Homework Assignment Report

Computer Music: Languages and Systems

Academic Year 2023-2024

SIPquencer by Rolandos

Corà Riccardo
Galante Niccolò
Moschen Riccardo
Polimeno Stefano

Table of contents

Aim and Overall Structure

The aim of this project is to convert an old telephone into a functional computer music system while retaining its original keypad, handset and circuits. The core objective is to utilize the original vintage hardware to interface with modern digital audio tools, thus merging analog interaction with digital music synthesis and generative art. The system will function as a sequencer capable of handling three different sequences and also offer the functionality of a traditional MIDI keyboard. Both synth and drum sounds will be available for experimenting with music creation. Through the GUI, users will be able to tweak sounds and effect parameters. The handset will be used to modulate these parameters, making live performances more exciting and interactive.

Signal Processing Workflow

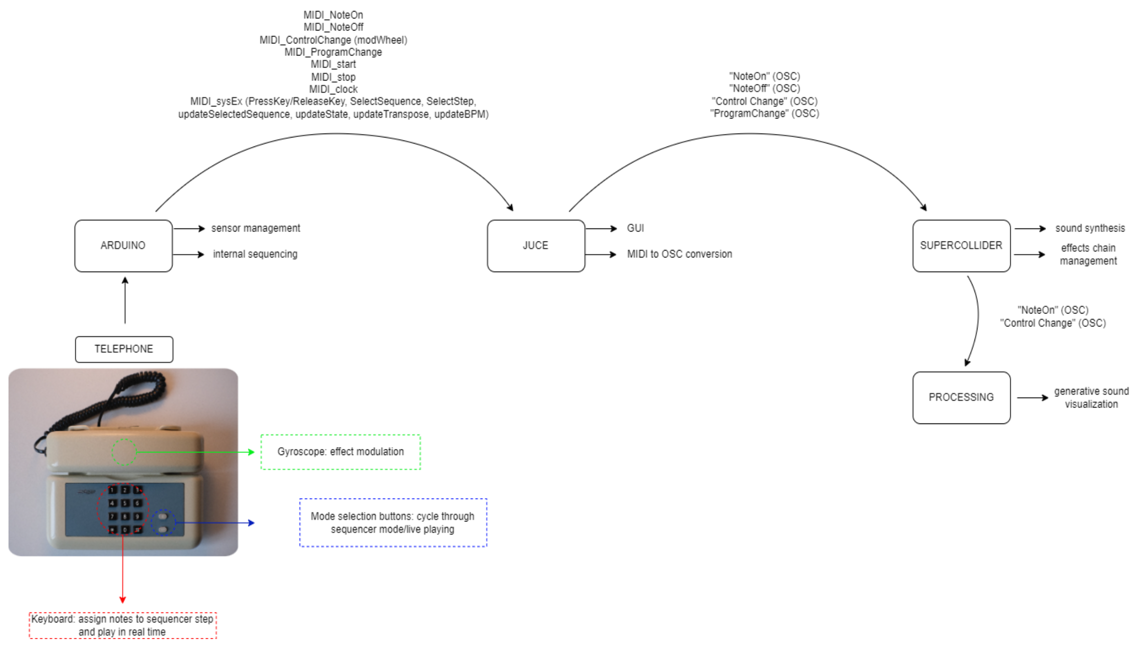
1. **Arduino to MIDI Conversion**:
   - The Arduino board receives a digital input from the keypad.
   - Inputs are translated into MIDI signals: each key press corresponds to a specific MIDI note, control change or program change.
   - MIDI messages are sent through 4 different channels, with the first channel being reserved to the live performance mode .

2. **MIDI to OSC Conversion with JUCE**:
   - The MIDI signals generated by Arduino are sent to a VST developed with JUCE.
   - JUCE converts the MIDI signals received as input into OSC (Open Sound Control) messages, which are used for communication between various multimedia devices.
   - A GUI in JUCE supports the user and helps them navigate through all of *SIPquencer*'s functionalities. JUCE opens an OSC communication channel and starts communicating with SuperCollider, converting MIDI messages into coherent OSC messages. Through the GUI, you can also change sound and effects parameters.
   - We opted for OSC communication over direct MIDI messaging due to the need for additional steps in some operating systems when sending MIDI, such as creating a MIDI bus for communication to occur effectively.Setting up these MIDI buses often requires external application, adding complexity to the process.
     This choice allows for smoother communication, both sending and receiving messages, within the user computer.

3. **Synthesis in Supercollider:**
  - SuperCollider receives OSC messages from JUCE and performs sound synthesis and effects.
OSC messages are used in order to trigger sounds (NoteOn, NoteOff) both of sequences and keyboards and to modulate sound parameters and effects (Attack, Decay, Release, Reverb, Decay).

4. **Generative Art with Processing:**
  - Processing, a flexible software sketchbook, receives the OSC messages, based on the note number and control parameters embedded in the OSC messages it generates visual art.
  - The generative art dynamically changes in response to the musical input, providing a visual representation of the music being played.

By integrating these components, the project effectively combines vintage telephone hardware with contemporary digital music and art technologies, creating an innovative and interactive computer music system.


MIDI_NoteOn
MIDI_NoteOff
MIDI_ControlChange (modWheel)
MIDI_ProgramChange
MIDI_start
MIDI_stop
MIDI_clock
MIDI_sysEx (PressKey/ReleaseKey, SelectSequence, SelectStep, updateSelectedSequence, updateState, updateTranspose, updateBPM)
"NoteOn" (OSC)
"NoteOff" (OSC)
"Control Change" (OSC)
"ProgramChange" (OSC)
ARDUINO
sensor management
internal sequencing
JUCE
GUI
MIDI to OSC conversion
SUPERCOLLIDER
sound synthesis
effects chain management
"NoteOn" (OSC)
"Control Change" (OSC)
PROCESSING
generative sound visualization
TELEPHONE
Gyroscope: effect modulation
Mode selection buttons: cycle through sequencer mode/live playing
Keyboard: assign notes to sequencer step and play in real time

Hardware configuration: Arduino

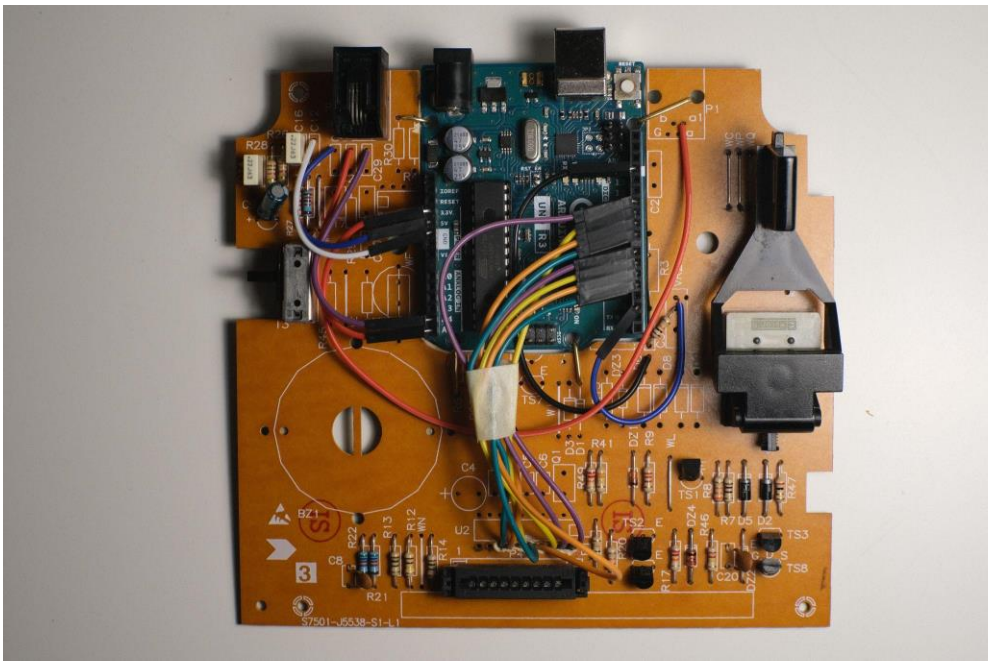
To accommodate Arduino, a portion of the circuitry within the telephone was desoldered and replaced with the board.

In order to preserve the structure of the telephone's original keypad, it was necessary to make use of its native button pressing system, which is based on a 4x4 keypad matrix. Each button on the keypad corresponds to a unique pair of row and column coordinates. When a button is pressed, it connects a specific row to a specific column.

To read the button presses, one digital input on the Arduino was dedicated for each row and one for each column. So, the entire keypad is controlled by 8 pins on the Arduino (4 for the rows and 4 for the columns).

When a button is pressed, the Arduino detects which row and column have been connected. By identifying the intersecting row and column, the Arduino determines exactly which button was pressed. For example, if the second row and the third column are connected (digital HIGH on their corresponding pins), Arduino knows that the button at the intersection of the second row and third column has been pressed.

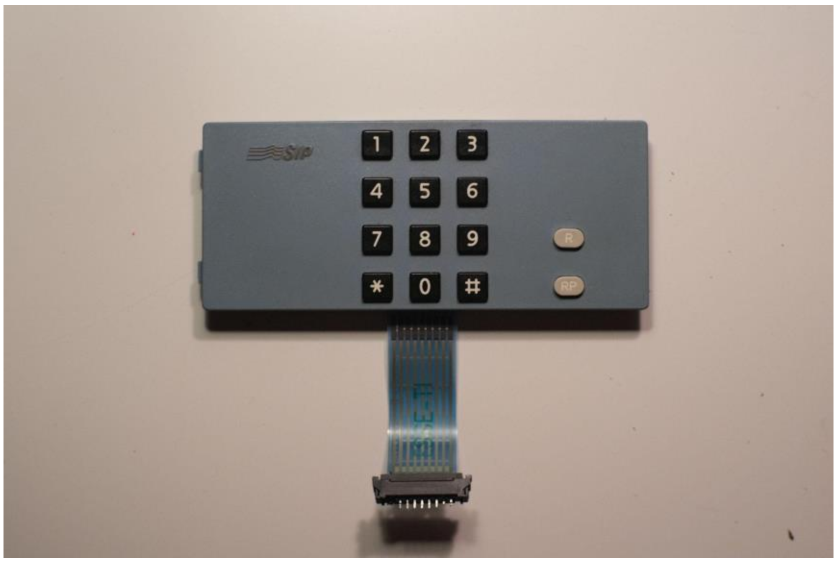
This setup allows for efficient input reads from the keypad and maps each button press to a specific function or MIDI command.



Serial to MIDI Conversion

The Arduino to MIDI conversion process involves several key components and steps.

The keypad setup is configured. A 4x4 keypad matrix is defined with specific pins and keys. The Arduino “Keypad” library handles the scanning and state management of these keys.



The *SIPquencer* system operates as a finite state machine (FSM), managing different states associated with key interactions, such as “default”, “sequence selection”, “step selection” and “note selection” states. This ensures proper handling of key presses and releases to trigger the appropriate MIDI actions.

The sequencer is then configured with four tracks, steps, note lengths, velocities, and other parameters. Each step in the sequencer can hold data such as the note, accent, glide and rest.

MIDI configuration follows, where the MIDI channel and standard MIDI message definitions (e.g., Note On, Note Off, MIDI Clock, Start, Stop) are set up according to MIDI standards.

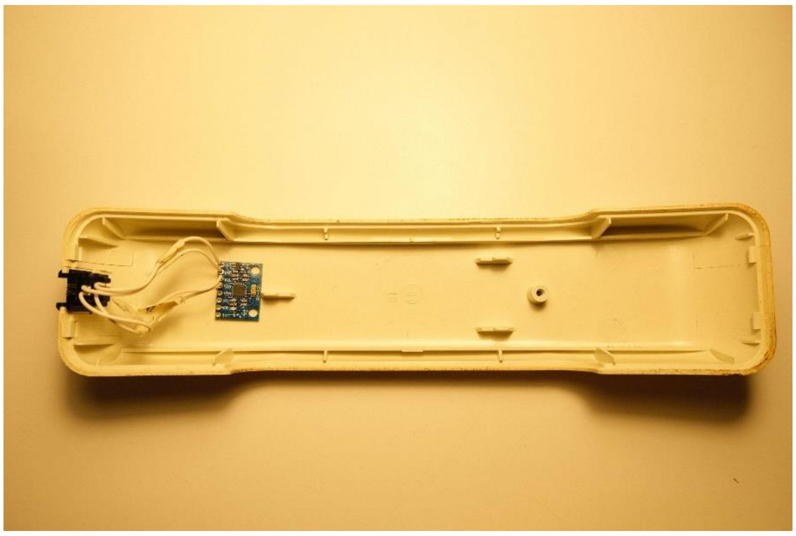
Finally, MIDI messages are sent directly from Arduino. This design allows the device to be used directly in any digital audio workstation (DAW) without needing a VST for MIDI to serial conversion. This makes the device a standalone solution for MIDI communication.

Gyroscope Implementation

Implementing the gyroscope consists of integrating the MPU6050 sensor to control sound parameters based on the orientation of the telephone's handset.
The "MPU6050_light" Arduino library is used to interface with the sensor, allowing for sensor initialization which sets up I2C communication and calculates coordinate offsets based on the initial position of the handpiece allowing for more accurate readings.The data being received from the sensor is then processed to allow for optimal user experience (the functional range of the sensor's rotation is limited to 120°) and mapped to a range that is compatible with MIDI protocol messages.

Whenever the hookswitch button is pressed and the sensor data changes, the processed value is then packaged as a MIDI CC (Control Change) message and sent through the serial port using the "MIDI" Arduino library. This allows for real-time modulation of MIDI parameters during sequence playback and adds a dynamic element to the performance, controlled by the movement and orientation of the device.

By combining the Arduino to MIDI conversion with the gyroscope implementation, this project enables innovative and expressive control over MIDI devices, enhancing the versatility and interactivity of electronic music performance.

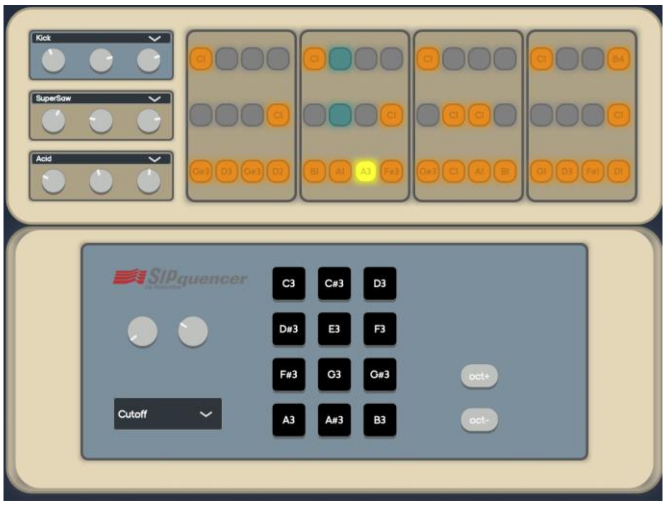


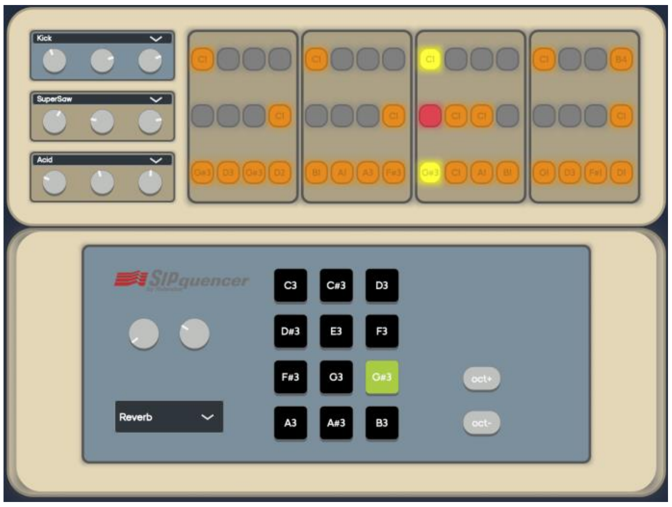
Juce

OSC handling and user interface

In the JUCE framework, the graphical user interface (GUI) can be used to modulate sound parameters and to select the controls associated to the handset movements.
JUCE plugin's processor also handles the conversion of MIDI data to OSC messages, enabling SuperCollider to receive OSC messages. This process ensures real-time control over SuperCollider's synths and parameters.

The GUI has been crafted using a "components approach" in order to optimize performance by refreshing only the relevant parts that undergo modifications, minimizing the computational power and time required for rendering graphical elements, leading to an efficient and responsive user experience. This solution also allows for modular development of future implementations.





Supercollider

Sound synthesis and effects

Various synthesizers, each with distinct timbres and characteristics, have been crafted for this system. Despite their differences, they share common modulation parameters, including attack, release and cutoff. This design allows users to seamlessly modulate these parameters across all synthesizers, providing a cohesive and intuitive experience.

Two effects, reverb and delay, are defined to enhance the synthesizer sounds. Parameters within these effects are chosen to enrich sounds with various presets, while the mix parameter (equivalent to DRY/WET) is made available for modulation.

To manage these effects, audio buses are set up to efficiently route audio signals, and the effects operate in parallel. Meanwhile, the “PolyphonicSynthManager” class, tailored specifically for this system, is instantiated for each synth type, facilitating polyphonic synthesis and dynamic note management.

OSC messages: receiving and sending

Supercollider both receives OSC messages from JUCE and forwards them to Processing. Note On, Note Off, Control and Program changes listeners are instantiated in the SIPquencer.scd file. Supercollider handles 4 channels: 3 sequence channels and 1 “live” channel.

For optimized control, OSC definitions handle incoming OSC messages, allowing for real-time adjustments of synth parameters and program changes. These messages are forwarded to another port to communicate with the Generative Art portion of the system.

PolyphonicSynthManager Class

The PolyphonicSynthManager class is essential for managing polyphonic sound synthesis within SuperCollider, allowing the system to handle multiple simultaneous notes from different MIDI channels and real-time parameter adjustments smoothly. Each MIDI channel output is characterized by an output bus and a program (Synth), which are also provided as arguments to the class constructor.
Within this class, polyphony is managed through a noteOn method and a noteOff method. When a note is played, a new Synth is instantiated and assigned to the corresponding index in the notes array. Conversely, when a note is released, the noteOff method sets the gate parameter of the corresponding Synth to 0, effectively turning off the note, and then removes the synth from the notes array by setting the entry to nil.
This approach ensures that the system can handle polyphony efficiently, with each note allocated a unique Synth in the notes array, allowing for simultaneous playback and precise control over each note's parameters across multiple channels. The design allows for flexible management of sequences, with each of them having their own dedicated output bus and synth program, facilitating a rich and versatile music creation experience.
Additionally, with this approach, we could hypothetically receive other added sequences and handle them seamlessly.

Overall benefits:

- **Flexibility**: Provides a flexible framework for handling complex polyphonic synthesis, suitable for live performances, interactive installations, and studio productions.

- **Real-Time Control**: Enables real-time manipulation of sound parameters, allowing performers and composers to dynamically shape the audio output.

- **Scalability**: Easily scalable to manage a wide range of synth sounds and effects, supporting intricate and evolving soundscapes.

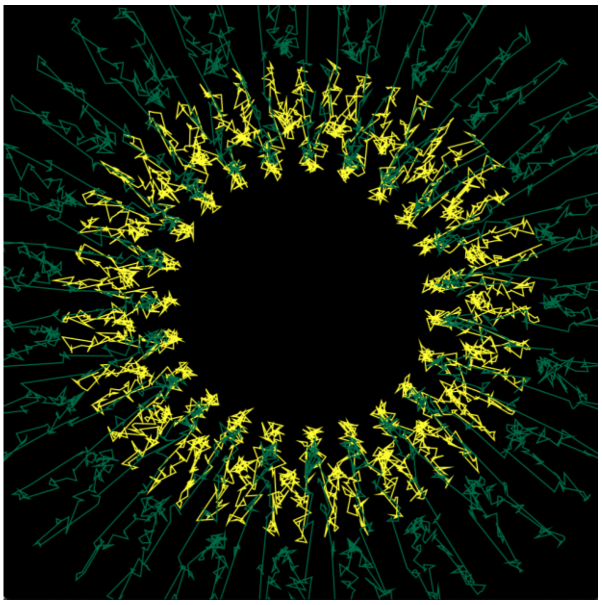
In summary, the “PolyphonicSynthManager” is a powerful tool within SuperCollider that orchestrates polyphonic synths and real-time control, enhancing the system's ability to create and manage complex, layered audio compositions.

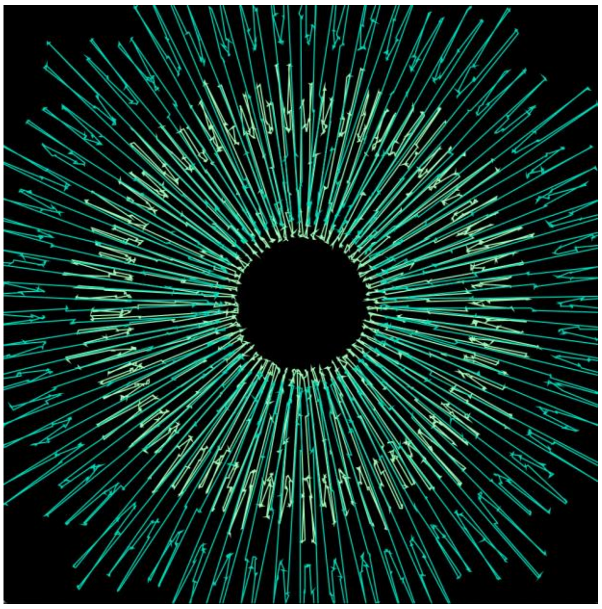
Processing

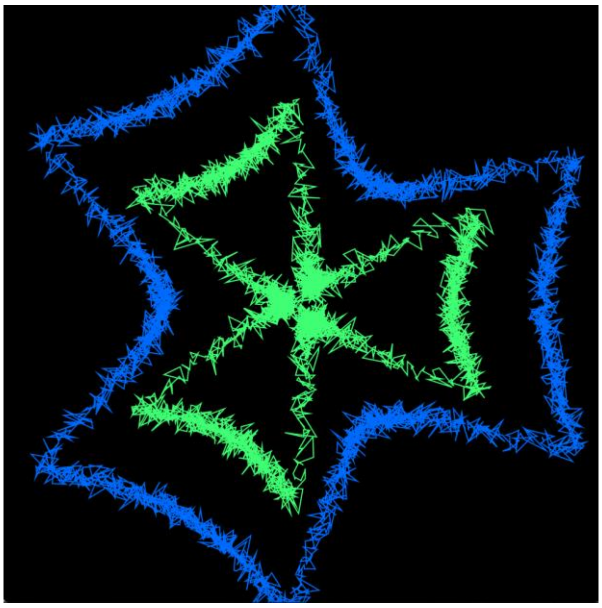
Generative Art

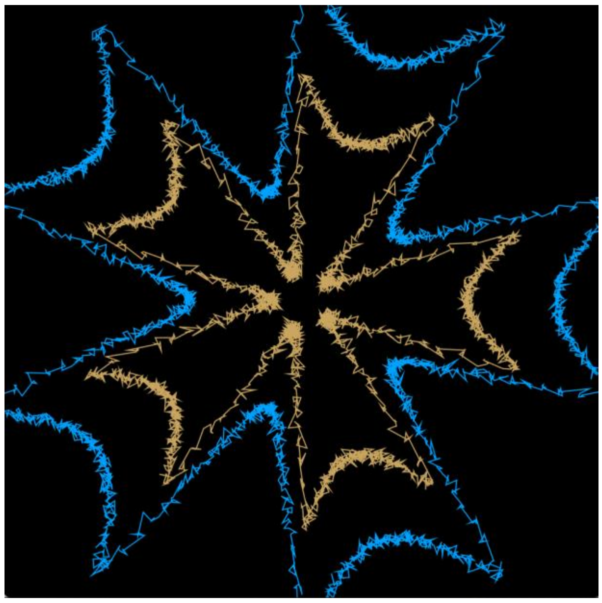
Processing utilizes OSC (Open Sound Control) messages to dynamically generate visual patterns.

Upon receiving an OSC message, the oscEvent function is triggered. This function extracts the address pattern and type tag of the received message and processes it accordingly. Depending on whether the address pattern is "/NoteOn" or "/Control", specific actions are taken.









In the drawing phase, implemented within the draw function, shapes are drawn for each channel. These shapes are based on the current channel note and parameters such as attack, decay, cutoff, reverb, and delay.
Attack, decay, cutoff modulates RGB color values of the figure, whilst reverb and delay modulate two extra visual effects.

Overall, the code dynamically generates visual patterns based on the received OSC messages, with the shapes and their properties influenced by the note numbers and control parameters received.