

MaestroMotion: A Gesture-Controlled Interactive Audio Installation

Author	Qinxuan (Leslie) Li
Date	@March 18, 2025
University	Imperial College London
Degree	(MEng) Design Engineering
Module	Audio Design Engineering

Concept Ideas & Main
- **Technology Stack**
- Work Schedule
  - Work I’ve done:

Work Packages
- WP1 - Concept Research
- WP2 - Motion Captures & Hand Tracking
  - Motion Tracking
  - Methods for Receiving Information in Max/MSP from Python
- WP3 - Multi-track Control
  - **Key Features**
  - The MaxMsP File
- WP4 - 3D Sound Spatialisation
  - Tracks Input & 3D Visualisation
  - Gesture Control
  - **Reference Frame Issue & Solution**
  - **Observations & Challenges**
  - **Reflections**
  - MaxMSP File & Interface
- WP5 - Filtering
  - **Filter Types & Functionality**
  - MaxMsp File & Visualisation effect
- WP6 - Generative Drum
  - Part 1: Drum Synthesis
  - **Part 2: Single hand motion-Tracking**
  - Part 3: Two Hands Motion-Tracking
  - **Part 4. Gesture-Based Beat & Groove Control**
  - **Conclusion & Future Refinements**
  - MaxMsp File

WP7 - Merge & UI
Conclusions, Potential & Future Work
Challenges & Areas for Improvement
Key Achievements & Learnings
Future Enhancements & Potential Expansions

https://youtu.be/Fj06jwNn5ml?si=xvnW9hXThcg3Nq2L

Concept Ideas & Main

MaestroMotion is an **interactive audio learning sandbox** powered by **computer vision**, designed to make **audio education more accessible** for school students. By enabling **physical interaction** with sound, it eliminates the need for expensive audio equipment.

The sandbox consists of **modular learning experiences**, each focused on different aspects of **audio technology and live event sound design**. From this foundation, we developed four **core modules** for **Maestro Motion**:

- **WP3: Multichannel Mixing (Haozhen & Tharindu):** Instead of relying on traditional mixers, students use **hand gestures** to control individual instrument levels, learning **real-time audio balance** techniques. *(Haozhen explains band mixing.)*
- **WP4: Spatialisation (Miaoyan):** Students dynamically position **sound sources in 3D space**, gaining an **experiential understanding of spatial audio** and its impact on **perception and immersion**.
- **WP5: Filtering (Liberty):** A **low-pass, high-pass, and notch filter** system lets students **manipulate frequencies in real-time**, bridging the gap between **audio theory and live performance**. Hand gestures control frequency cutoff and resonance across different axes.
- **WP6: Generative Drumming & Visualisation (Qinxuan & Vihan):** Drumming is a **natural response to music**, so we created an **interactive rhythm module** that teaches **drum types, beat structures, and timing**. A **visual representation of rhythm** enhances learning through **real-time feedback**.

Technology Stack

To achieve this **seamless interaction**, we integrated:

- **MediaPipe** for **gesture tracking**
- **Python** for **data transmission**
- **MaxMSP & Jitter** for **audio synthesis & visualisation**

By combining these **modules and technologies**, **Music Maestro** provides a **scalable foundation** for **expanding audio education**, offering **endless possibilities** for future development.

The input audio files contains the basic musical composition (bass, guitar, keys, vocals), with the drum track intentionally removed since we will generate our own drum sets

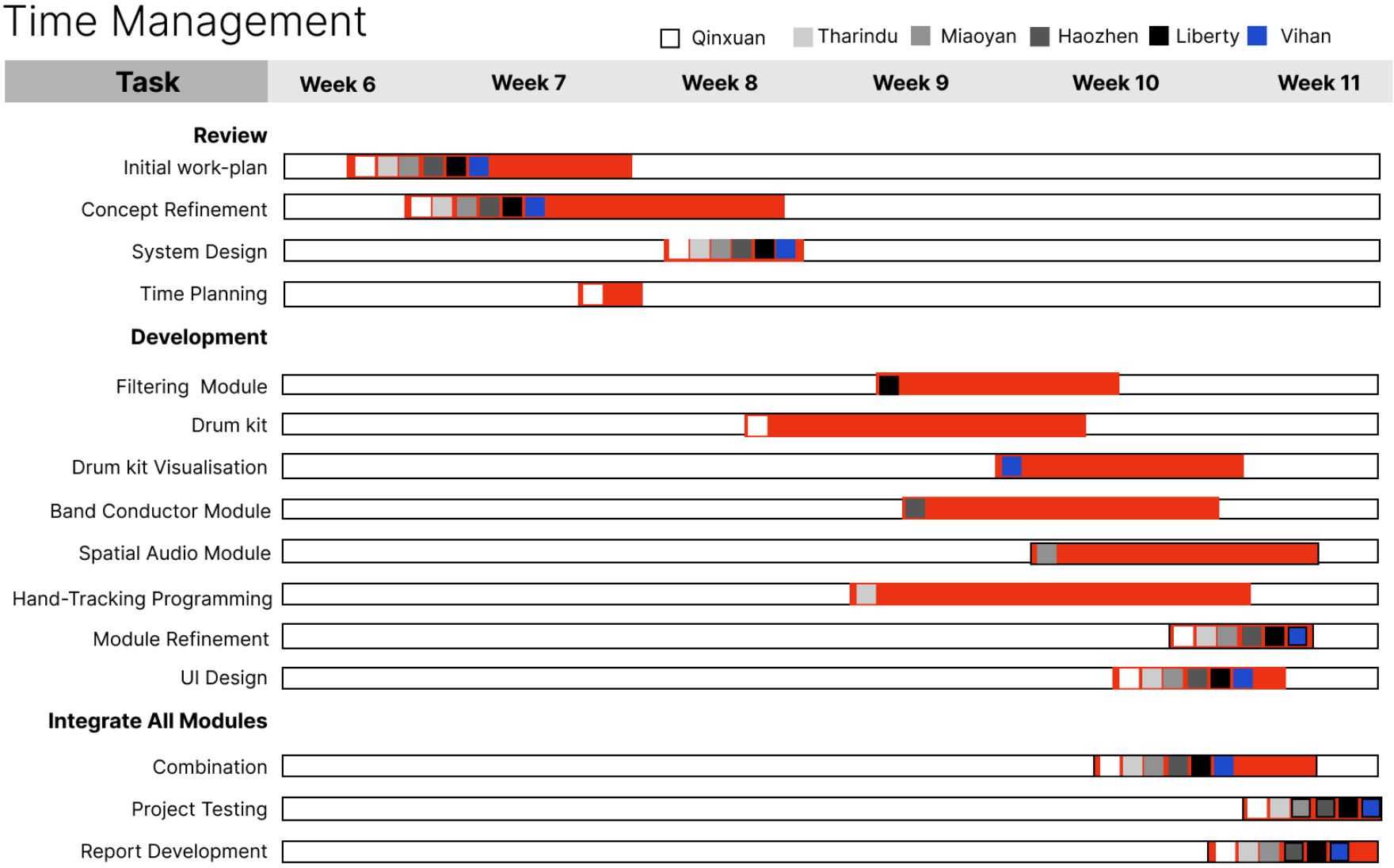
Work Schedule

Work I’ve done:

- **Developed generative drum synthesis**, creating realistic kick, snare, and hi-hat sounds.
- **Implemented Python-to-MaxMSP communication**, enabling seamless data transfer.
- **Designed bidirectional data transfer**, converting object coordinates from the secondary to the primary reference frame and sending the corrected data back to MaxMSP.
- **Created the final UI**, ensuring clarity and usability.
- **Developed generative algorithms for beat generation**, allowing dynamic rhythm creation.
- **Led project coordination**, organising meetings and structuring the workflow.

Project Time line

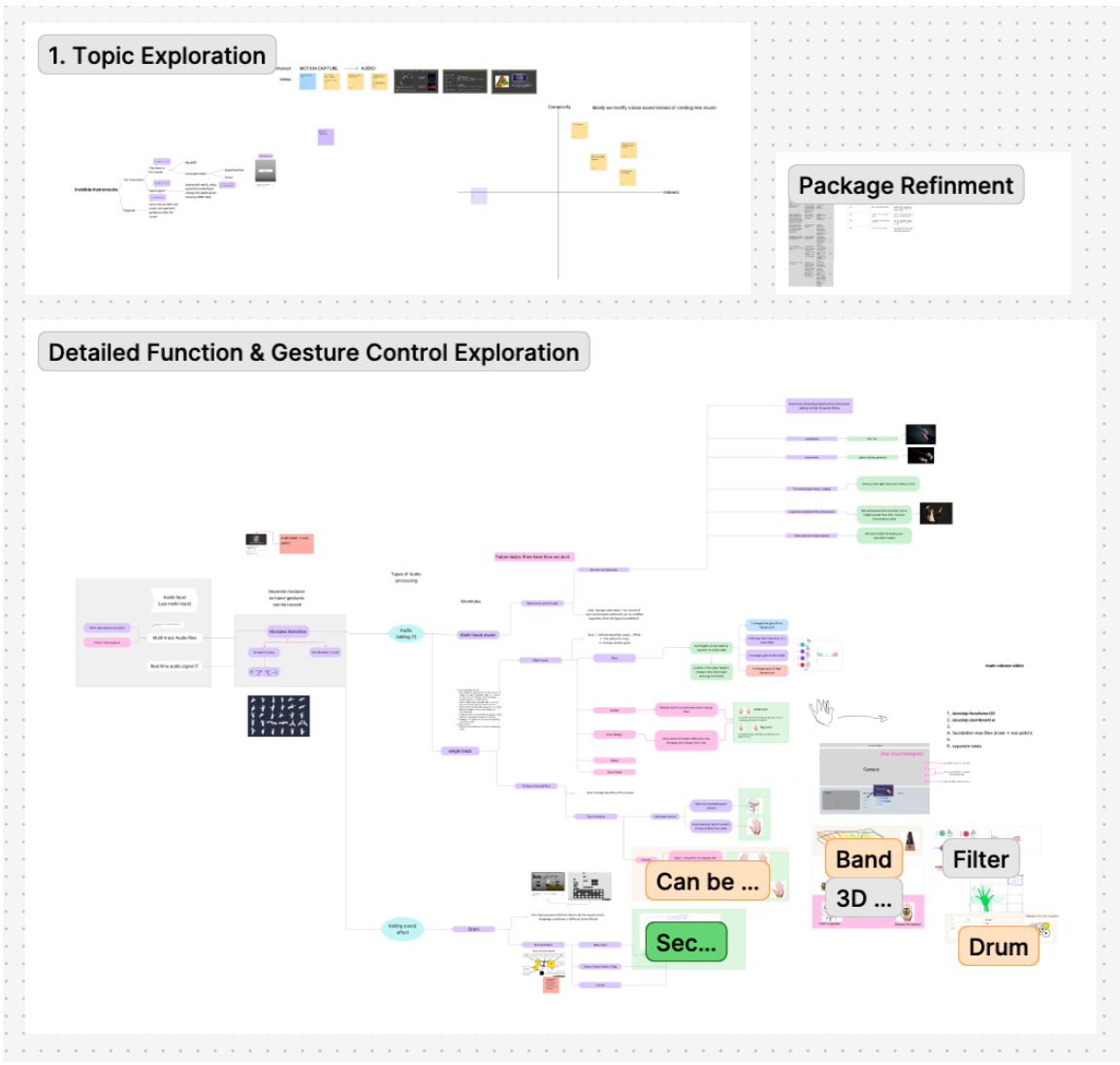
Time Management



Work Packages

WP1 - Concept Research

This phase aligns our **initial plan—motion tracking and multitrack audio processing**—with the **marking criteria**. We assessed team strengths and interests to define **work packages** and explored possible **gesture mappings** for intuitive **motion-to-audio control**. Our research focused on ensuring logical interactions between **motion capture** and corresponding **audio functions** for clear user understanding.

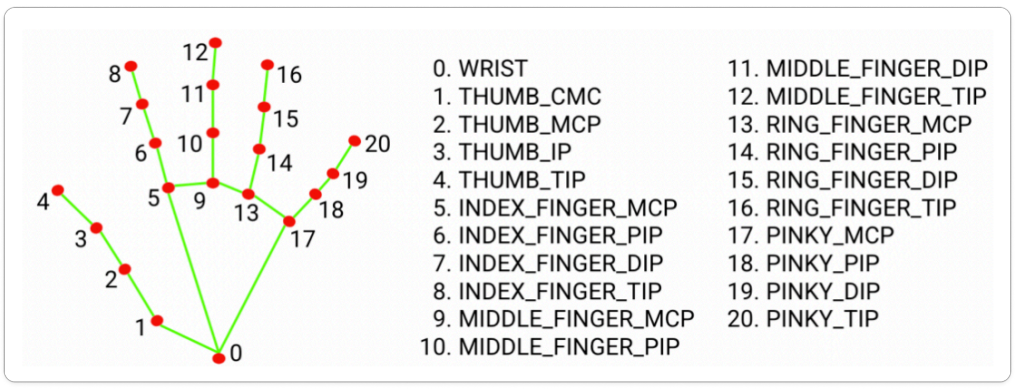


Click to access the Figjam: https://www.figma.com/board/DP2p47ZFD8j1mFAs1O7WNZ/Audio-Engineering?node-id=0-1&t=XBj5iQAVA5G9OrpL-1

WP2 - Motion Captures & Hand Tracking

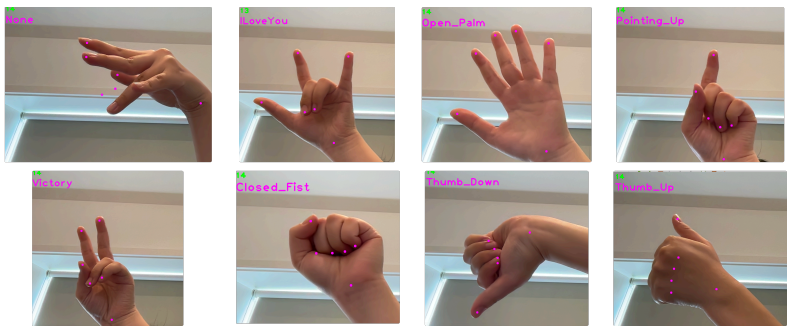
Motion Tracking

The system tracks **all hand joints**, capturing their **x (left-right), y (up-down), and z (front-back)** positions **relative to the laptop's camera**. This allows for precise motion-based interaction with audio parameters. In addition to hand joint tracking, it also provides **gesture recognition**, enabling the use of different gestures to activate various functions.

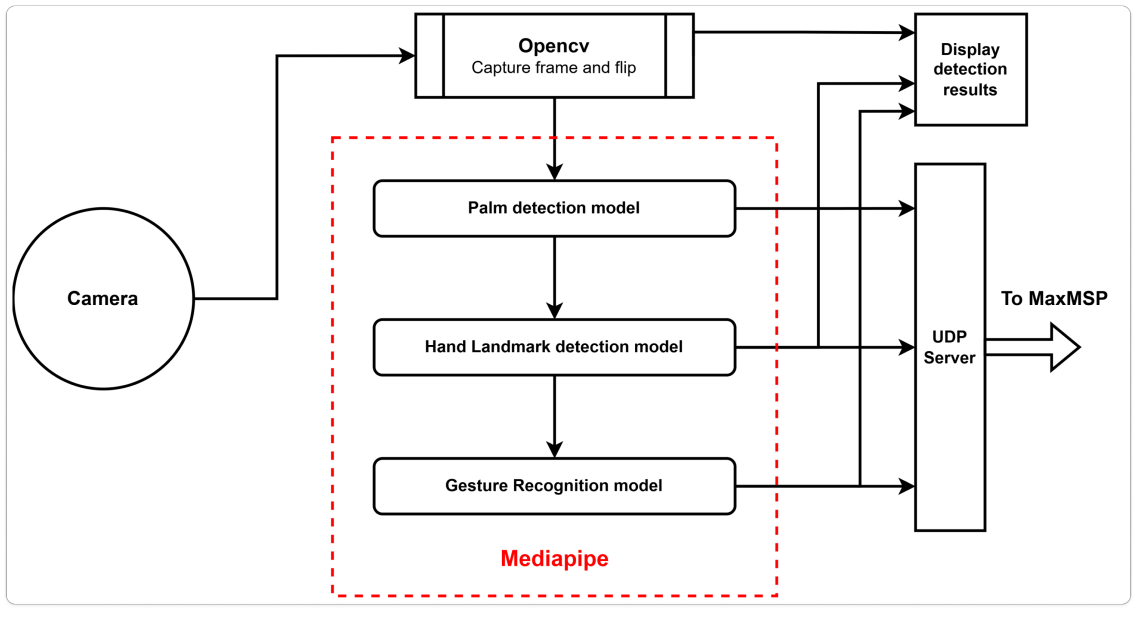


hand joints detected (click to see mediapipe doc)

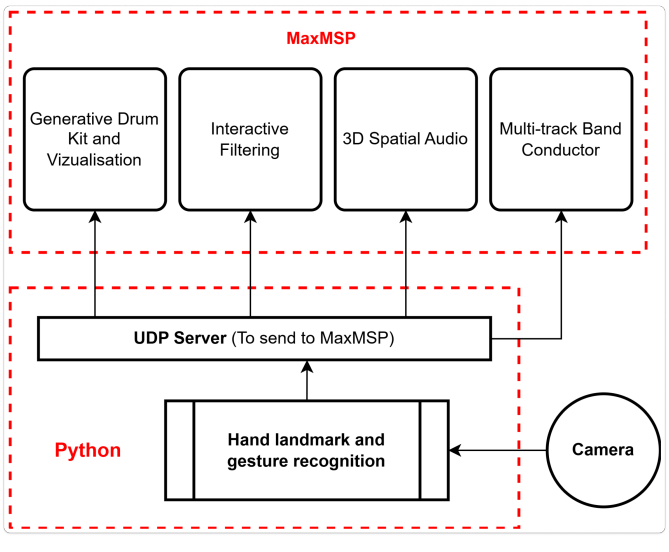
Available hand gestures: "None" (gestures that cannot be detected), "Closed_Fist," "Open_Palm," "Pointing_Up," "Thumb_Down," "Thumb_Up," "Victory," and "ILoveYou."



The basic logic of the motion tracking system is shown as below. To get the details of the code, pls visit 🔗 **GitHub Repository and also see Tharindu's report.**



Computer vision system overview



System overview

Methods for Receiving Information in Max/MSP from Python

In the python report, to enable communication between **Python (MediaPipe) and Max/MSP**, we send **hand tracking data** via **UDP** using:

```
ip = "127.0.0.1"
port = 7400
```

On Max/MSP, we receive the data using:

- **Python:** `send` → `SimpleUDPClient.sendmessage()`
- **Max/MSP:** `udpreceive 7400`

This setup ensures seamless **real-time motion tracking and audio interaction** across platforms.

WP3 - Multi-track Control

This module enables **gesture-controlled mixing** of a **five-track band system**, integrating **Max/MSP** with **Python-based gesture recognition**. Users can **select tracks and adjust audio parameters in real time** using **hand gestures**.

Key Features

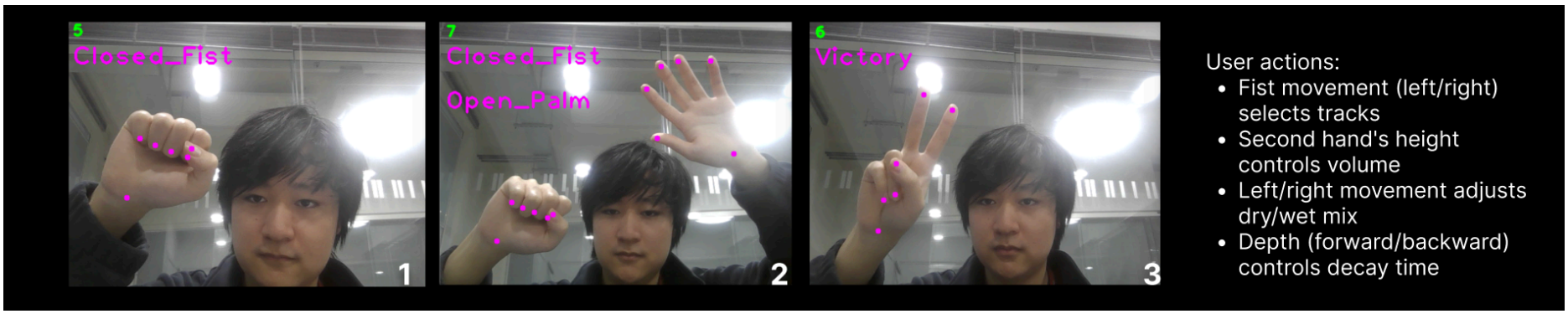
Track Selection - (First hand detected)

- A **fist gesture's x-coordinate** dynamically selects one of four tracks.
- Selection is confirmed upon **gesture release**.

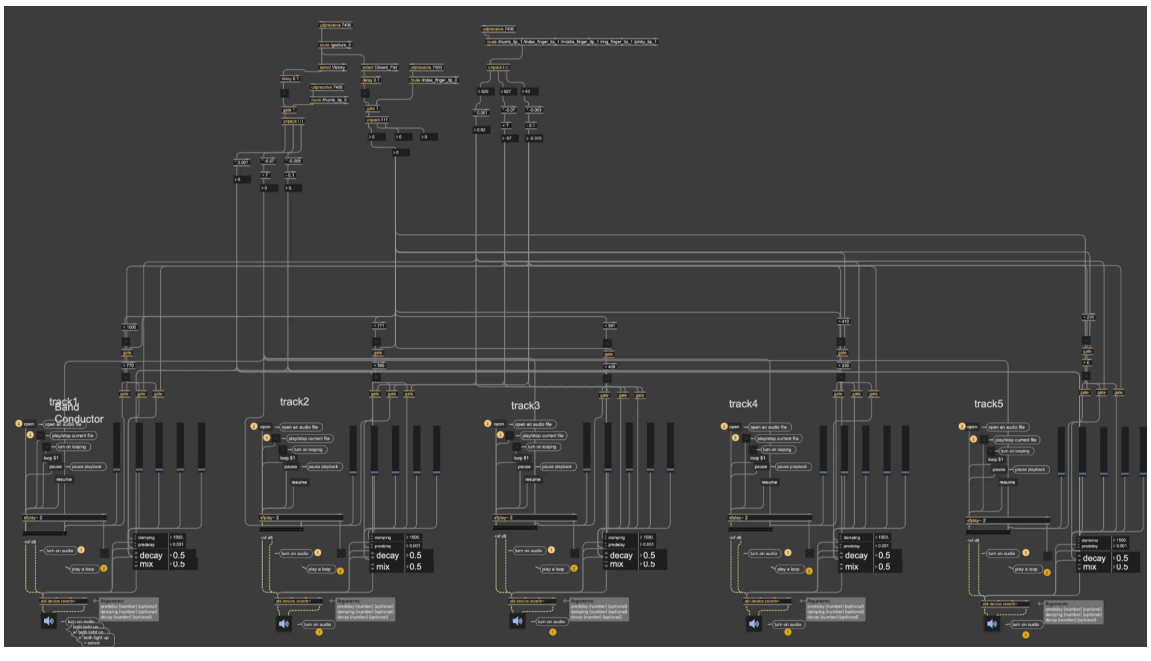
Real-Time Parameter Control - (Second hand detected)

Once a track is selected, the **index finger of the second hand** is used to modify key parameters:

- **Moving Up/Down (y-axis)** → Adjusts **Volume** (`@live.gain~`) from **60 dB (down) to +7 dB (up)**.
- **Moving Left/Right (x-axis)** → Adjusts **Dry/Wet Mix** (`@mix`), where **left = dry (0), right = wet (1)**.
- **Moving Forward/Backward (z-axis)** → Adjusts **Decay Time** (`@decay`), where **closer = shorter decay, further = longer decay**.



The MaxMsP File



WP4 - 3D Sound Spatialisation

This patch implements **gesture-controlled spatial audio positioning**. Users control the **x, y, z coordinates** of an **audio source** using **hand tracking**. However, an issue arose due to **misalignment between reference frames** in Max/MSP's **3D visualisation environment**.

Tracks Input & 3D Visualisation

Max Patch

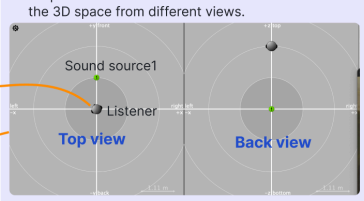
The patch for the Spatial Audio module includes 3 parts: Tracks input, 3D Visualise and Hand coordinates input

Part1: Tracks input

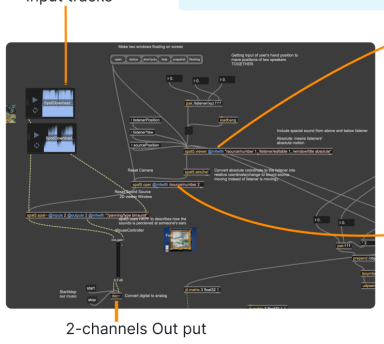
The aiming of this part is to input sound track and to find out how does the sound source move relative to listener through hearing the effect using **2-channels headphone output.**

Here, we enable the listener's orientation (the virtual 3D space camera) to rotate(which is implemented in later section), while keeping the sound source fixed at a specific position in the 3D space.

To test how do the sound sources move, the window is opened to see the movement of sound sources in the 3D space from different views.



Input tracks



In this window, we adjust some 3D sound effect(eg. the sound dropping was increased to make sound level drops more obvious when moving away from the user).



2-channels Out put

Part2: 3D Visualise

After ensuring that the sound sources function correctly within the 3D audio environment, we attempted to **visually represent their positions** in 3D space. This section of the patch serves precisely that purpose.

This is used to receive the **hand coordinates from the next section** (Hand coordinates input) to individually adjust the coordinates of the two sound sources.

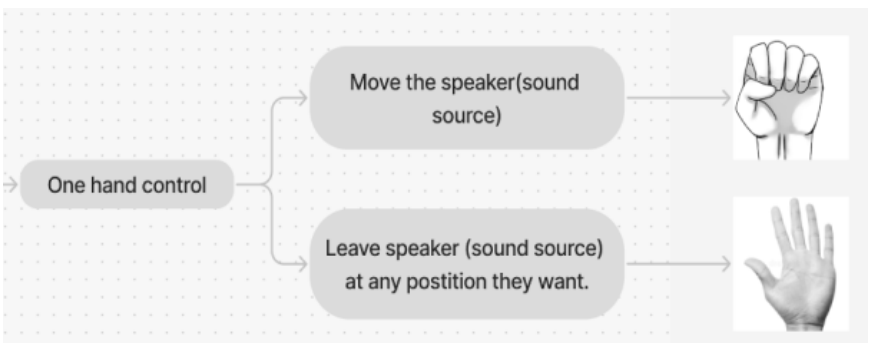
Generate a virtual 3D space and import a rendered background image.

Generate sound source1

Generate sound source2



Gesture Control



Reference Frame Issue & Solution

The issue arose when **camera rotation caused incorrect spatial positioning** of the **audio source**. My teammate encountered this problem while working on spatial audio placement, and I joined to analyse the cause with her.

Identified Issue:

- The **hand-tracked XYZ coordinates** were interpreted relative to the **camera's reference frame** (**secondary frame**), while the **3D sound environment** expected them in the **global room's reference frame** (**primary frame**).
- This mismatch led to **unexpected movement**—for example, when the camera rotated 180 degrees, moving the hand **right** resulted in the sound source moving **left** instead.

Solution: I wrote a Python file to externally transform the x, y, z coordinates from the secondary frame to the primary reference frame

1. **Send the hand-tracked x, y, z coordinates (secondary reference frame) to a Python script**.
2. **Retrieve the transformation matrix** using:
   - **Primary reference frame's global position & orientation**.
   - **Camera's (secondary reference frame) dynamically updating position & orientation**.
3. **Apply the transformation matrix** to correctly convert hand-tracked x, y, z into the primary reference frame.
4. **Send the corrected x, y, z coordinates back to Max/MSP** for **accurate spatialization**.

This method is similar to **Virtual Loudspeaker-Based Binaural Audio**, where **sound sources are spatialized in a virtual 3D space before being rendered into stereo audio using external processing**.

Observations & Challenges

- The **2D representation interface** works well—when manually adjusting position sliders, the **x, y movement correctly aligns with head orientation**, rather than the global room frame.
- However, the **3D space implementation remains glitchy** and requires further tuning.
- **Hand tracking introduces noise & irregularities**, making real-time gesture control **unstable**. Due to **time constraints**, achieving **fully smooth hand control** wasn't feasible.
- As a workaround, we successfully used **mouth movements to toggle spatial positioning**, ensuring a **demonstrable result** despite tracking limitations.

Reflections

This project highlighted the **complexity of spatial audio encoding & decoding**—ensuring sound moves correctly relative to a listener's head is **highly intricate** and requires **precise transformations** across **multiple coordinate systems**. Future refinements could improve **gesture stability** and **real-time spatial accuracy**.

```
import socket
import re
from scipy.spatial.transform import Rotation as R
from pythonosc.udp_client import SimpleUDPClient
import re
import numpy as np
# UDP Setup
UDP_IP = "127.0.0.1"
UDP_RECEIVE_PORT = 7500
UDP_OUT_PORT = 7600  # Sending back to MaxMSP
client = SimpleUDPClient(UDP_IP, UDP_OUT_PORT)  # OSC Client


# Define the primary reference frame (default position and rotation)
primary_position = np.array([0, 0, 0])
primary_rotation = np.array([0, 0, 0])  # No rotation

# Define the camera reference frame (relative to primary)
camera_position = np.array([0, 0, 2])
camera_rotation = np.array([0, 0, 0])  # Will be updated dynamically

# Transformation matrix (identity by default)
transformation_matrix = np.eye(4)

# Create a UDP socket
recv_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
recv_sock.bind((UDP_IP, UDP_RECEIVE_PORT))
send_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
print(f"Listening on {UDP_IP}:{UDP_RECEIVE_PORT}...")


def compute_transformation_matrix(rotation, position):
    """
    Compute 4×4 transformation matrix from Euler angles and position.
    - Rotation: roll, pitch, yaw in degrees
    - Position: (x, y, z)
    """
    rot_matrix = R.from_euler('xyz', rotation, degrees=True).as_matrix()  # Convert rotation to matrix
    transform_matrix = np.eye(4)  # Initialize identity matrix (4×4)
    transform_matrix[:3, :3] = rot_matrix  # Set rotation part
    transform_matrix[:3, 3] = position  # Set translation part
    return transform_matrix

while True:
    data, addr = recv_sock.recvfrom(1024)
    message = data.decode('utf-8', errors="ignore").strip()  # Remove decoding errors

    # Debug: Print raw received message
    print(f"Raw Message: {repr(message)}")

    # Split the message
    parts = message.split()
    print(f"Split Parts: {parts}")

    # Ensure valid format (prefix + 3 values)
    if len(parts) >= 4:
        # Clean up the prefix (remove null bytes at the end)
        parts[0] = parts[0].replace("\x00", "")



        # Remove unwanted prefix in the first float
        parts[1] = re.sub(r"^s\x00+", "", parts[1])

        # Remove unwanted null bytes in the last float
        parts[3] = re.sub(r"[\x00,]+$", "", parts[3])

        # Ignore any unexpected additional parts (remove index 4+)
        parts = parts[:4]


        try:
            # Extract and convert float values
            values = list(map(float, parts[1:4]))  # Expecting exactly 3 floats
            prefix = parts[0]

            if prefix == "/camera_rotation":
                # Update camera rotation
                camera_rotation = np.array(values)

                # Compute transformation matrix for camera (secondary frame to primary frame)
                transformation_matrix = compute_transformation_matrix(camera_rotation, camera_position)

                print(f"Updated Camera Rotation: {camera_rotation}")
                print(f"Transformation Matrix:\n{transformation_matrix}")
```

```
elif prefix == "/object_location":
    # Receive object location relative to the secondary frame (camera)
    print("Received `/object_location`!")
    object_position = np.array(values + [1])  # Convert to homogeneous coordinates (x, y, z, 1)
    print(f"Object Position (Secondary Frame, Homogeneous): {object_position}")

    print(f"Applying Transformation Matrix:\n{transformation_matrix}")

    # Transform object position to primary reference frame
    transformed_position = transformation_matrix @ object_position
    transformed_position = transformed_position[:3]  # Extract (x, y, z)

    print(f"Object Location (Secondary): {object_position[:3]}")
    print(f"Transformed Location (Primary): {transformed_position}")

    # Send transformed object position back to MaxMSP
        #send_udp(f"/object_location {transformed_position[0]:.2f} {transformed_position[1]:.2f} {transformed_position

        client.send_message("/object_new_location", [transformed_position[0], transformed_position[1], transformed_p

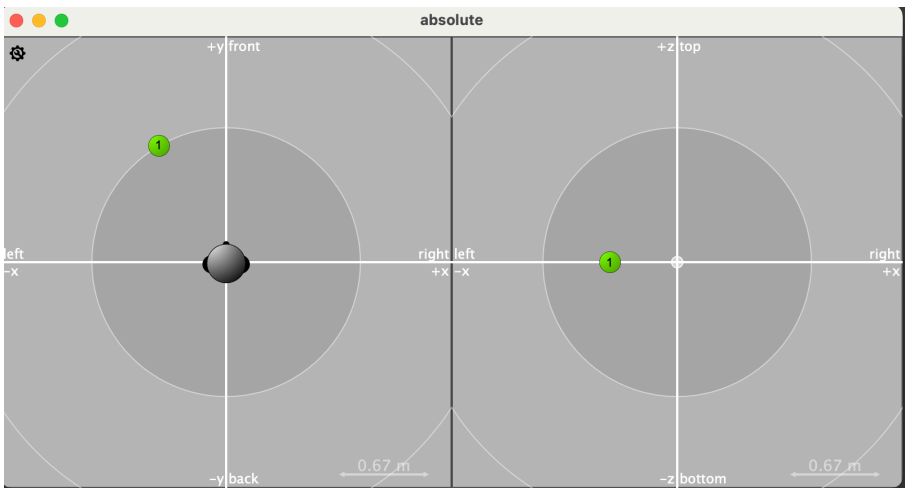

      print(f"Prefix: {prefix}")
      print(f"Extracted Values: {values}")

    except ValueError:
      print(f"Invalid numerical conversion: {message}")

else:
    print(f"Invalid data received: {message}")
```

MaxMSP File & Interface

- Sound Source visualisation interface

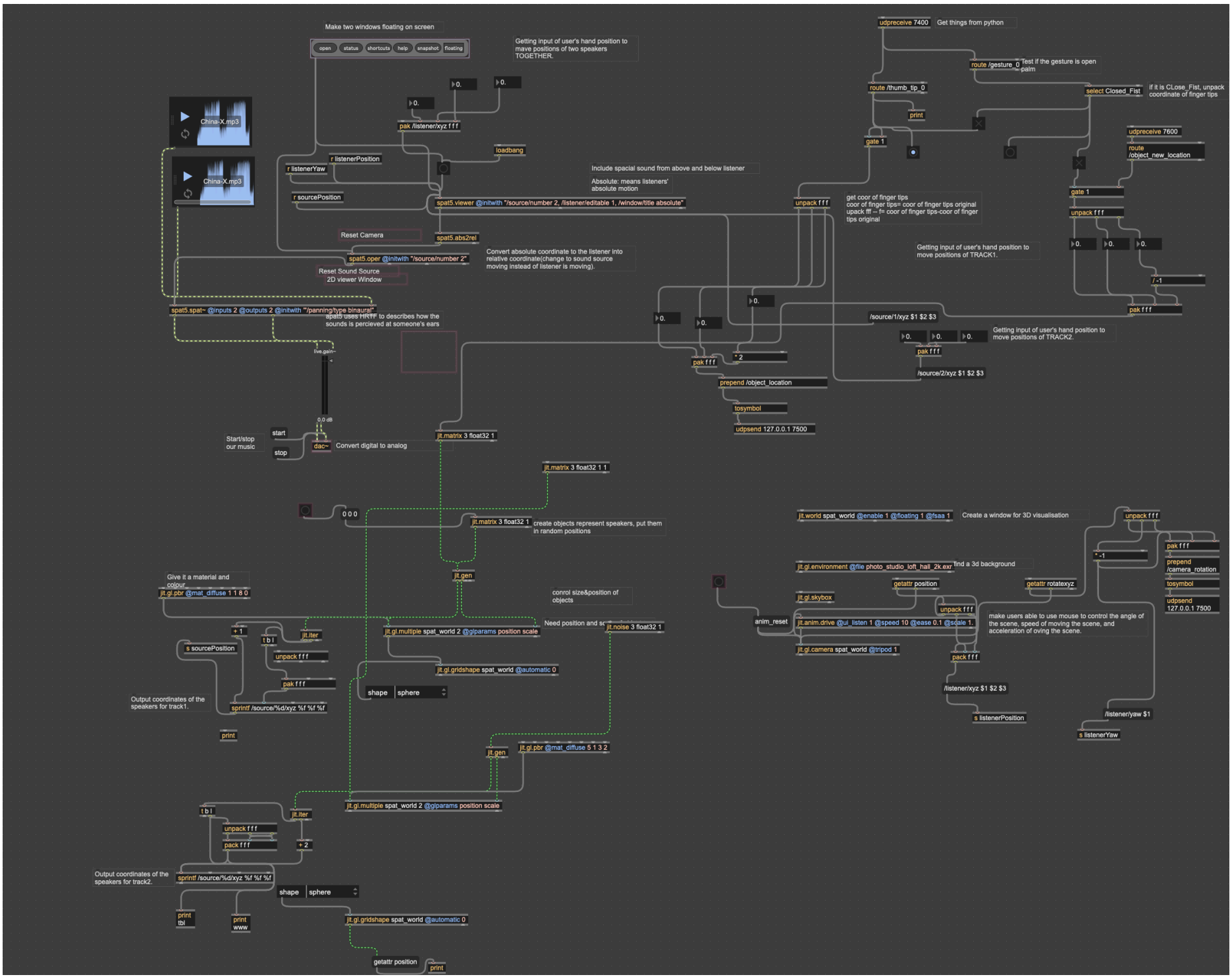


2D Visualisation



2D Visualisation

- MaxMSP File



WP5 - Filtering

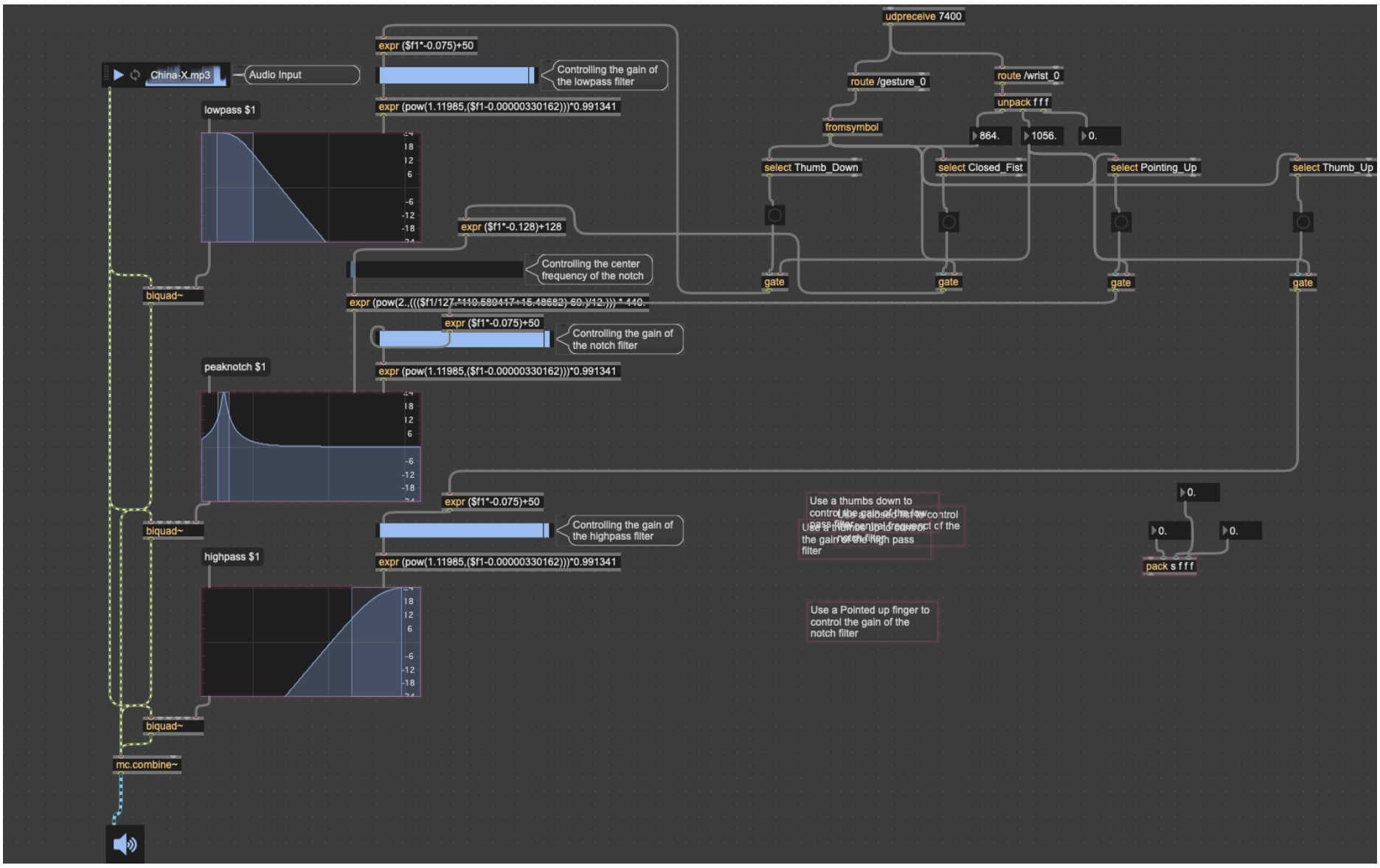
This module explores **frequency manipulation through filtering**, allowing users to intuitively adjust how sound frequencies are shaped. The system replicates real-world **sound mixing desk operations**, enabling **interactive learning** of frequency attenuation and enhancement. A **logarithmic scaling function** was applied to match human perception of frequency response.

Filter Types & Functionality

The system integrates **three filters**:

1. **Low-Pass Filter** → Controlled by **Thumb-Down** gesture.
2. **Peak-Notch Filter** →
    a. **Closed Fist** controls the **centre frequency**.
    b. **Pointing-Up** controls the **gain**.
3. **High-Pass Filter** → Controlled by **Thumb-Up** gesture.

MaxMsp File & Visualisation effect



WP6 - Generative Drum

This module integrates **gesture control** with **drum synthesis** to create a **natural-sounding beat system**. The core functions include:

1. **Drum sound synthesis**: Crafting realistic **kick, snare, and hi-hat** sounds through the use of **delay, reverb, noise, attack envelopes, and function curves**.
2. **Gesture-based beat control**: Using hand gestures to dynamically create or modify beats and groove patterns using **Beat Grid / Drum Matrix** → A similar term used in **DAWs (Digital Audio Workstations)** like Ableton Live or FL Studio.
    a. Random beat generation
    b. Reset
    c. Manual beat generation
3. **Integration with visual feedback**: Linking generated beats to **Vihan's visualisation module** for an intuitive and engaging learning experience.

Part 1: Drum Synthesis

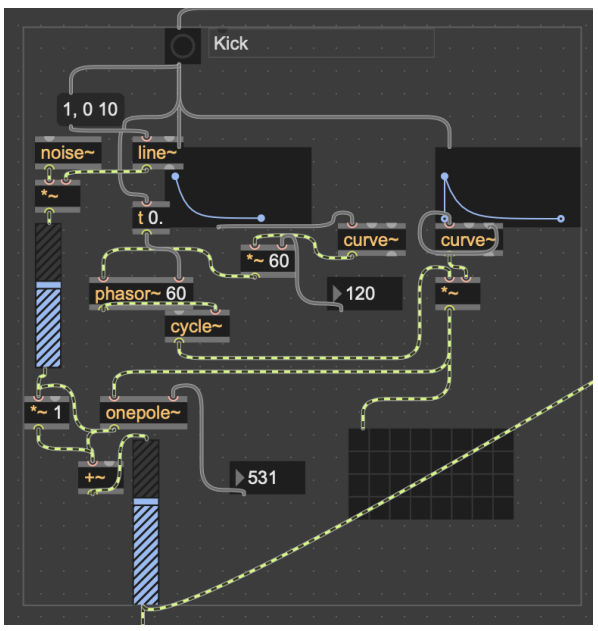
- **Processing Effects Used to create more natural sound (structure shown in screen shot)**
    - **Delay**: Controls rhythmic timing and depth.
    - **Reverb**: Creates spatial perception.
    - **Attack Envelope**: Shapes transient response.
    - **Noise Generation**: Adds natural texture.
    - **Function Curve Mapping**: Ensures realistic behaviour based on human perception.

This max drum synthesis is learnt from the video https://www.youtube.com/watch?v=86EpMhBCNbo&list=PL2gYK1Gjidb3VSqZHChYOtEdIAXw9YXeG, https://youtu.be/5VBmqk9v4Kk?si=9rTD6PaxZJDDFAhe, https://youtu.be/H6Yr7iVX0jM?si=zm2xYEQQpdlgFy3e

- **Kick Drum:** A deep, low-frequency sound synthesized using oscillators and pitch envelopes to
- **Snare Drum**: A sharp, mid-frequency hit combining tonal
- **Hi-Hat:** A bright, high-frequency percussive sound generated using filtered noise and envelope

create a punchy attack and resonant body.

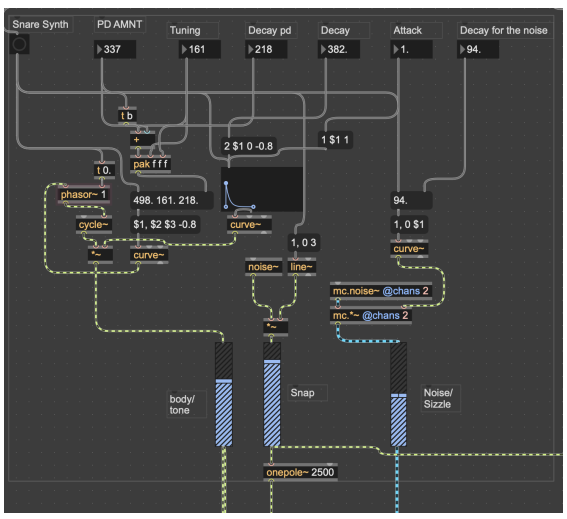
- **Noise Layer** → Creates texture and realism through an **envelope (line~)** that shapes the attack.
- **Pitch Envelope (curve~)** → Controls the **oscillator frequency** to produce a punchy transient that mimics a natural kick drum's decay.
- **Phasor & Cycle Oscillators** → Create the fundamental **low-frequency body** of the kick.
- **Filtering (onepole~)** → Smooths the waveform by removing harsh high frequencies.



oscillation and noise to replicate the body and snare wire sizzle.

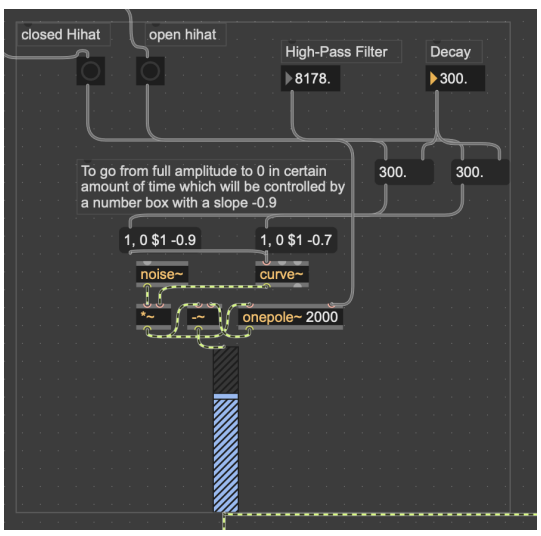
A **snare drum** is a **percussive instrument** found in **drum kits** and **marching bands**. It creates a **sharp, crisp sound** through **metal wires (snares)** stretched across its bottom drumhead.

- **Oscillators (phasor~, cycle~)** → Generate the **body/tone** of the snare.
- **Noise (noise~, mc.noise~)** → Creates the **sizzle effect** of the snare wires.
- **Envelope (curve~, line~)** → Controls the **decay and transient attack** for realism.
- **Filtering (onepole~)** → Adjusts the **damping and resonance** of the snare sound.



shaping to differentiate closed and open variations.

1. **Noise Generator (noise~)** → Creates the **metallic, noisy texture** of a hi-hat.
2. **Envelope (curve~)** → Controls the **amplitude decay** to differentiate **open** (longer decay) and **closed** (shorter decay) hi-hats.
3. **High-Pass Filtering (using -~ with onepole to create high-pass filter & HPF at ~8kHz)** → Removes low frequencies to **emphasise the crisp high-end** typical of hi-hats.
4. **Decay Control** → Adjusts how long the hi-hat **rings out**, making it **tight (closed)** or **sustained (open)**.



Part 2: Single hand motion-Tracking

1. **Thumb_Down Gesture** → Resets the beat (disables all active beats).
2. **I_Love_You Gesture** → Randomly generates a new beat (with a short delay to avoid re-triggering).
3. **Thumb_Down Gesture** → Start the beat visualisation

Part 3: Two Hands Motion-Tracking

([*Diagrams Needed: Finger Coordinate-to-Matrix Conversion Logic*])

- **Tharindu's Python Script** retrieves:
  - **Hand gestures** via **MediaPipe**.
  - **Finger joint coordinates** for mapping.
- **I did Pre-processing in Python**:
  - Converts **raw fingertip coordinates** into **matrix activation data**.
  - Sends **binary activation values (1 = active, 0 = inactive)** to Max/MSP.
  - Improves **efficiency** by handling calculations externally before sending data.

```python
# Grid configuration
GRID_COLS = 8  # Number of columns
GRID_ROWS = 3  # Number of rows
X_MIN, X_MAX = 100, 800  # X range for filtering
Y_MIN, Y_MAX = 200, 600   # Y range for normalization
def get_cell(x, y):
    """Convert X, Y to grid row/column."""
    if x < X_MIN or x > X_MAX:
        return None  # Ignore invalid X

    # Normalize X and map to column index (0-7)
    col = 7 - int(((x - X_MIN) / (X_MAX - X_MIN)) * (GRID_COLS))

    # Normalize Y and map to row index (0-2)
    row = int(((y - Y_MIN) / (Y_MAX - Y_MIN)) * (GRID_ROWS))

    return row, col
```

```python
            for landmark in hand:
                client.send_message("/" + lmkNames[id] + "_" +
                str(handID), [int(landmark.x * 1000), int
                (landmark.y * 1000), int(landmark.z * 1000)])
                # draw landmarks in video output
                if id in lmksToDraw:…
                if id in fingerTips:
                    '''Added parts'''
                    x, y = int(landmark.x * 1000), int(landmark.
                    y * 1000)
                    cell = get_cell(x, y)
                    if cell and cell not in active_cells:
                        active_cells.add(cell)  # Store active
                        cell

                id = id + 1
            handID = handID + 1

        # to update matrix for the drum beats
        grid_matrix = [1 if (row, col) in active_cells else 0
        for row in range(GRID_ROWS) for col in range(GRID_COLS)]
        print("Grid Matrix:", grid_matrix)  # Debug the entire
        matrix
        client.send_message("/grid_matrix", grid_matrix) #
        Send all grid data at once
```

Part 4. Gesture-Based Beat & Groove Control

The track uses a 500ms metronome setting, creating steady 4/4 time signature beats

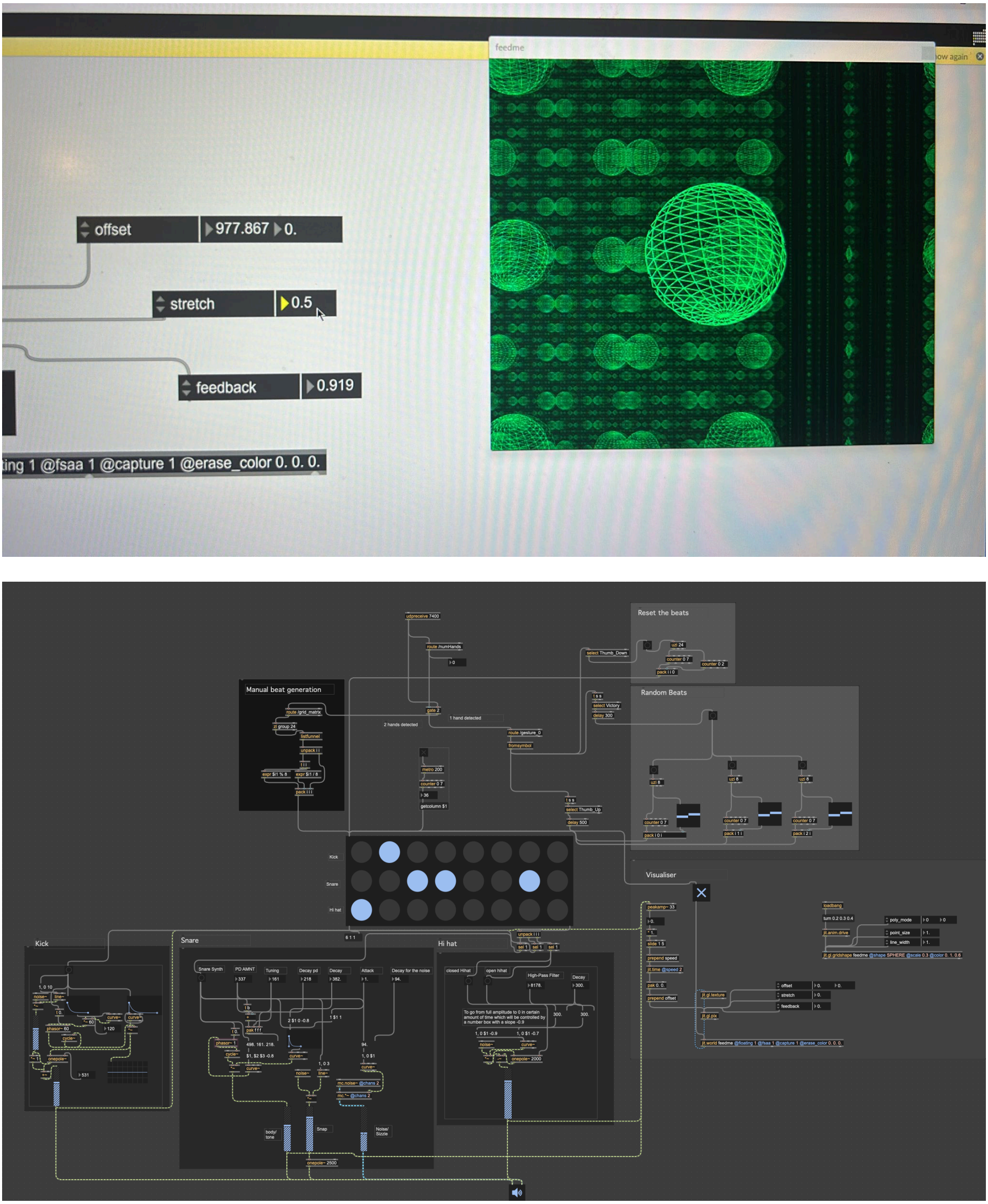
- **Hand Gesture Mapping Strategy**
  - **One Hand Detected:**
    - **Thumb_Down Gesture** → Resets the beat (disables all active beats).
    - **I_Love_You Gesture** → Randomly generates a new beat (with a short delay to avoid re-triggering).
    - **Thumb_Down Gesture** → Start the beat visualisation
  - **Two Hands Detected:**
    - **Fingertip Tracking**:
      - X & Y coordinates map to a **3-row × 8-column** matrix (representing drum sequencing).
      - **Rows** correspond to the **three drum sounds** (Kick, Snare, Hi-Hat).
      - **Columns** r**epresents a single time step** in the beat pattern. sequencing control(sequencing the cell in the matrix to trigger each drum sound at the correct ime based on the programmed pattern.
      - The corresponding `matrixctrl` object in Max/MSP lights up when a beat is activated.
- **Additional Control Methods**
  - Users can also still **manually adjust beats in Max/MSP's UI by clicking the matrix cell**.
  - The output sound is **linked to Vihan's visualisation**, synchronising beats with **real-time visual cues**.

Conclusion & Future Refinements

- The system **successfully integrates gesture-based drum control** with **real-time audio synthesis**.
- **Challenges & Improvements:**
  - Gesture noise and tracking inconsistencies affect **precision**.
  - Potential refinements include **smoother motion tracking** and **gesture filtering**.
  - Future updates could incorporate **dynamic tempo adjustments and genre-based beat presets**.

MaxMsp File

The sphere and background will change based on the beats and the drum sound:



WP7 - Merge & UI

1. **Merging Workflow & Signal Routing**

   (The full logic diagram is shown at the introduction of the report)

   All 4 modules are running parallel with each other.

   All modules (except the generative drum) are connected to a the same sound source, by change the gate we can set which one to be activated and apply the filters in the module on the sound source.

2. **Using Gates for Selective Activation**

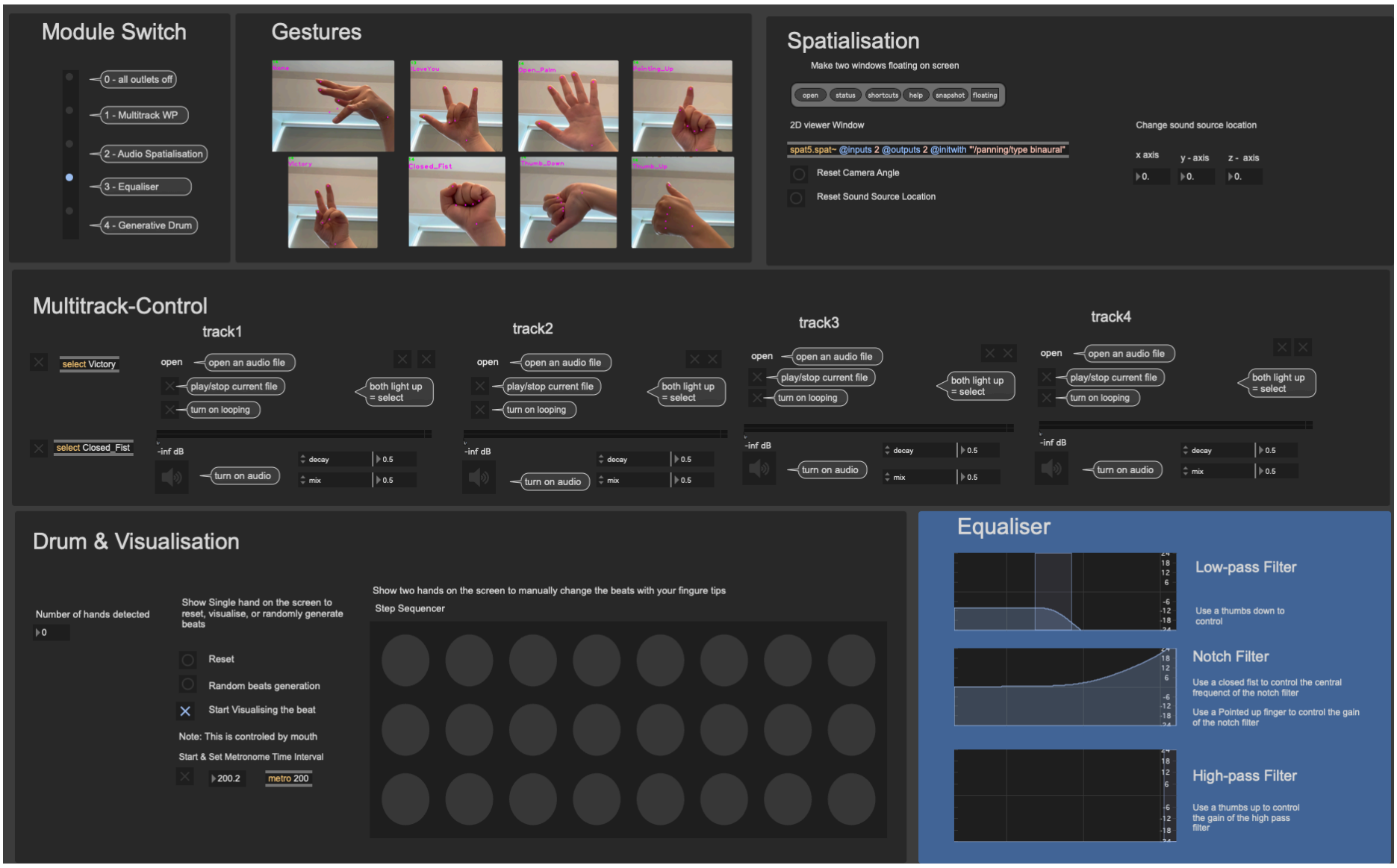
To **optimise performance** and **streamline user interaction**, only the **selected Work Package (WP)** is activated at a given time. This prevents unnecessary processing and ensures a **clear, structured workflow**.

To provide **visual feedback**, each WP has a **corresponding panel** that dynamically changes colour based on its activation state:

- **Inactive** → Default state (Black)
- **Active** → Indicates the selected WP (Blue)

3. **Real-Time Video Handling**
   - The **three live video feeds** (**motion tracking, 3D spatialisation, and drum-triggered effects**) are displayed dynamically.
   - For **time efficiency**, we extended the Mac **with an iPad** to use **two screens**:
     - **Main Screen** → Max/MSP Presenter Mode
     - **Secondary Screen** → Live visual outputs

This is the UI dashboard I made, will definitely make it more user friendly and aesthetically beautiful in the future



Conclusions, Potential & Future Work

Challenges & Areas for Improvement

- **Early Planning & Coordination** → Starting tasks earlier and improving **team communication** could enhance workflow efficiency.
- **Gesture Responsiveness** → Achieving **instantaneous beat updates** while preventing accidental changes remains a challenge. Balancing **real-time responsiveness** with **stability** is crucial to avoid unintended modifications when users move their hands out of the detection range.
- **Fine-Tuning Gesture Mapping** → Ensuring **precise control** over beat patterns without misinterpretation, particularly near the edges of the **matrixctrl**, requires further refinement.
- **Synchronising Sound Source Movement with Camera Angle** → Moving a **sound source dynamically in 3D space** while ensuring it remains synchronised with **camera movements** is highly complex. Fine-tuning the **motion tracking system** to provide **smooth and natural** sound positioning requires further optimisation.

- **3D Spatial Audio Tracking** → Achieving **fluid, precise movement** of sound sources within a **spatialised 3D environment** is challenging, particularly in translating **hand gestures into consistent, smooth audio positioning** without lag or jitter.

Key Achievements & Learnings

- **In-Depth Technical Exploration** → Improved understanding of **audio synthesis, gesture mapping, and external file interaction** in Max/MSP.
- **Advancements in 3D Spatialisation** → Gained insights into the complexity of **spatial audio integration**, highlighting the level of effort required in industries like **game development** to achieve immersive 3D sound experiences.
- **Stronger Analytical Approach** → Developed a **more structured method** for assessing technical challenges, ensuring efficient implementation strategies.

Future Enhancements & Potential Expansions

1. **Multi-Hand Tracking for Group Interaction** → Extending the system to support **multiple users simultaneously** for a more collaborative and dynamic experience.
2. **Seamless Gesture-Based Navigation** → Enabling users to **switch between modules** using gestures alone, removing the need for manual computer input.
3. **Classroom Integration & Lesson Plans** → Developing structured **educational materials** to make **Maestro Motion a viable learning tool** in school environments.
4. **Enhanced Drum Sound Design** → Expanding **interactive drum channels** with customisable **delay, reverb, and echo parameters**, allowing users to create and fine-tune unique sounds.
5. **Web-Based Implementation & API Development** → Transforming **Maestro Motion into a fully online platform**, providing an **open API** for third-party developers to build **custom learning modules** for niche audio concepts.