aquaMIDI

Dominic Matos

MUS 141: Electronic Musical Instrument Design

December 16th, 2025

Introduction

Our group wanted to make a water-based instrument. At first, we tried to make a musical submarine, but this did not work. We pivoted to a different idea based on an embellishment we planned to use for our original submarine idea. We wanted "buoys" in a body of water to produce sounds when disturbed by water movement. The aquaMIDI is a two-person instrument that uses water to create a different aquatic ambience every time it is played. Creating this instrument required 3D printing, power tools, electronics programming, and computer software to convert position data into sound.

Materials Used:

1. PLA
2. Rubber O Rings
3. Metric M3 Hardware
4. Flex Seal
5. Arduino Nano BLE 33
6. ESP 32 + IMU
7. Portable 5V batteries
8. Micro USB cables
9. Copper Wires
10. Food coloring
11. Rubber ducks
12. Wooden dowels
13. Brass rods
14. Plastic bags
15. Plastic tank
16. Silicone molding materials

Design Goals

1. Create ambient music reactive to water motion.
    a. IMUs are used to detect and send position information.
        i. Each IMU corresponds to a unique sound
        ii. Sounds reflect the intensity of water movement
    b. Axes values received in MAX
    c. MAX values treated and sent to Reason
2. Wirelessly communicate data between IMUs and computer software
3. Construct waterproof, floating buoys
    a. House IMUs inside Buoys
4. Map axes of IMU to parameters of sound modules
5. Keep water mess minimal

Mechanical Design

The finalized key, or “wave-pusher”, design took many iterations. It was initially designed as a circle with linkages to convert the vertical motion of pushing down on it to horizontal motion, displacing water. After iterating, a more straightforward yet still effective key wave-pusher design was finalized.

There are a total of 10 keys, 3D-printed and joined to brass cylindrical rods (Figure 1), which form our wave-pusher system. The brass rods were sourced and then machined in Bray using the vertical bandsaw and an automated sander. The aquaMIDI is intended to be played by two people, therefore there are five keys on each side of the tank for each player. Silicone is molded onto the buoy-facing ends of the brass cylinders to reduce the sound of the rods hitting the buoys. The molding was done with Let's Resin silicone. We sourced a mastermold (a plastic cup), preparing the Mastermold with isopropyl alcohol, placing the brass cylinders in the cup, preparing the silicone (precise weights and mixing), pouring the silicone into the plastic cup, letting the silicone cure for 24 hours, then isolating each cylinder within the mold (Figure 2).

The key system is mounted to the tank using wooden dowels and couplers (Figure 1). The couplers restrict motion in one axis to ensure the keys are stable, while the brass rods move through the water. The wooden dowels were mounted to the tank using 3D-printed mountings with holes for screw attachments. Holes were drilled into the tank to make this connection.

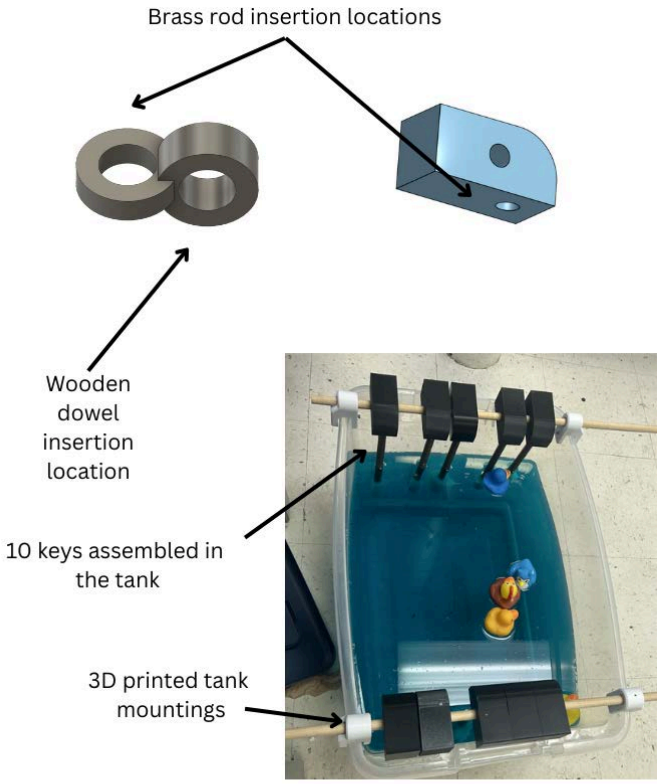


Figure 1: Key-wave maker assembly components

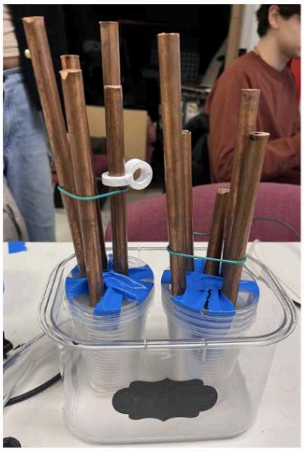


Figure 2: Silicone molding process

Each buoy needed to be wireless, so they were designed to hold a small battery pack and other electronic components underwater. In addition to being waterproof, the buoys needed to stay upright despite the water disruption to keep the IMU from going underwater. Bluetooth data cannot be transmitted underwater. To address this issue, the initial buoy model featured a circular platform at the waterline, with internal architecture ensuring that the 5V battery is below the platform and the is IMU above it (Figure 3). The initial buoy design did not float sufficiently. After testing, the team decided the buoy was sinking because its center of mass was too low. The final buoy design did not have the circular platform, as it was not serving its purpose and the buoys were going too far underwater (Figure 3). The final design floated but was not waterproof, and it did not remain upright. After attempting to anchor the buoys to the bottom of the tank and trying many combinations of weights in their bottoms, it was decided it was okay that they were not upright because they were floating and the IMUs remained above the waterline. We took many precautions to ensure that our buoys were water-tight. First, we created a tight seal between the buoy's lid and body using an O-ring. This ensured that both 3D printed parts sat flush against each other and did not allow any water into the buoy. Additionally, we applied the adhesive Flex Seal to create a waterproof seal around the 3D print and to patch any microholes inherent to a 3D-printed design.

Initial vs final buoy design

Final buoy design disassembled for IMU and battery insertion

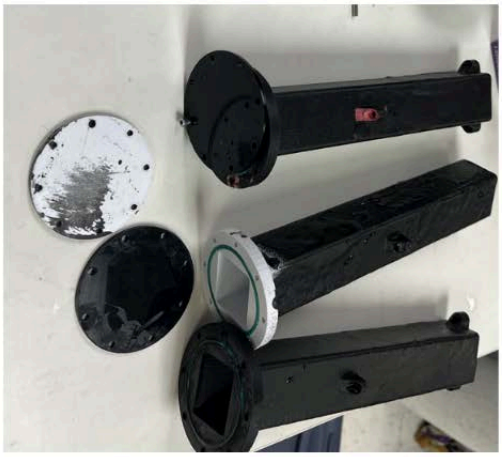


Figure 3: Buoy design assembled with and without the circular platform. Buoy design in its dissembled state with O-rings secured and Flex Seal applied.

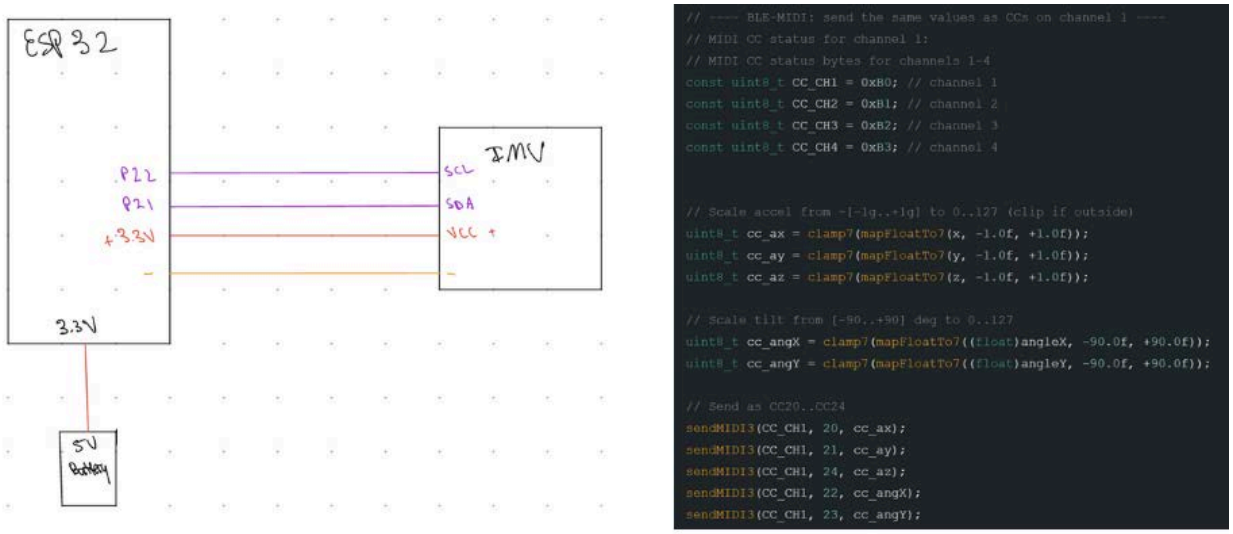
Electrical Design

Two electronic components were utilized to communicate the buoy motion to the MAX Patch. Both components run on 3.3V, but can be powered by a 5V battery as was done in this project. The parameters detected by the IMU in the Nano BLE and the ESP32 include acceleration in the x, y, and z, directions.

The first is a battery-powered Arduino Nano 33 BLE, which has an embedded IMU and Bluetooth capabilities. The Nano and battery were inserted into the buoys and, as long as it remained above water, could effectively communicate its accelerometer data with the Max Patch. All Arduino programming was done using a simplified version of C++. Some code was taken from datasheets and adapted to fit the needs of this project. For example, the Bluetooth and

accelerometer data collection was configured using the Nano data sheet, however, the code was modified (controller and channel numbers) to output the accelerometer data to MAX.

The second is a battery-powered ESP32 with Wi-Fi capabilities and an IMU. The circuit diagram for this system is shown in Figure 4. The ESP32 also has three outputs that are directly connected to the MAX patch, making them more accessible. Thus, it is not necessary to define channels and controllers in the ESP32 code. The ESP32 code was found here, then modified.




```
// ---- BLE-MIDI: send the same values as CCs on channel 1 ----
// MIDI CC status for channel 1:
// MIDI CC status bytes for channels 1-4
const uint8_t CC_CH1 = 0xB0; // channel 1
const uint8_t CC_CH2 = 0xB1; // channel 2
const uint8_t CC_CH3 = 0xB2; // channel 3
const uint8_t CC_CH4 = 0xB3; // channel 4


// Scale accel from -[-1g..+1g] to 0..127 (clip if outside)
uint8_t cc_ax = clamp7(mapFloatTo7(x, -1.0f, +1.0f));
uint8_t cc_ay = clamp7(mapFloatTo7(y, -1.0f, +1.0f));
uint8_t cc_az = clamp7(mapFloatTo7(z, -1.0f, +1.0f));

// Scale tilt from [-90..+90] deg to 0..127
uint8_t cc_angX = clamp7(mapFloatTo7((float)angleX, -90.0f, +90.0f));
uint8_t cc_angY = clamp7(mapFloatTo7((float)angleY, -90.0f, +90.0f));

// Send as CC20..CC24
sendMIDI3(CC_CH1, 20, cc_ax);
sendMIDI3(CC_CH1, 21, cc_ay);
sendMIDI3(CC_CH1, 24, cc_az);
sendMIDI3(CC_CH1, 22, cc_angX);
sendMIDI3(CC_CH1, 23, cc_angY);
```

Figure 4: Circuit Diagram for ESP32 with the IMU and the code used to send accelerometer IMU data from the Nano to Max

Max Patch

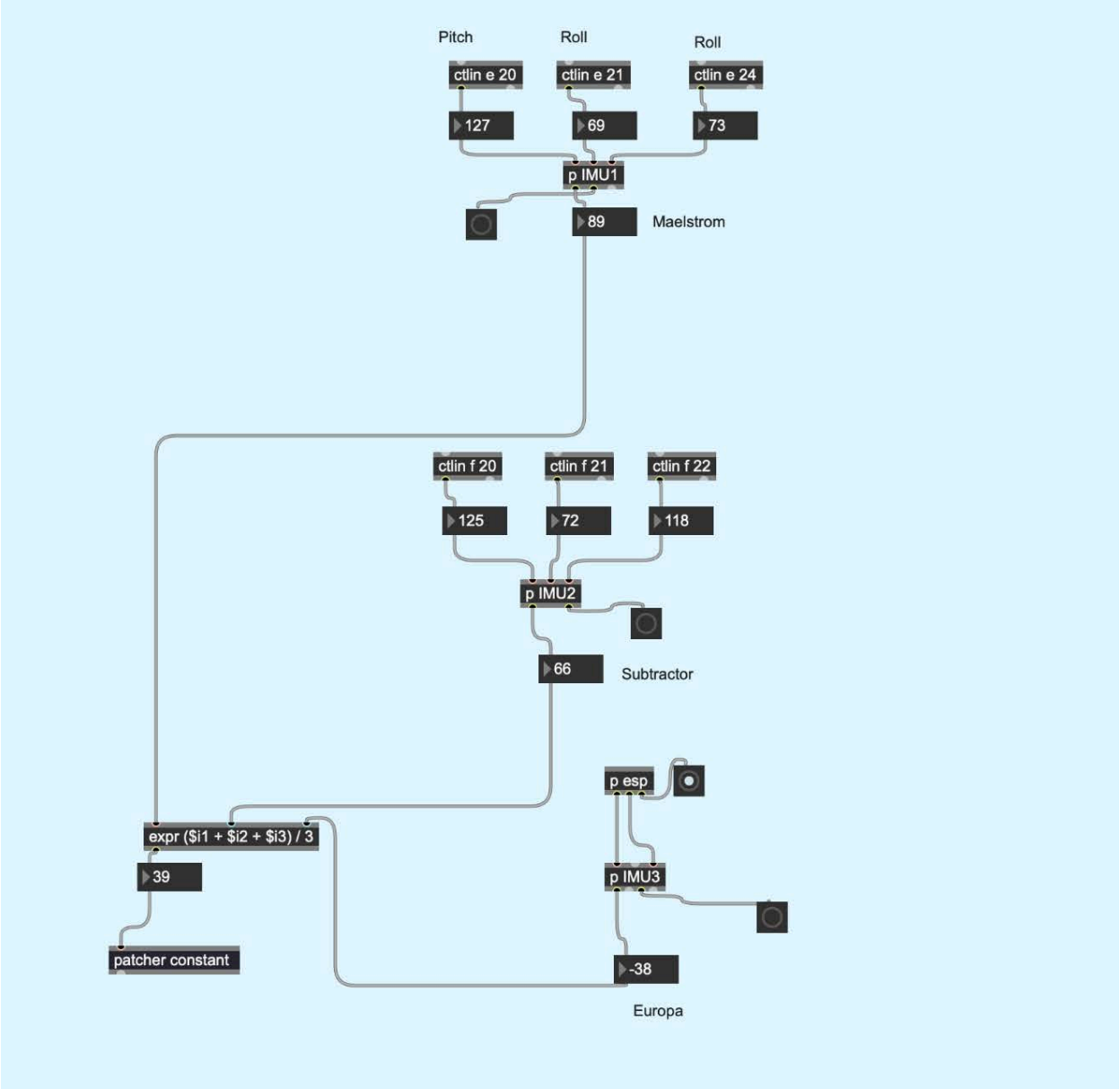


Figure 5. Base Patch

The Max Patch controlled each IMU separately through subpatches (Figure 5).

I used **ctlin** objects to receive the MIDI data from the Arduinos and assigned each Arduino to a symbol (e, f, g). The Max Patch received x-, y-, and z-acceleration data from each Arduino via channels 20, 21, and 22/24.

The third Arduino was damaged in the process, so I replaced it with an ESP32. This sent its data over wifi and UDP instead of Bluetooth and MIDI. (Figure 6)

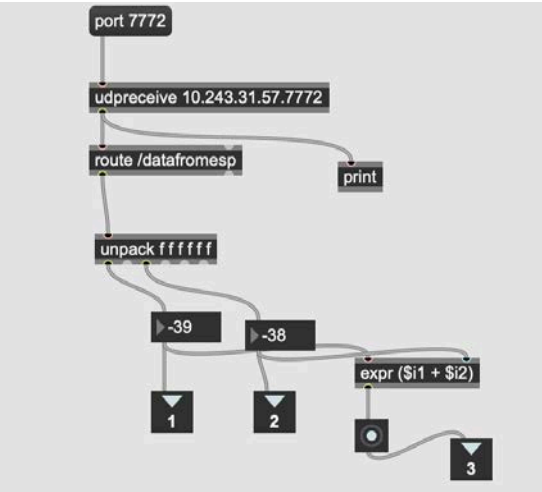


Figure 6. ESP32 Subpatch

Each IMU subpatch had the same layout. The values of each axis were scaled as needed for each parameter, and smoothed using a **slide** object. (Figures 7, 8, and 9).

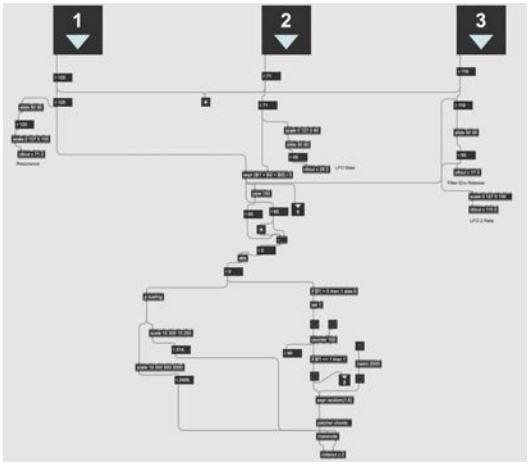


Figure 7. IMU 1

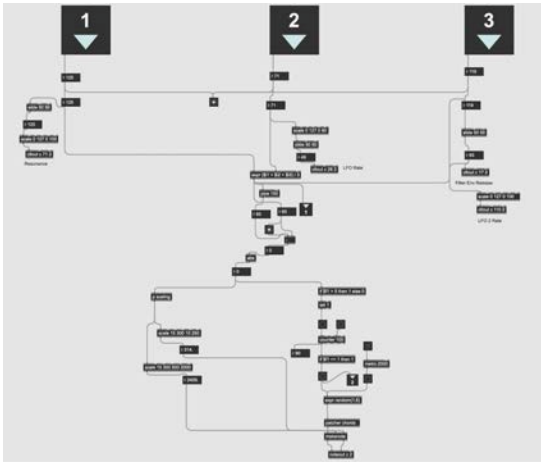


Figure 8. IMU 2

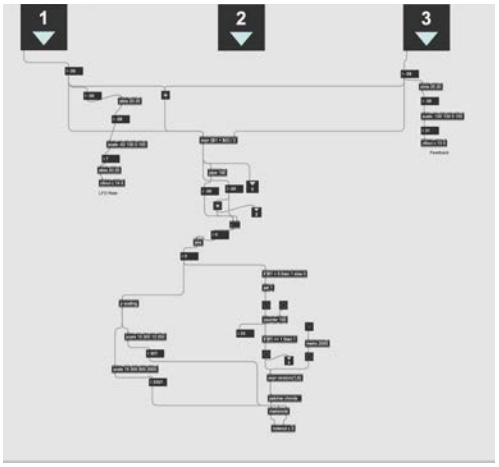


Figure 9. IMU 3

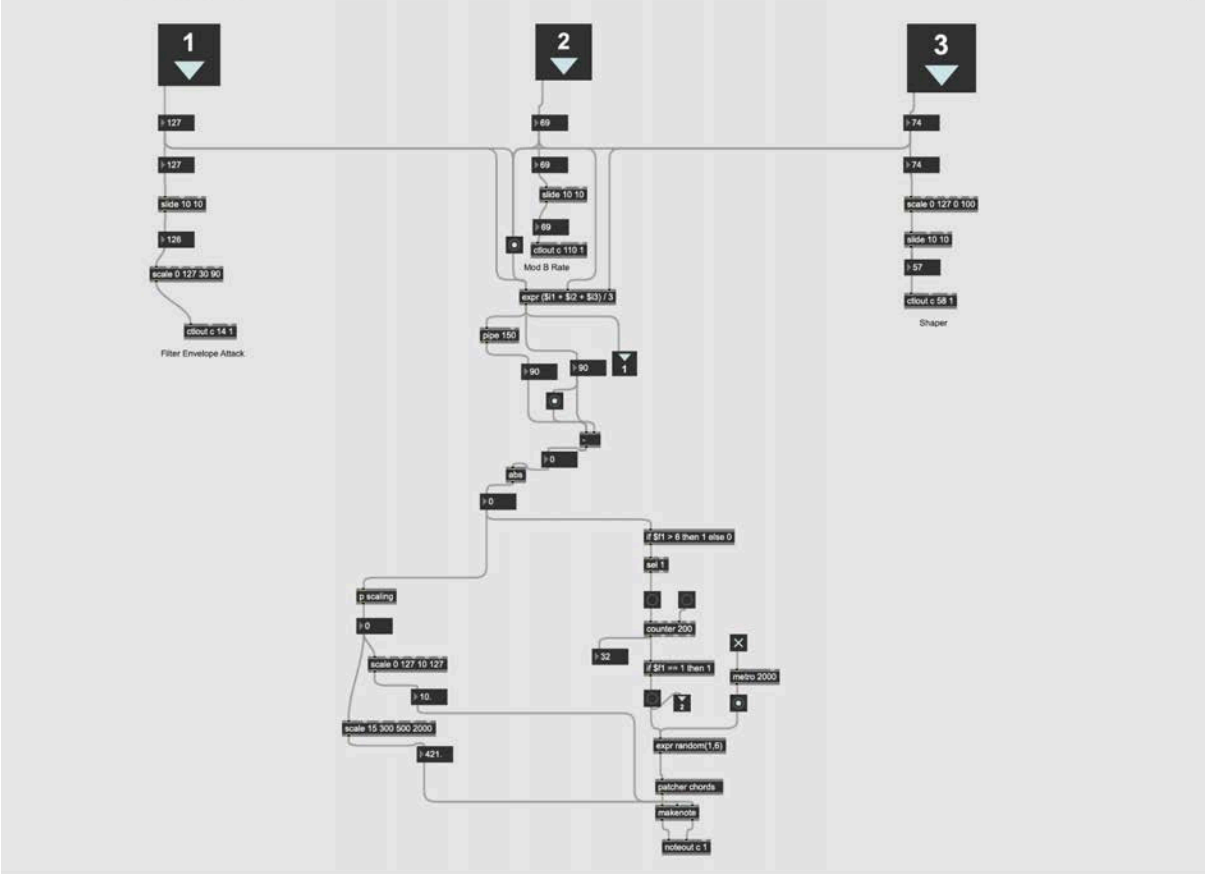


Figure 10. IMU1 Enlarged for clarity

These values were then sent to two locations: their respective **ctlout**s to control different parameters of the reason module, and to the **expr** object to trigger notes.

The **expr** object takes all three values from the IMUs, averages them, and sends the outcome to two number objects, one of which is delayed with a **pipe** by 150ms. These two values are then sent to a subtract object (**-)** and then to an absolute value object (**abs)**. This system continuously checked for changes in any IMU axis by comparing the current value with the previous value. The **abs** object ensured that the values stayed positive.

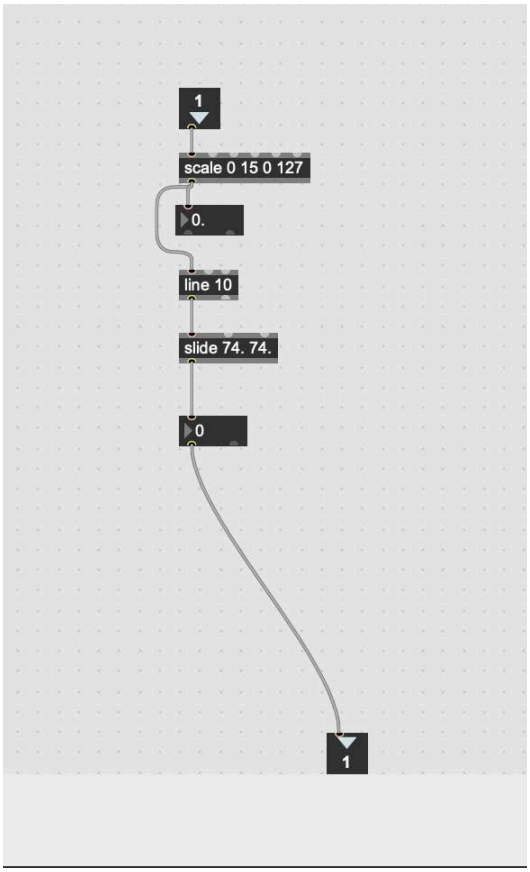


Figure 11. Scaling subpatch

The value is then sent to two locations: to noteout velocity and duration, and to threshold logic to create the notes.

Before reaching the noteout velocity and duration, the values are sent through the scaling subpatch (Figure 11), which applies a series of smoothing operations using line and slide to stabilize the values.

The threshold logic was our solution to the constant water movement (Figure 12). I initially set the threshold to a high number (~10-15) so that the noise would only play when the IMU was moved enough, but I realized I wanted even small water movements to produce sounds as well, just quieter ones. I lowered the threshold, but quickly realized that our system outputs a bang every time it receives a value larger than the threshold, and because our values were being updated every 150ms, they were stepping through almost every number, so if our threshold was 15 and it received a value of 30, it would output about 15 bangs and 15 noteouts, almost simultaneously. This resulted in unfavorable sounds. To solve this problem, I used a **counter** and an **if** statement to limit the bangs to one bang output every 100 bang inputs. This bang triggered one of nine random chords to be sent to Reason through the chords subpatch (Figure

13). The chords were chosen from the C major scale, including dissonant chords to keep it interesting. Using long, drawn-out chords instead of individual notes was what made our aquatic/spatial atmosphere possible. The chords evolved over time and had long attacks, reflecting the smooth movement of water.

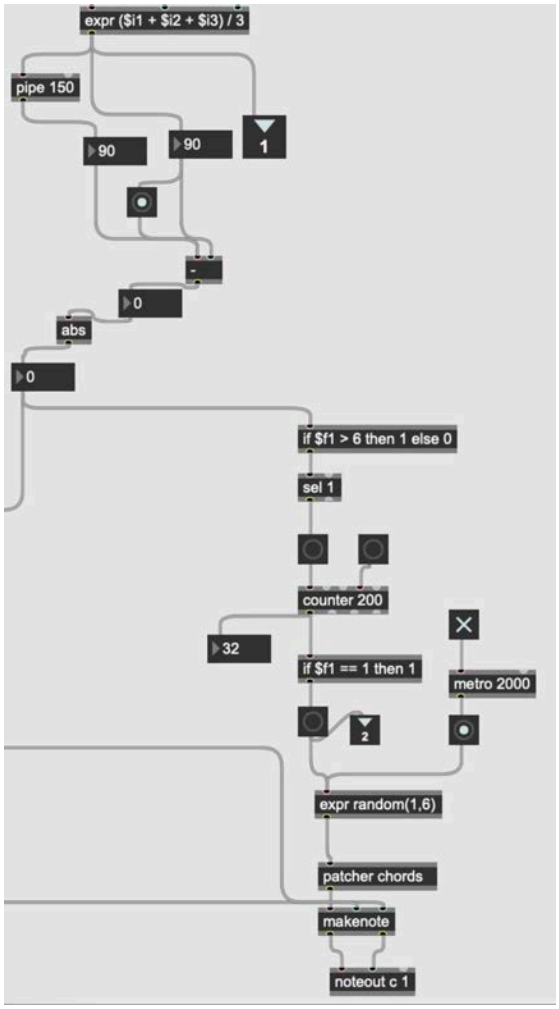


Figure 12. Threshold Logic (toggle and metro for debugging purposes)

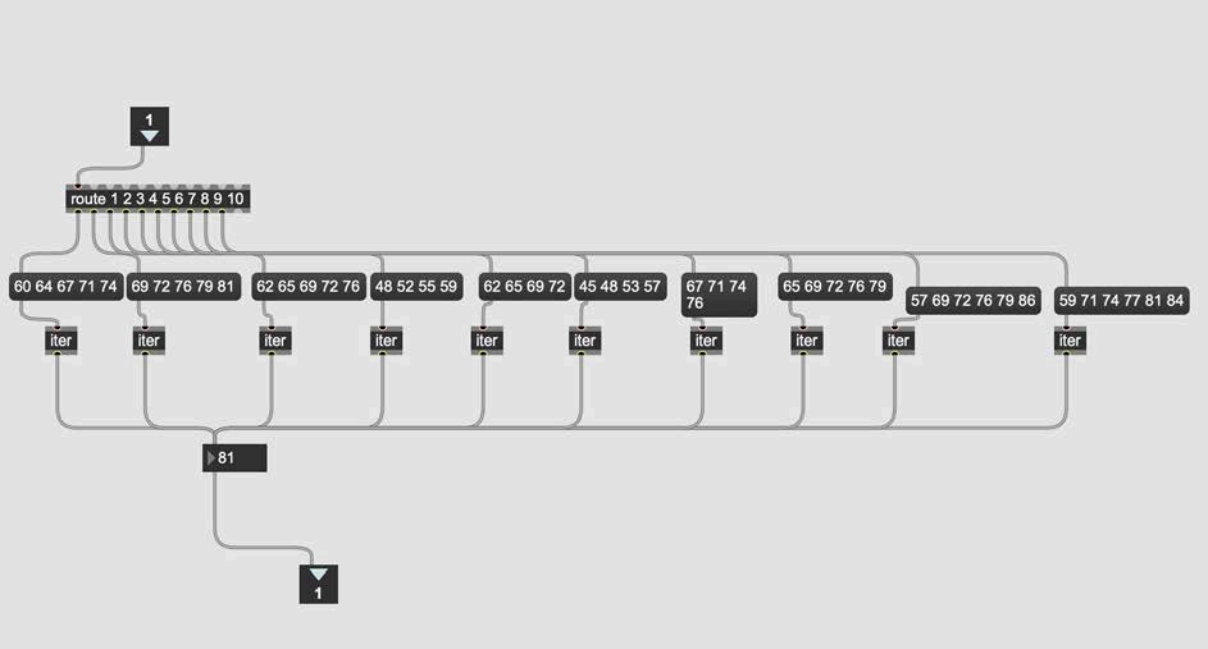


Figure 13. Chords subpatch

IMU1 sent MIDI Data to a Malström Graintable Synthesizer module in Reason on Channel 1, with the X, Y, and Z axes controlling Filter Envelope attack, Modulation B rate, and Shaper amount, respectively.

IMU2 sent MIDI Data to a Subtractor module in Reason on Channel 2, with the X, Y, and Z axes controlling Resonance, LFO 1 rate (controlling mix between Osc 1 & 2), and Shaper amount and LFO 2 rate (controlling amplitude), respectively.

The ESP32 sent MIDI data to a Europa shapeshifting synthesizer on Channel 3; we were only able to get two functional axes from the UDP, being X and Z.

We wanted a base sound playing at all times, inviting the user to play the AquaMIDI, and this was an NN-19 Sampler. This was supposed to be a simple sound, but I wanted it to reflect the activity in the AquaMIDI, so I used the average of the average of each IMU's axes for the note's velocity and duration. This constant sound was contained in a subpatch using the same pipe-delay method as the main patches (figure 14). If any of the IMU values change, a bang is sent to a timer. If the time since the last bang exceeds 100,000ms, the timer sends a bang to a metro, triggering the sound. This made the sound automatically turn on the first time we used it each day. The MIDI messages for the constant sound were sent over Channel 4.

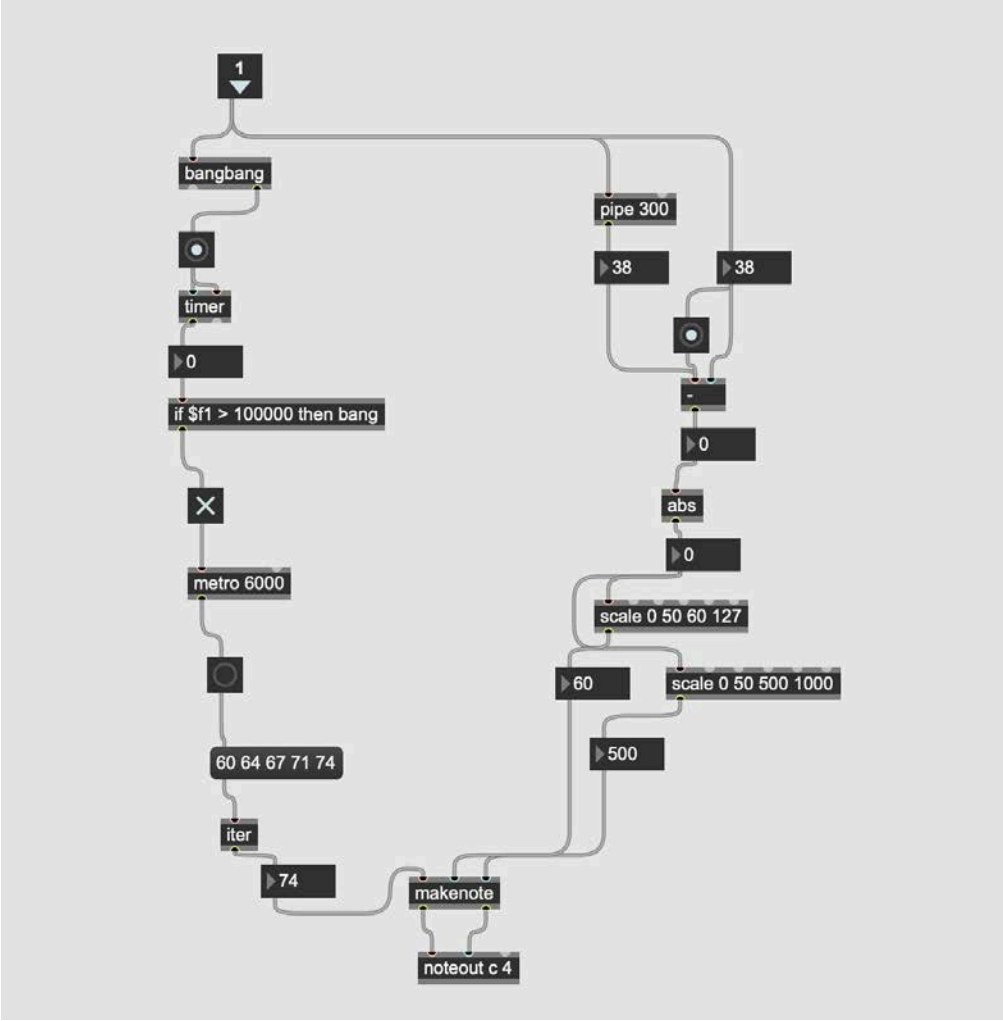


Figure 14. Constant sound subpatch

Reason

To enhance the water-based quality of our project, the mixer each instrument was being sent to was routed through an RV7000 Mk II Reverb module before reaching the master output. This gave a more ambient impression and enforced our original idea from the submarine: “what does music sound like under water?” The philosophy for the sound design in this project was to create sounds that resemble the feeling of floating in water. Sometimes big waves come by, lift you up momentarily, and then you return to your resting state. We wanted it to be calm and relaxing, but for each sound to still be distinct.

Figure 15. Reverb Module

Malstrom

The Malstrom synthesizer was producing a loud, low-end sound at certain parameter values, so it was routed through a PEQ-2 instrument before going to the mixer, high-passing the sound at around 150Hz.

Figure 16. Malstrom module and EQ module

Europa

Our main issue was the Europa synthesizer's lack of targetable parameters. There is no documentation on the MIDI controller numbers for Europa, and I could not get the standard MIDI controls (Expression, Breath) to work in Europa either. I still wanted to be able to manipulate the sound of this synthesizer, so I opted for a CF-101 chorus/flanger, an instrument I knew had documentation. The Europa was sent through the CF-101 before being sent to the mixer, and the X and Z axes of the Esp32 controlled the Feedback and LFO rate, respectively. The CC messages were sent to the CF-101 on channel 5.

Figure 17. Europa module and CF-101 Module

Subtractor

The Subtractor patch featured four changing parameters rather than three to make the sound change noticeable. The filter envelope release was chosen as an incentive to hit the buoys harder; if the received value was low, the filter would have a very short release time, and the sound wouldn't play for long.

Figure 18. Subtractor Module

NN19

The constant sound was triggered every six seconds, and was a single C-major chord with a triangle-wave LFO controller the filter frequency, creating a calm, up-and-down wave sound. I wanted at least some change, but the NN19 only has one LFO, so I used a Pulsar instrument to create a slow LFO that would lightly modulate the NN19's resonance.

Figure 19. NN-19 Module and Pulsar Module (LFO 2 On but not being used)

Conclusion

Overall, our project was a success. We accomplished all of our design goals in one form or another. If I were to do this project again, I would use a larger tank or smaller keys so that the players of the instrument could have more control of the sounds, and attempt to play something intentional. Ideally, a player could try to target one specific buoy if they wanted to play the sound from that buoy, and use this to play music. The instrument functions as an ambient sound creator and is fun to use.

My groupmates all contributed heavily to the project and did their part. I commend everyone's work and their ability to utilize their strengths and expertise as well as acknowledging their weaknesses when working together.