Creating sonic experiences and installations with low-cost, low-power microcontrollers

**Aman Jagwani\*** and **Victor Lazzarini** 

Department of Music, Maynooth University, Maynooth, Ireland

(This article belongs to the *Special Issue: Ubimus Cross-Cultural Approaches to Interaction and Creativity*)

## Abstract

Interactive sound installations and experiences are increasingly prevalent in museums, galleries, events, and various spaces worldwide. These installations are often powered by microcontroller board running programs that control interactivity and possibly even audio processing. Commonly used microcontrollers such as the ESP32 and Raspberry Pi Pico are generic, necessitating custom implementations for audio-specific functionalities. Existing audio libraries often have limited signal processing features and may not be designed with interactivity in mind. This paper presents the Interactive Audio Toolkit, which focuses on the seamless integration of sound generation with sensors to create interactive sound experiences with low-power, low-cost microcontrollers. This C++ toolkit is structured around sensor input and audio output classes, providing a simple interface to generate flexible audio responses from sensor interactions. This paper details the toolkit's structure, components, and workflow, highlighting its ability to foster new forms of sonic and musical interactions for artists and audiences. A case study of an installation utilizing the toolkit is also presented.

**Keywords:** Sound installation; Interactive audio; Embedded audio; Interaction design; Ubimus toolkits

**\*Corresponding author:**Aman Jagwani  
(aman.jagwani.2023@mumail.ie)

**Citation:** Jagwani A, Lazzarini V. Interactive audio toolkit: Creating sonic experiences and installations with low-cost, low-power microcontrollers. *Arts & Communication*. 2026;4(2):025180032. doi: 10.36922/AC025180032

**Received:** April 30, 2025**1st revised:** July 29, 2025**2nd revised:** August 18, 2025**3rd revised:** October 28, 2025**Accepted:** November 3, 2025**Published online:** November 20, 2025

**Copyright:** © 2025 Author(s). This is an Open-Access article distributed under the terms of the Creative Commons AttributionNoncommercial License, permitting all non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

**Publisher's Note:** AccScience Publishing remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

## 1. Introduction

Embedded systems such as microcontrollers and single-board computers are important parts of sonic experience, performance, and interaction design in ubimus practices, both in terms of do-it-yourself (DIY) approaches<sup>1</sup> and more comprehensive, professional, or commercial settings. In a previous work,<sup>2</sup> a spectrum of embedded audio platforms was explored, from low-cost, computationally simple microcontrollers to high-performance field programmable gate arrays (FPGAs), within the context of live performance customization. Building on this foundation, this paper delves deeper into the former area and expands into the context of interactive experience design with the Interactive Audio Toolkit (IAT). IAT is a header-only C++ toolkit that was developed to facilitate flexibility, creativity, and efficiency in designing interactive audio experiences with simple, inexpensive, and ubiquitous microcontrollers such as the ESP32,<sup>3</sup> STM32,<sup>4</sup> and Raspberry Pi Pico.<sup>5</sup>

The reason for targeting low-cost, low-power microcontrollers in this context is their widespread availability and versatility, allowing selection from a variety of generic microcontrollers with a broad range of features. For instance, if an interactive audio project requires wireless networking, the ESP32 may be suitable. For balancing ultra-low power consumption and high performance, the appropriate STM32 series chip can be used.

Furthermore, using generic microcontrollers allows the leveraging of existing knowledge, libraries, and experience. All of these features, along with their low price, align with the ubimus concept of scalability. Often, interactive installations are extremely large-scale, covering huge expanses of area and requiring the use of large numbers of microcontroller boards. For example, the installation *Chaal*, which will be discussed in depth as a case study in Section 4, comprises a massive bamboo structure designed for participant exploration across a broad area. To support audio interaction throughout the installation, multiple microcontroller boards were required. The use of low-cost microcontrollers was helpful in making this level of distribution feasible. In such scenarios, scalability becomes beneficial.

IAT is a set of C++ classes or objects that was created within the Arduino<sup>6</sup> and PlatformIO<sup>7</sup> frameworks, enabling cross-platform development and deployment. The objects mainly encompass different sensor inputs, such as ultrasonic and LIDAR sensors, audio outputs, and a control signal generator in the form of a sequencer.

One unique aspect of the toolkit is that the audio outputs are not limited to digital audio; they also include mechanical audio generation with actuators such as stepper motors and relays, expanding the possible sound palette. The sequencer maps common musical parameters such as pitch and velocity to all of the audio outputs, including the mechanical outputs. For example, with relays, pitch is mapped to the frequency at which the switch is clicked. Each sequencer step can have a different clicking rate, resulting in interesting syncopated rhythms.

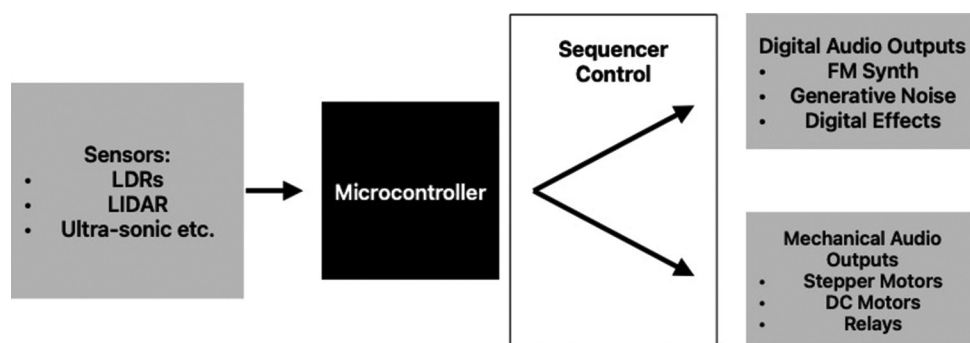
IAT operates on a trigger-based mechanism in the following manner: participants' movements, actions, or gestures in a sensor-equipped space cause sensor readings to reach specific thresholds, triggering sequencers assigned to various outputs. Each sensor can have multiple thresholds, producing different sonic outcomes based on the sensor reading ranges. In addition, different sensor combinations with assigned outputs can interact to create layered soundscapes in real-time. [Figure 1](#) provides an overview of the structure of IAT. Artists or designers can “compose” their interactive piece based on the installation

structure or space, with participants actively realizing the composition. This approach is exemplified in *Chaal*, where the bamboo structure and the pathways through which participants navigated influenced the sonic and musical outcomes. The strategic placement of sensors along the bamboo framework enabled sonic layering and precise triggering, shaping the overall experience.

These ideas embody the ubimus aspects of social interaction and audience participation, as demonstrated in previous projects like The Memory Tree installation.<sup>8</sup> This installation heightened participants' awareness of a passive object, such as a tree, during social interactions. Similarly, IAT fosters deeper interactions between the audience, artwork, space, and structure through evolving soundscapes resulting from their participation. This is because the process of triggering can result from relatively passive interactions with sensors, such as walking past a LIDAR sensor or stepping on a load cell. Consequently, the sonic results can make the participant more aware of the artwork as well as their own movements and gestures.

Another connection with previous ubimus work is the Handy Metaphor.<sup>9</sup> This work explored touch-less interaction in musical situations and developed examples based on ultrasonic and video-based motion tracking, highlighting the potential benefits and insights of such interactions. The IAT can be seen as an extension of this work as a tool to facilitate the design of these kinds of interactions. In addition, the ability to combine multiple sensors at strategically placed locations can enhance the motion-tracking by evaluating their readings in coherence. The IAT can also provide an enhancement to musical results through its sequencer. Just like in a modular synthesizer, a sequencer is a core signal generator, enabling the generation of a variety of sounds from slow. Evolving drones to fast, syncopated micro-patterns, IAT's sequencer can achieve similar results but with a completely different sonic palette and context.

IAT also addresses an important gap in resources available in this area. There are several drivers easily available for almost any sensor and actuator within the Arduino framework. There are also audio libraries such as the Arduino Audio Tools library,<sup>10</sup> which cover a range of sound effects and generators. However, while designing interactive audio experiences, programmers usually need to link these aspects together by hand, which may be time-consuming and even inaccessible for artists and creatives with limited programming experience. IAT unifies the interaction design workflow by connecting sensors, actuators, audio processing, and musical control in a simple and intuitive manner.



**Figure 1.** Structure of the Interactive Audio Toolkit, demonstrating the sensor inputs, audio outputs, and sequencer-based control system, in relation to the microcontroller

In addition, while domain-specific frameworks such as Max/MSP, PureData,<sup>11</sup> and Csound<sup>12</sup> provide conducive platforms for sound design with a vast array of audio processing and synthesis functions, they may not be suitable in the context of low-powered, low-cost computing. First, the available memory and computational capacity may not be sufficient for these frameworks, which are primarily desktop-based. Second, due to their desktop orientation, these platforms can lack direct integration with hardware, which IAT aims to address by providing a musical link between sensor inputs and audio outputs. Moreover, to use such platforms, additional tools and programming may be required. For example, in an installation using PureData, digital audio processing can be handled in PureData, but for sensor processing, a microcontroller board such as an Arduino may be needed. This requires sensor driver programming, along with establishing a communication interface between the Arduino and PureData, either wired through serial communication over USB or wirelessly using OSC, each adding complexity and reducing portability as compared to a self-contained microcontroller-based system. A toolkit such as IAT helps bridge this gap.

While systems like Csound do have bare-metal implementations<sup>13</sup> that can run on microcontroller boards such as the Electrosmith Daisy,<sup>14</sup> many of the target platforms for IAT, such as the ESP32, may not have sufficient memory resources to run Csound, which requires additional memory, as seen with the usage of the 65 MB external SDRAM used in the Daisy chip. Moreover, IAT can also be used in conjunction with a system like bare-metal Csound, where Csound handles digital audio processing, while IAT manages sensor and actuator interactions along with trigger-processing and sequencing.

Although these domain-specific systems may not be suitable for the target contexts of IAT, they play an important role in establishing the foundation for toolkits such as this. Audio programming in systems such as PureData and Csound revolves around modular structures

or unit generators such as oscillators, envelopes, and filters that are interconnected through their inputs and outputs, visually in PureData or textually in Csound, to form a signal flow. Similarly, the objects in IAT exemplify modular structures, with their interconnection establishing the signal flow. In the following sections, the paper will present IAT in greater detail, highlighting its classes, features, and workflow. Then, an installation, *Chaal* by Asim Waqif, where this toolkit was deployed, will be discussed.

## 2. Toolkit components

The toolkit is organized into three main components:

- **Inputs** – Includes sensors such as light-dependent resistors (LDRs), load cells, ultrasonic sensors, and LIDAR
- **Outputs** – Comprises digital audio outputs (e.g., FM synthesizers) as well as mechanical sound actuators such as motors and relays
- **Sequencer** – The main control system of IAT that unifies the inputs and outputs in a musical context.

This enables the artist, or designer, to focus at a macro level on the relationships between sensors, audio, and music. The sequencer also adds a compositional aspect to the programming process with its ability of sonic pattern creation.

The toolkit relies on various common Arduino drivers for getting some sensor values and driving some actuators but provides a musical, intuitive interface for them. It is important to note that some supported sensors, like load cells, and actuators, such as stepper motors, require additional hardware components such as breakout boards or motor drivers to work with them. This toolkit focuses on the software aspects of musically communicating with these peripherals and assumes that appropriate hardware circuit design is already in place.

In addition, there is no fixed user interface or control system provided in IAT; however, there is support for

connecting with a variety of different user interfaces through a universal JSON control message interface. Users can easily use this to add any control system that they prefer such as web interfaces, physical circuit-based interfaces, or software GUI applications. The code for the toolkit is available on GitHub (URL is given in the Availability of data section).

## 2.1. Inputs

The input classes derive from a Sensor base class and are organized by type of sensor rather than communication method or protocol to keep the design more high-level and intuitive for artists, along with programmers. This means that there are separate classes for LIDAR sensors,<sup>15</sup> SONAR or ultrasonic sensors, and load cells,<sup>16</sup> currently in the toolkit, instead of I2C<sup>17</sup> or UART<sup>18</sup> input classes, which are the protocols that these sensors would use for communication. The exception is the ADC or analog-to-digital converter class. This is because ADCs, which entail simple analog readings from input pins, are generic and support various different simple sensors such as LDRs and force-sensitive resistors. Moreover, these sensors are easily interchangeable across these pins. Sensors that can communicate with multiple protocols, such as some Maxbotix Ultrasonic Sensors,<sup>19</sup> will have methods in their classes to initialize and use either of them depending on the available I/O pins on the board. At present, a limited number of commonly used sensors are supported. But this can easily be expanded in the future.

As shown in Table 1, each sensor has its own communication protocol and requires associated drivers. Studying the drivers and implementing interfaces to read all of their values can be time-consuming. Thus, IAT provides a unified way to use any sensor. The programmer has to simply call the sensor type's constructor and its read method as follows:

```
Lidar lidar(LIDAR_UPDATE_INTERVAL);
lidar.read();
```

Each input sensor has a way to control the rate of readings taking place. In complex systems with many

**Table 1. Interactive Audio Toolkit's supported sensor types and their communication protocols**

Sensor type	Communication protocol
LDR	ADC
VL53L1X Time-of-Flight (ToF)	I2C
TFMini Long-range LIDAR	UART
TFMINI Plus LIDAR	I2C
MB1604 Ultra-sonic	UART or ADC or I2C
HX711 Load Cell	Serial

inputs and outputs, it may be beneficial to slow down sensor readings to allow the processor time to complete other tasks, especially in a limited powered, resource-constrained environment like a microcontroller. In addition, if different sensors have additional parameters or features that need to be set, there is an intuitive method provided for that. For example, the `setMode` method of the LIDAR class enables the sensor to be toggled between long and short distance modes.

There is also a method to enable and disable sensors, providing users the option to manage all sensors even after a program is running on a board. This allows users to disable unused sensors in real-time through a user interface rather than during programming. This flexibility in sensor configuration helps conserve resources. For example, in the case study below, the web interface was used to enable and disable sensors while the program was running, demonstrating this capability.

With the universal JSON communication interface, described in Section 2.6, the real-time sensor readings can also be sent to a graphical user interface for visualization. This can be beneficial in devising strategic placements for sensors within an installation.

Finally, all of the inputs have the ability for users to set minimum and maximum values to work and tailor thresholds within their usable ranges in a particular space.

Figure 2 provides a representation of the IAT sensor classes and their configurations.

The visualization and configuration of a LIDAR sensor with a web interface connected to IAT has been shown through a YouTube video.<sup>1</sup>

## 2.2. Outputs

All the different types of outputs in IAT have their own classes that derive from a base output class. This base class contains common member variables, structures, and handles essential tasks for output generation and sequencing. It contains a sequencer object that will control the particular output and a pointer to a SequencerConfig object that holds parameter values such as subdivision and number of steps. These are dynamically assigned to an output's sequencer by associated OutputHandler objects, based on sensor thresholds being reached.

Each SequencerConfig and OutputHandler is associated with a sensor threshold. Hence, whenever a sensor is initialized, its corresponding SequencerConfigs and OutputHandlers are also initialized for three different user-definable thresholds—low, mid, and high. The default

<sup>1</sup> <https://youtu.be/lezHXtCVWvs>

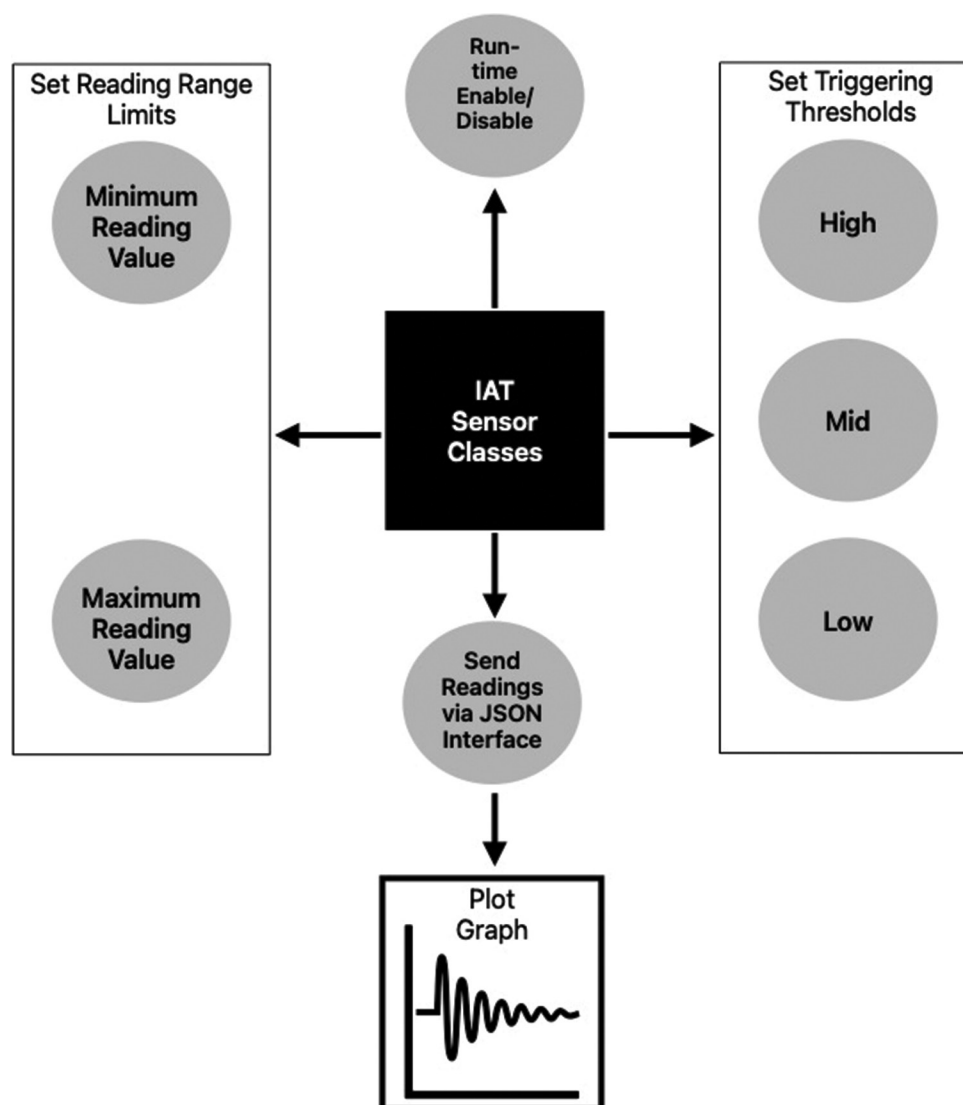


Figure 2. Interactive Audio Toolkit sensor classes and their configurations

number of thresholds for each sensor is three, but this can be expanded or reduced as per project requirements. The user can map any chosen output to the `OutputHandlers`, so when a threshold is reached, the `OutputHandler` assigns its corresponding `SequencerConfig` to its mapped output.

This enables enhanced flexibility because the same output can be triggered by different sensors, or at the same time, the same output can be triggered by different thresholds of the same sensor but with different `SequencerConfigs` and thus different patterns or musical outputs. For example, a LIDAR sensor's three thresholds can be thought of in terms of proximity as: close, medium, and far. Changing the distance can trigger either three different outputs to layer patterns or evolve the same output's pattern by altering parameters such as rhythmic subdivision and

melodic scale. This second approach represents a different form of continuous control. Often, in sensor-based audio projects, sensor readings directly control audio parameters like cutoff frequency, where greater distance means higher cutoff, for example. However, this can be linear and musically one-dimensional. With IAT's pattern evolution possibilities, continuous control gains an added musical dimension.

To maintain uniformity across different outputs and adhere to the modular synthesis and electronic music sequencer paradigms, commonly seen parameters such as pitch and velocity are mapped for each sequencer step to the available parameters across each output, maintaining a consistent interface for control. Pitch is in the MIDI note number range of 0–127 and the velocity range of



0.00–1.00. All of the outputs have a process method that needs to be continuously called to run their sequencers upon triggering. The different outputs supported by IAT are highlighted in the subsections below.

Figure 3 provides an overview of each of the outputs currently supported by IAT and the mapping of the sequencer's pitch and velocity values to their respective parameters.

### 2.2.1. Audio synthesis

The toolkit contains an expandable set of commonly used sound synthesis<sup>20</sup> objects which presently include:

- **Oscillator:** The fundamental unit of sound synthesis, enabling audio waveforms such as sine, sawtooth, square, and triangle waves to be generated
- **ADSR Envelope:** A segmented sound shaping tool that enables audio amplitude and timbre to be modulated over time across attack, decay, sustain, and release phases
- **FM Synthesizer:** A synthesis technique that comprises one oscillator modulating the frequency of another oscillator at audio rate, producing rich textures and harmonics in the resultant sound. This technique was implemented due to the vast timbral possibilities that it can create with relatively low computational requirements, remaining consistent with the resources available on the chosen platform

- **Generative Noise Synthesizer:** This includes white noise going through a second-order resonator filter, which is modulated by an interpolating random number generator. The user has control over cutoff frequency, bandwidth, random modulation depth, and random modulation rate, allowing the creation of evolving and sweeping textures.

The toolkit will be expanded in the future to include additional synthesis techniques such as subtractive and granular synthesis with the addition of objects covering different kinds of filters, more flexible envelopes, *etc.*

### 2.2.2. Stepper motor

For stepper motors,<sup>21</sup> the toolkit uses the AccelStepper Arduino Library.<sup>22</sup> With the sequencer, the pitch parameter for each step is mapped to the speed at which the motor runs or the frequency of steps, and the velocity is mapped to the acceleration or the rate of change of speed, enabling interesting rhythms and melodies to be created by sequencing this motor. Below is the stepper motor's process method that highlights this:

```
void process(bool click)
{
    if (!seqConfig)
    {
```

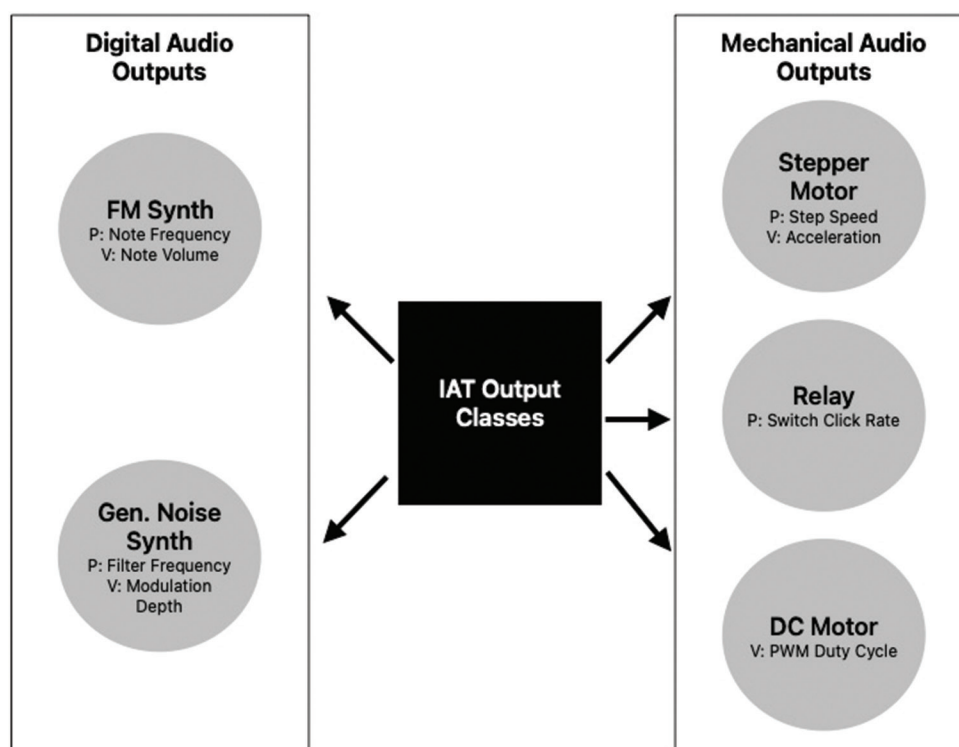


Figure 3. Interactive Audio Toolkit output classes and their respective pitch and velocity mappings where applicable

```

        return;
    }
    seqTrig = getSubDivTrig(click);
    bool gate = sequencer.updateStepAndGate(
        seqTrig, seqConfig->sequencerActive);

    if (gate)
    {
        float newFreq =
            midiNoteToFrequency(sequencer.getCurrent
            Pitch());
        float velocity = sequencer.
            getCurrentVelocity();
        if (newFreq != freq)
        {
            freq = newFreq;
            stepper.setSpeed(freq);
            stepper.setAcceleration(velocity*1000);
        }
        stepper.runSpeed();
    }
};

```

### 2.2.3. DC vibration motor

For DC motors,<sup>23</sup> the velocity is the main controlling parameter. It is mapped to the duty cycle of the pulse width modulation (PWM) controlling the motor. Going from 0 to 1 increases the intensity and speed of the vibration. The pitch value is treated more as a switch, where a value above 0 is on and 0 is off for a particular step. This is because the possibilities of control for PWM and this type of motor are limited. Nevertheless, these mappings enable several interesting syncopated patterns to be sequenced by interspersing fast and slow vibrations among the steps of the sequencer. The following is this motor's process method:

```

void process(bool click)
{
    if (!seqConfig)
    {
        return;
    }
    seqTrig = getSubDivTrig(click);
    bool gate = sequencer.updateStepAndGate(seqTrig,
        seqConfig->sequencerActive);
    if (gate)
    {
        int pitch = sequencer.getCurrentPitch();
        if (pitch > 0)
        {
            digitalWrite(MOTOR_PWM_PIN, HIGH);
        }
    }
}

```

```

float velocity = sequencer.getCurrentVelocity();
dutyCycle = static_cast<int>(scale(velocity, 0.0f,
    1.0f, 0, 254));
}
else
{
    dutyCycle = 0;
    freq = 0;
    digitalWrite(MOTOR_PWM_PIN, LOW);
}
ledcWrite(channel, dutyCycle);
}

```

### 2.2.4. Relays

For the relays, the pitch parameter is mapped to the rate at which the switch is clicked. This results in several sonic possibilities from chirp-like textures at high speeds to clicking textures at low speeds. Again, a combination of different speeds at different sequencer steps can result in interesting syncopations. The following is the relay's process method:

```

void process(bool click)
{
    if (!seqConfig)
    {
        return;
    }
    seqTrig = getSubDivTrig(click);
    bool gate = sequencer.updateStepAndGate(
        seqTrig, seqConfig->sequencer
        Active);
    unsigned long currentTime = micros();
    if (gate && sequencer.getCurrentPitch() > 0)
    {
        stepDivider = scale(sequencer.getCurrentPitch(),
            0, 127, 1, 128);
        if (stepDivider > 0)
        {
            duration = noteDuration * 1000000 /
                stepDivider;
        }
        if (currentTime - lastSwitch >= duration)
        {
            lastSwitch = currentTime;
            if (relayState == LOW)
            {
                relayState = HIGH;
            }
            else
            {
                relayState = LOW;
            }
        }
    }
}

```

```

        digitalWrite(relayPin, relayState);
    }
}
};

```

### 2.3. Sequencer

The sequencer is central to the effectiveness of IAT. It provides a familiar musical interface for controlling all the outputs, from conventional audio synthesis to the less conventional mechanical audio outputs. It includes several features tailored to musical creativity in interactive audio experiences. Users can control the following:

- Number of active steps
- Sequence direction
- Clock division – All the sequencers in a program are driven by a common clock or metro object. Each sequencer can divide the metro's rate to run at different rhythmic subdivisions
- Sequence trailing and tapering – While participants are within a threshold range, a particular sequence is triggered and plays continuously. However, when they exit the range, it may not be musical for the sequence to have a hard stop. Thus, the sequencer provides for a trailing mechanism. The programmer or user can set a sequence running time over which the sequence tapers off once its triggering range is exited. The tapering is done by randomly eliminating steps from the sequence, adding gaps, and reducing sequence intensity gradually. The rate at which steps are eliminated can also be controlled by users by setting the probability weight of the sequencer. The higher the weight, the greater the likelihood that steps are eliminated, and the higher the rate of tapering.

The sequencer is a musical and aesthetic design choice in IAT made due to the ease and flexibility with which it enables complex and involved musical ideas to be established in real-time by participants. Instead of triggering singular events or notes, each gesture of the participant can instantiate a complete musical “part” or member of the sonic orchestra. This takes the mechanism of triggering beyond a binary on and off switch into deeper pattern generation and layering.

Figure 4 visually presents the sequencer's configuration parameters along with an example eight-step sequence of pitch and velocity steps.

Examples of IAT sequences running on a relay, stepper motor, and FM synthesizer are demonstrated in a YouTube video,<sup>2</sup> with the sequencer visualized and configured with a web interface in these examples.

<sup>2</sup> <https://www.youtube.com/playlist?list=PL0g25mjdSj84d-Jfj8xoFcCIEc8DvcsM>

### 2.4. Trigger-rate processing

IAT incorporates trigger-rate processing at both the output and sensor threshold levels, facilitating adaptive control over system responsiveness. This functionality is achieved through a cool-down mechanism and an auto-play system.

Users can define a trigger-rate threshold, specifying the maximum number of times a given output or sensor threshold can be activated per minute. If this threshold is exceeded, the corresponding output or sensor enters a cool-down state for a user-defined duration, during which it remains unresponsive to additional triggers. This prevents excessive activation, mitigating potential issues such as sound buildup, over-triggering, or unwanted cacophony.

Conversely, if an output or sensor threshold is not activated frequently enough, users can define an auto-play threshold, which determines the maximum time that can elapse before a sequencer is automatically triggered on a specific output or threshold. This ensures sustained engagement and prevents prolonged inactivity in the system.

These trigger-rate processing methods enhance user interaction by integrating the common musical principle of dynamics into the system's behavior. Periods of increased or reduced user activity influence the perceived loudness or quietness of the sonic output, allowing for a fluid and evolving interaction between participants and the installation. This adaptability introduces a simple yet effective mechanism for shaping musical expression based on audience engagement, making IAT well-suited for installations, performances, and interactive environments that need to dynamically evolve in response to user participation.

### 2.5. Trigger-priority level

In large or complex installations where IAT may be deployed, multiple sensors or threshold conditions may target the same output, leading to overlapping triggers. IAT's trigger priority system resolves these conflicts by allowing the user to define precedence with the help of priority levels, among trigger sources, thereby determining which sequence is activated.

This mechanism also permits the integration of spatial dynamics into the sonic composition process. For example, there may be an installation in which a LIDAR sensor at the entrance and a load cell at the exit both target a common stepper motor. When both sensors are triggered simultaneously by participants, the trigger with the higher priority level is executed first. When combined with the trailing and tapering mechanism, a lower-priority trigger with a longer sequence duration can follow a high-priority trigger with a shorter duration, resulting in an extended



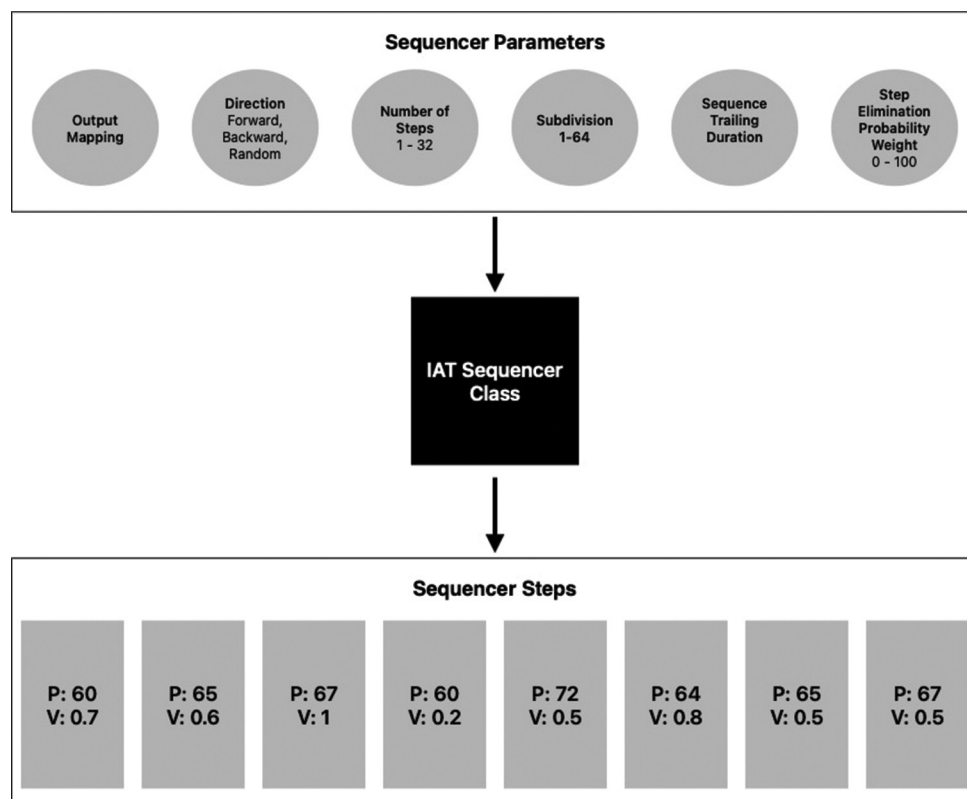


Figure 4. Interactive Audio Toolkit sequencer configuration overview and an example of the eight-step sequence

overall pattern. In this way, the user can use trigger priority as a compositional tool across multiple sequencers.

Another application of the trigger priority system is in perceiving and responding to gesture or movement direction. For instance, if a long-range LIDAR sensor defines three thresholds—where the furthest threshold has the lowest priority and the closest the highest—the participant’s movement across the sensor’s range can be mapped to corresponding changes in the same output through the three different threshold sequencers. As the participant approaches the sensor, the higher-priority sequences are activated immediately. In contrast, when moving away, the trailing of the highest-priority sequence allows it to persist during its running time over the lower-priority sequences, preserving a coherent transition in the overall pattern. Both examples illustrate how the trigger priority feature serves as a compositional or musical tool, enabling the user to customize the interaction between sensor input and output in a dynamic manner.

## 2.6. Control system connection

IAT does not include a fixed user interface but provides a universal JSON interface, implemented through ArduinoJSON,<sup>24</sup> to connect with various control systems such as web interfaces, physical circuits, or desktop

applications. This interface expects JSON messages with two fields: a controlName, which uniquely identifies a parameter based on a particular sensor or sequencer ID (e.g., ldr1-max or LIDAR-isEnabled), and a value to be assigned to that parameter.

For example, a JSON control message may appear as follows:

```
{
  "controlName": "ldr1-max",
  "value": 1000
}
```

For all configuration parameters, control handlers are maintained in a mapping that associates each unique controlName with a corresponding function, allowing for dynamic and flexible updates to sensor and sequencer parameters. The function below demonstrates how incoming JSON messages are parsed and routed to the appropriate control handler:

```
void processControlMessage(const String &jsonMessage)
{
  StaticJsonDocument<512> doc;
  DeserializationError error = deserializeJson(doc,
    jsonMessage);
```

```

if (error)
{
    Serial.println("JSON parsing error");
    return;
}
String controlName = doc["controlName"];
JsonVariant value = doc["value"];
if (controlHandlers.find(controlName) != controlHandlers.end())
{
    controlHandlers[controlName](value);
    Serial.println("Control updated:" + controlName);
}
else
{
    Serial.println("No control handler for:" + controlName);
}
}

```

Each control message is directed to update the corresponding sensor or sequencer configuration. For instance, a control message may modify the high threshold of an LDR sensor or change the directional mode of a sequencer.

This design provides a standardized interface for controlling parameters across sensors and outputs, allowing the user interface to be developed independently of the hardware configuration, while providing pluggability and flexibility. The following excerpt provides a high-level, simplified demonstration of how the JSON control message interface may be integrated with a WebSocket connection, for instance:

```

void onWebSocketMessage(const String& msg) {
    processControlMessage(msg);
}

void loop() {
    if (websocket.hasMessage()) {
        String msg = websocket.readMessage();
        onWebSocketMessage(msg);
    }
}

```

## 2.7. Save and recall

Often, interactive installations run in a space for a prolonged period of time, which can be days, weeks, or months. This means that they need to run consistently for their entire duration. If they are switched off, they need to recall the same state each time they are restarted. This also needs to be done in a simple way with

minimal to no personal involvement because technical knowledge or support may not always be available on-site. Especially in the case of audio projects, several parameters, thresholds, *etc.*, are carefully set at the time of programming/design that need to be recalled at boot time to align with the artist's vision. For this reason, IAT contains methods for saving and loading configurations in each of its classes that require parameters to be set. It uses the ArduinoJson<sup>24</sup> library, once again, to store configurations in simple JSON files that can be loaded and read upon booting.

## 3. IAT programming flow

The IAT organizes the development of interactive audio experiences into a clear, modular workflow. The IAT GitHub repository contains an example, in the main file of the PlatformIO project, that illustrates the typical steps involved in setting up an IAT project. In this example, two sensor types (an LDR sensor and a LIDAR sensor) and two output types (an FM synthesizer and a stepper motor) are integrated.

The programming flow can be summarized as follows:

- (i) Includes and Configurations: Import the required headers and define hardware pins and parameters (*e.g.*, BPM, GPIO assignments)
- (ii) Instantiate Sensors and Outputs: Create sensor and output objects, storing their pointers in arrays for later initialization
- (iii) Initialization: Initialize sensors and outputs through `initSensors()` and `initOutputs()`, and set up the control handlers with `setupAllControlHandlers()`. Additional peripherals, such as the I2S audio interface and metronome, are also initialized
- (iv) Initial Configuration: Define sequencer parameters (*e.g.*, number of steps, run-time, mode, threshold, subdivision, and trigger priority) for each sensor's threshold. Set pitch and velocity values for the output sequence. This step is just for demonstration in the example; once a user interface is integrated, these configurations would typically be dynamically handled during run-time
- (v) Initialize Output Handlers: Map sequencers to outputs and configure parameters such as probability weight for sequence tapering
- (vi) Set Output Parameters: Configure the outputs (*e.g.*, FM synthesizer settings) to tailor the resultant audio
- (vii) Main Processing Loop: Handle sensor readings, sequencer processing, and dynamic triggering.

This modular structure allows for rapid development and easy integration of additional sensors, outputs, or control mechanisms in various projects.

## 4. Case study: *Chaal* installation

### 4.1. Overview

*Chaal* is an installation by artist Asim Waqif. The first version of this was showcased in Mumbai, India, between March and June 2024. The artist's note, images, and details about the installation can be seen on the installation website.<sup>3,25</sup> In addition, video excerpts and insights can be seen in another work.<sup>26</sup>

The installation consisted of a large bamboo structure that encompassed a whole floor of a building, built on-site over a month. Participants were able to walk, climb, and explore into the structure, discovering different pathways, tunnels, details, and meanings embedded into the artwork. In addition, an interactive audio system was also embedded into the installation with various sensors, speakers, and actuators placed in strategic locations. IAT was used by the author to develop the software for this.

### 4.2. Interactive audio system and sensor placement

The ESP32 microcontroller was at the core of this system. The diagram in Figure 5 shows the various inputs and outputs that were available and connected to each ESP32 board. In total, there were three ESP32 boards spread across the installation. Not all inputs and outputs were used on all the boards, and they were enabled as they were required.

With IAT, each of the enabled sensors was assigned three thresholds that would trigger outputs. Then, the sensors were placed meticulously and thoughtfully around the installation to enable a sonic journey as participants ventured into the structure. For example, the first interaction was usually stepping on a load cell, which triggered two relays playing syncopated rhythms on either side of the participant. This created a slightly startling, stereo-like effect that could spark the participants'

curiosity. Next, the participant would enter the range of a long-range LIDAR sensor, triggering a bass pattern playing on an audio synthesis output. As the participants reached the shorter range of the LIDAR sensor, the bass pattern used smaller subdivisions and became more urgent. Simultaneously, they passed through the range of a SONAR sensor which triggered some high-pitched audio synthesis on an overhead speaker and frantic vibrations from a DC motor. Thus, as they moved deeper into the structure, they discovered the entire piece coming together. The aesthetic choice for the music and sounds was that of slightly unsettling, industrial, and atonal soundscape, with all sequences composed accordingly.

### 4.3. Wireless control and real-time configuration

The installation involved embedding the ESP32 boards into inaccessible parts of the bamboo structure. This meant that it would be hard to keep making changes to parameters by programming them manually. Hence, the board's wireless connection capabilities were used to provide a web interface. Using web-sockets,<sup>27</sup> IAT's JSON control interface, and the React JavaScript framework,<sup>28</sup> a web application was made. This enabled the sensor values to be monitored and plotted in real-time at the installation venue. It also enabled the different thresholds to be set, outputs to be assigned, and the sequences to be composed wirelessly, while being within the installation and interacting with it. All the ESP32 boards were connected to a common local network for the installation, and their web applications could be reached by accessing their IP addresses or a custom web address given to them with the help of the ESPmDNS library.<sup>29</sup> They could be accessed with mobile devices, tablets, or laptops, providing great convenience and portability.

### 4.4. Second iteration: Expanding interactivity with Raspberry Pi

The second version of the installation was exhibited in Jeddah in early 2025, introducing significant advancements

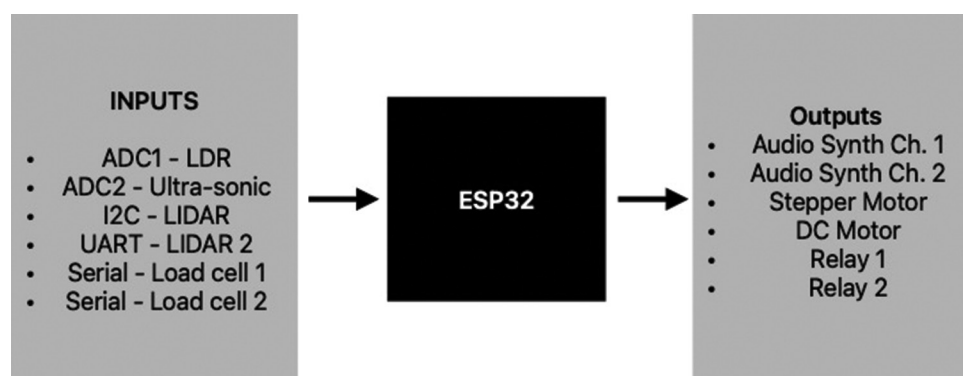


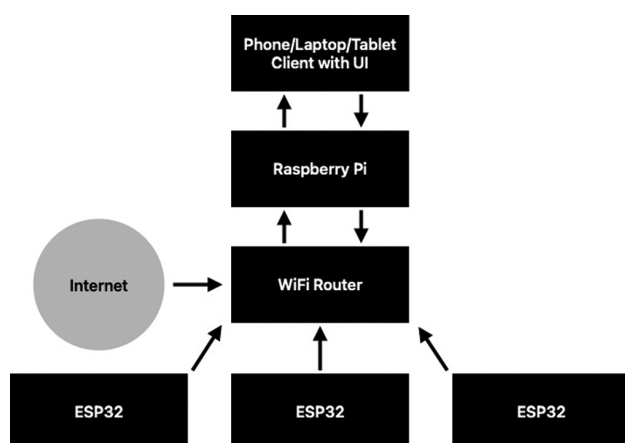
Figure 5. ESP32 I/O diagram depicting an example configuration of an ESP32 board used in the installation with its inputs and outputs

in the system architecture. While the use of IAT remained the same and a wireless interface remained central to configuration and monitoring, a Raspberry Pi was introduced as an intermediary server between the ESP32 boards and client mobile devices. Instead of each ESP32 acting as an independent server, all boards were connected to the Raspberry Pi, which hosted the web interface through a Node.js server.<sup>30</sup> This centralized approach streamlined control and facilitated cross-board communication.

This architecture enabled advanced interactions, such as a LIDAR sensor on one ESP32 triggering a stepper motor on another, thereby expanding the scope of interactive responses. In addition, the Raspberry Pi had access to online data sources, such as weather conditions, allowing external environmental factors (*e.g.*, humidity, temperature, atmospheric pressure) to influence the installation's sonic behaviors. This introduced a new dimension of automation and autonomous triggering, making the system feel more dynamic and responsive to its surroundings. A dedicated global processing interface was developed, allowing inputs from any sensor threshold across the ESP32 network, external environmental parameters, or even predefined time-based triggers to influence outputs. The available outputs included activated sequencers on any ESP32 board or custom sequences on any actuator across the installation. Figure 6 illustrates the second iteration of the system architecture.

#### 4.5. Global processing and the internet of musical things

The wireless communication used in this project links with the *ubimus* concept of the Internet of Musical Things (IoMT) as described previously.<sup>31</sup> In that work, three types of “musical things” are mentioned: smart instruments, mobile devices, and wearables. While mobile devices



**Figure 6.** Second iteration of system architecture, highlighting the integration with the Raspberry Pi, expanded interactivity, the networking system and the inter-board communication

were used here as well, this installation can extend the discussion into an additional type of “musical thing,” the custom microcontroller-based embedded system within interactive experiences.

The three ESP32 boards, along with the client devices used for configuration, operated as a networked system, collectively shaping the installation's sonic environment. The introduction of the Raspberry Pi as a central server not only streamlined control but also enabled global processing, where inputs from any ESP32 board could influence outputs across the entire installation. This meant that sensor activations were no longer confined to their respective boards; instead, they could trigger responses on other ESP32s, facilitating coordinated sonic and mechanical interactions.

With the global processing framework, the ESP32 boards could be thought of as interconnected performers that were orchestrated or conducted through the central Raspberry Pi server and control system. It also introduced new possibilities for interaction, such as integrating environmental data (*e.g.*, weather conditions) or time-based triggers into the installation's sonic behaviors. These advancements refined the system's capabilities while laying the groundwork for future installations that could leverage both local and internet-based data sources to create even more dynamic and responsive interactive audio experiences.

This kind of networked, reflexive system, in which devices respond not only to their local environment but also to one another, can be historically situated within broader theories of open, adaptive systems. Cybernetics, as introduced by Norbert Wiener,<sup>32</sup> and General Systems Theory, formulated by Ludwig von Bertalanffy,<sup>33</sup> both described foundational models of systems capable of feedback-based self-regulation and complex interactions. These early frameworks provide a theoretical context for understanding the global processing framework developed here.

#### 4.6. Exploration, participation, and sonic rewards

An important aspect of this installation was that curiosity was rewarded: the deeper the explorations by a participant, the greater the sonic rewards. For example, at some hidden locations, tactile transducers were installed. They were driven by the audio synthesis output, and when a participant entered their area, they vibrated the floor in a sequenced rhythm. Similarly, deeper exploration also revealed fine visual detailing and hidden nests, hammocks, and tunnels in the installation. This idea of discovering a piece through movement and exploration invited the participant to re-examine their relationship with their



surroundings. The fact that simple interactions with an extremely ubiquitous material like bamboo could yield a plethora of sonic and visual results can encourage audiences to think deeper about aspects that can be passive parts of regular interactions. This is similar to the ideas presented in the Memory Tree installation.<sup>8</sup>

#### 4.7. Social interaction and gamification in the sound experience

Audience participation was central to the installation experience. Social interactions also played a key role. The musical results for groups of people were vastly different from those for individuals. More participants would result in more sequences being triggered simultaneously. Also, as a big group of people would spread into different spaces of the installation, the sounds would also be spatialized with them. The participants were the central part of the music-making or *performance* process while the artist was solely part of the *composition* process, highlighting new kinds of musical relationship dynamics for both groups.

The process of moving, gesturing, and discovering the sound piece in this installation was extensively game-like, as intended by the artist. This is similar to Koszolko and Studley's study,<sup>34,35</sup> which examined gamification in a performance context, revealing new insights into audience engagement through playfulness. Here as well, the change in relationship to music and sound as a game proved to be very liberating and curiosity-inducing. It enabled a new sense of attachment to the musical process for participants who may not have had any musical background, embodying the *ubimus* concept of inclusivity.

In addition, the concept of collective participation enabled by this installation highlights a parallel with team-based dynamics in games. Just as many games rely on collaborative effort, the participants within the installation at any given time can be seen as a team collectively shaping the sonic environment. This team-like engagement reinforces the idea of music-making as a collective experience resulting from audience participation in this installation.

Furthermore, the playful, exploratory qualities of the installation helped reduce barriers and inhibitions among participants, similar to the effects of gamification discussed in Koszolko and Studley's paper.<sup>34</sup> This openness encouraged people to engage with the installation and with one another. The resulting soundscapes often embodied a sense of cohesion and unity. Across both iterations of the installation, despite their differing locations and cultural contexts, this aspect of playful collaboration remained consistent. The gamified musical process acted as a form of cultural bridge, with curiosity-driven interaction drawing

together diverse participant groups into a shared act of musical creation.

#### 4.8. Reflections and broader implications

This installation is a commentary on sustainability, anthropology, and ecology. By combining the bamboo-based vernacular architectural systems and modern technological resources, it presented a new way to look at sustainable and ecological future development with both aspects working together in a complementary, rather than contrasting manner, and possibly mitigating each other's limitations to some extent.

The installation, through its architectural-technological juxtaposition of vernacular materials with modern tools, also highlighted cross-cultural approaches to music-making. The architectural elements, artisanal techniques, and the artisans themselves originated primarily from north-eastern India, while the musical tools and frameworks, such as the sequencer and conceptual parallels to modular synthesizers, were rooted in Western electronic music traditions. The musical palette of the installation itself also leaned toward industrial, abstract electro-acoustic music. This created an interesting amalgamation of two artistic domains. In addition, the situating of these musical systems within locations such as Mumbai and Jeddah, each with its own local sonic cultures, further encouraged diverse interpretations and responses to the resulting soundscapes. Finally, the participatory aspect of the installation enabled cross-cultural interaction not just in form, but in practice, as audiences from varied cultural backgrounds collaboratively engaged in the musical process.

A YouTube video<sup>4</sup> provides a walkthrough of the installation, highlighting some of the sensor interactions and sonic results.

### 5. Conclusion

This paper presents IAT, highlighting its structure, components, and ability to streamline the interactive audio experience design process. Low-cost, low-power microcontrollers were chosen as the target for this toolkit due to their widespread availability, extensive set of options, features, and affordability. In spite of the inherent limitations imposed by these platforms, IAT provides a level of adaptability and evolutionary ability with features such as the trigger-rate processing and trigger priority levels. In addition, this toolkit, aimed at these platforms, complements the work being done in parallel on high-performance embedded systems such as FPGAs.<sup>2</sup>

<sup>4</sup> <https://youtu.be/EvB-BPgV3ww>



The case study demonstrated the effectiveness of the toolkit in creating meaningful experiences, enabling both audiences and artists to explore new ways of connecting with sound and music, while also fostering cross-cultural interaction in music-making.

At present, the toolkit contains a limited set of sensors and outputs. A future development would be the expansion to cover additional hardware as well as advanced motion-tracking techniques like computer vision. Machine learning techniques with frameworks like TensorFlow Lite<sup>36</sup> can also be incorporated to enrich the interactions and pattern recognition of the system. Finally, while the toolkit is limited to audio generation at this point, support for audio processing can also be added in the future.

## Acknowledgments

Aman Jagwani would like to acknowledge the support of Maynooth University through the Hume Scholarship program.

## Funding

This work was funded by the Hume Scholarship from Maynooth University.

## Conflict of interest

Victor Lazzarini is the Guest Editor of this special issue but was not in any way involved in the editorial and peer-review process conducted for this paper, directly or indirectly. Separately, other authors declared that they have no known competing financial interests or personal relationships that could have influenced the work reported in this paper.

## Author contribution

*Conceptualization:* All authors

*Investigation:* Aman Jagwani

*Software:* Aman Jagwani

*Writing—original draft:* All authors

*Writing—review & editing:* All authors

## Ethics approval and consent to participate

Not applicable.

## Consent for publication

Not applicable.

## Availability of data

The data and software are available from the git repository (<https://github.com/amanjagwani/interactive-audio-toolkit>).

## Further disclosure

This manuscript was presented in an initial, shorter form at the Ubiquitous Music Symposium in Macao, which took place from October 31 to November 02, 2024.

## References

1. Timoney J, Lazzarini V, Keller D. DIY electronics for ubiquitous music ecosystems. In: Lazzarini V, Keller D, Otero N, Turchet L, editors. *Ubiquitous Music Ecologies*. Milton Park: Routledge; 2020.
2. Jagwani A. Creative possibilities and customizability of live performance systems with open source programming platforms. In: Yaseen A, Bridges B, Messina M, Keller D, eds. *Proceedings of the 2<sup>nd</sup> International Symposium on Ubiquitous Music (UbiMus 2023)*. Northern Ireland: Ulster University; 2023:133-144. Available from: <https://www.ulster.ac.uk/conference/ubimus>
3. Espressif Systems. ESP32. Available from: <https://www.espressif.com/en/products/socs/esp32> [Last accessed on 2024 Apr 24].
4. STM32. *STM32 32-bit ARM Cortex MCUs*. Available from: <https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html> [Last accessed on 2024 Apr 24].
5. RaspberryPi. Available from: <https://www.raspberrypi.org> [Last accessed on 2024 Apr 24].
6. Arduino. Available from: <https://www.arduino.cc> [Last accessed on 2024 Apr 28].
7. PlatformIO. Available from: <https://platformio.org> [Last accessed on 2024 Apr 28].
8. Ribeira Netto A, Casthologe L, Oliosi A, Mateus A, Costalonga L, Coura D. Árvore das Memórias: Instalação Multimídia Interativa. In: *Proceedings of the Brazilian Symposium of Computer Graphic and Image Processing*; 2015.
9. Keller D, Gomes C, Aliel L. The handy metaphor: Bimanual, touchless interaction for the internet of musical things. *J New Music Res*. 2019;48:1-12. doi: 10.1080/09298215.2019.1654525
10. Schatzmann P. *Arduino-Audio-Tools*. Available from: <https://github.com/pschatzmann/arduino-audio-tools> [Last accessed on 2024 Apr 28].
11. Puckette M. Pure Data: Another Integrated Computer Music Environment. In: *Proceedings of the Second Intercollege Computer Music Concerts*, Tachikawa, Japan; 1997.
12. Lazzarini V, Yi S, Ffitch J, Heintz J, Brandtsegg Ø, McCurdy I. *Csound: A Sound and Music Computing System*. Cham, Switzerland: Springer International Publishing; 2016. doi: 10.1007/978-3-319-45370-5
13. Lazzarini V, Jagwani A. CSOUND 7: Bare Metal and

- Co-Processing Hardwares. In: *Proceedings of the 19<sup>th</sup> Linux Audio Conference*. Lyon, France; 2025. Available from: <https://hal.science/hal-05096036>
14. Electro-Smith. *Daisy*. Available from: <https://www.electro-smith.com/daisy> [Last accessed on 2023 Jun 08].
  15. SparkFun. *SparkFun VL53L1X Arduino Library*. Available from: [https://github.com/sparkfun/sparkfun\\_vl53l1x\\_arduino\\_library](https://github.com/sparkfun/sparkfun_vl53l1x_arduino_library) [Last accessed on 2024 Apr 28].
  16. SparkFun. *Load Cell Amplifier HX711 Breakout Hookup Guide*. Available from: <https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide/all> [Last accessed on 2024 Apr 28].
  17. Arduino. *Wire Library Documentation*. Available from: <https://docs.arduino.cc/learn/communication/wire> [Last accessed on 2024 Apr 28].
  18. Xilinx. *PS UART*; 2023. Available from: <https://xilinx-wiki.atlassian.net/wiki/spaces/a/pages/18842340/ps+uart> [Last accessed on 2024 Feb 10].
  19. MaxBotix. *Arduino Ultrasonic Sensors*. Available from: <https://maxbotix.com/pages/arduino-ultrasonic-sensors> [Last accessed on 2024 Apr 28].
  20. Lazzarini V. *Spectral Music Design: A Computational Approach*. Oxford: Oxford University Press; 2021.
  21. Adafruit. *All About Stepper Motors: What is a Stepper Motor?* Available from: <https://learn.adafruit.com/all-about-stepper-motors/what-is-a-stepper-motor> [Last accessed on 2024 Apr 28].
  22. Arduino. *AccelStepper Library*. Available from: <https://www.arduino.cc/reference/en/libraries/accelstepper> [Last accessed on 2024 Apr 28].
  23. Precision Microdrives. *Vibration Motors*. Available from: <https://www.precisionmicrodrives.com/motors/vibration-motors> [Last accessed on 2024 Apr 28].
  24. ArduinoJson. *ArduinoJson Library*. Available from: <https://arduinojson.org> [Last accessed on 2024 Apr 28].
  25. Waqif A. *Chaal*. Available from: <https://asimwaqif.com/chaal> [Last accessed on 2025 Apr 20].
  26. DesignOwl. *Chaal by Asim Waqif: Art Installation Using Bamboo*. Available from: [https://www.youtube.com/watch?v=kmql\\_mw3m0i&ab\\_channel=designowl](https://www.youtube.com/watch?v=kmql_mw3m0i&ab_channel=designowl) [Last accessed on 2024 Apr 28].
  27. MDN Web Docs. *WebSockets API*. Available from: [https://developer.mozilla.org/en-us/docs/web/api/websockets\\_api](https://developer.mozilla.org/en-us/docs/web/api/websockets_api) [Last accessed on 2024 Apr 28].
  28. React. *React Documentation*. Available from: <https://react.dev> [Last accessed on 2024 Feb 16].
  29. Espressif. *mDNS API Reference for ESP32*. Available from: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/protocols/mdns.html> [Last accessed on 2024 Apr 28].
  30. Contributors F. *Fastify: Fast and Low Overhead Web Framework for Node*. <https://fastify.dev> [Last accessed on 2024 Feb 16].
  31. Turchet L, Essl G, Fischione C. Ubiquitous music and the internet of musical things. In: Lazzarini V, Keller D, Otero N, Turchet L, editors. *Ubiquitous Music Ecologies*. London: Routledge; 2020.
  32. Wiener N. *Cybernetics: Or Control and Communication in the Animal and the Machine*. United States: MIT Press; 1948.
  33. Von Bertalanffy L. *General System Theory: Foundations, Development, Applications*. New York: George Braziller; 1968.
  34. Koszolkó MK, Studley T. From site-specific sampling to gamification: An exploration of performative engagement with the environment. *Organised Sound*. 2023;28(3):338-351. doi: 10.1017/S1355771823000547
  35. Bridges B, Lazzarini V, Keller D. Editorial: Ecologically grounded creative practices and ubiquitous music: Interaction and environment. *Organised Sound*. 2024;28(3):321-327. doi: 10.1017/S1355771823000663
  36. TensorFlow. *TensorFlow Lite for Microcontrollers*. Available from: <https://www.tensorflow.org/lite/microcontrollers> [Last accessed on 2024 Apr 28].