Design Lab, 2021

Audio analysis and MIDI generation

Krzysztof Kumka
Filip Imiela

Cracow, January 2021

1. Main goal

The main object of this project was to create a fully functioning device that manages to analyse input audio in .wav format and generate corresponding MIDI signals.Created file can be further processed with DAW like Ableton, FL studio and more.

2. Problem analysis

We can divide the whole project into two separate parts.

First: Audio analysis:
It should be based on the well-known Discrete Fourier Transform and particularly in this case Short-time Fourier Transform for signals changing over time. At this point we decided to choose the McAulay-Quatieri method for its simplicity. It is useful for pulling out hearable frequencies in audio tracks based on musical instruments having harmonic nature.
MQ Method:
• Calculate STFT
• Find local maxima
• Reject values smaller than the threshold
• Tracking – creating continuous frequency tracks with use of maxima
• Reject short tracks

Calculation of STFT:

$X_{STFT}\left[m,n\right]=\sum_{k=0}^{L-1}x\left[k\right]g\left[k-m\right]e^{-j2\pi nk/L}$, where
g[k] is a so-called window function which defines time resolution. There is a problem at this point. To find notes it is required to have specific frequency resolution which can be calculated with basic formula df = Fs/N, where, where Fs is sample frequency and N - number of windowed samples (or length of window) in one DFT calculation. If we extend time resolution, we reduce frequency resolution. For our reference we chose E2 for lowest note, so the smallest absolute value between frequencies of F2 and E2 is about 5 [Hz] that is why it is necessary to ensure frequency resolution at this level.

Local minima:

Finding local minima from STFT should pull out hearable basic frequencies including their harmonics.

Rejecting smaller values:

It ensures that noise isn’t taken under account.

Cracow, January 2021

Rejecting short tracks:
Short tracks can be treated like error frequencies so we delete them.

Second: Arduino MIDI controller:
After creating the MIDI array, it was supposed to be sent via serial port into Arduino memory. After collecting data, Arduino should play received notes in digital audio workstation

3. SWOT analysis

Before starting the project we conducted SWOT analysis to see what are our weaknesses and what problems can we face during implementation. SWOT helps to take advantage of our strengths. Result of this presented below:

Threats:

- Failure to deliver additional Arduino components on time,
- problems caused by remote communication between team members,
- problem with delivered sounds that are more real, which includes noise or ambient sounds, not only "clean" frequency,
- problem with analysing pitch bended or vibrating sounds.

Weaknesses:

- Lack of experience in creating embedded software,
- Lack of experience in hardware usage,

Strengths:

- Low cost hardware components,
- Interest of the subject by team members, both digital signal processing and music,
- Wide set of components and devices to record audio files with different level of complexity (microphone, synthesizer etc.),
- Wide range of dedicated libraries for digital signal processing and MIDI programming.

Opportunities:

- Lack of similar systems on the market,
- Growing demand on digital audio accessories,
- Working system may be used by team members for musical purposes.

Cracow, January 2021

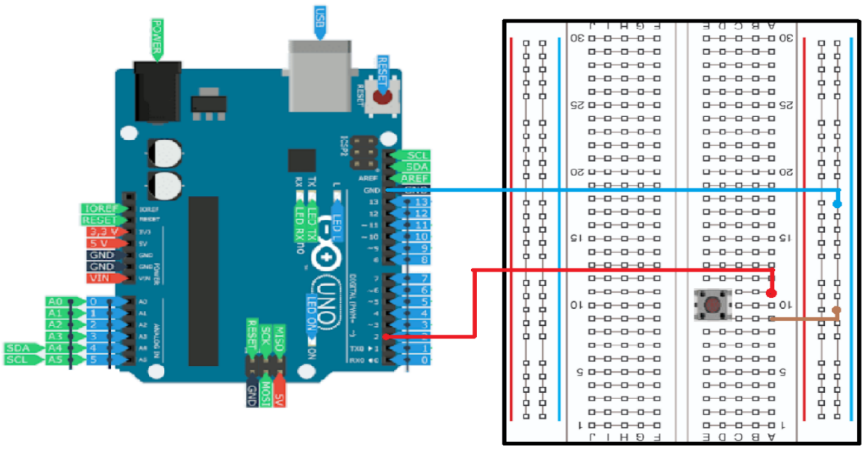
4. Project implementation

HARDWARE ARCHITECTURE

In order to create MIDI controller our team decided to use Arduino UNO in specified configuration:

- USB cable, for communication with a computer.
- Protoboard.
- Male jumper wires.
- Button.

Arduino scheme:



The idea of this connection is to send MIDI sequences into digital audio workstations.

SOFTWARE ARCHITECTURE

Environments:

- Pycharm - Analysis script, main purpose is to process audio files and generate proper 2D arrays of frequencies and gain.

Used libraries: numpy, matplotlib, scipy.io.wavfile, scipy.signal.

- Arduino IDE - Analysing delivered MIDI notes and sending them to digital audio workstations.

Cracow, January 2021

Used libraries: MIDIUSB, MIDI Library.

- Hairless MIDI & LoopBe Internal MIDI - Creating MIDI connection between Arduino and computer through USB cable.
- Ableton live - Final stage of connection, digital audio workstation for checking proper device performance.

I/O DOCUMENTATION

Python I/O:

- Input: *.wav* type file.
- Output: 2D array with MIDI data.

Arduino I/O:

- Input: 2D array with MIDI data from Python script.
- Output: Sequence of MIDI signals transferred to digital audio workstation.

Python script realizes analysis of audiofile. As mentioned, it takes a .wav file and creates a basic spectrogram based on STFT implementation. For increasing time resolution we implemented overlapping with 50% window length. Final time/frequency resolutions are written in the tests report below. After calculating STFT we take local maxima according to the MQ method. Script prints this chart too. For clarity range of printed frequencies is 0-1 [kHz] as these frequencies are the most common in the music world. Another function takes a newly created gain array and cuts rows which do not contain any hearable frequencies. At this point calculated frequencies are matched to the real ones with least absolute value function and converted to proper MIDI values with basic formula $m_n = 69 + log_2(\frac{f}{440})$.

With collected MIDI data from analysis, Python script is able to send data through serial port into Arduino memory.

As for the MIDI controller, Arduino is capable of sending previously declared MIDI signals in real time into a digital audio workstation with Hairless MIDI and LoopBe.

Cracow, January 2021

5. Planned tests and results

At the final step of the project we planned tests we should conduct to confirm if the system works properly.

Functional testing	
Positive testing	Negative testing
Testing reliability of hardware and software connection. Checking if Arduino generates correct MIDI notes based on Python analysis.	Testing range of frequencies, audio volume and file length that are acceptable by the program. Determining optimal parameters ranges for a program.
Testing alternate flow: • Uploading file with wrong format. • Uploading “silent” audio file. • Using blurred audio files.	Changing quality of input data: • Clean frequencies generated with Matlab and Python. • Basic sounds generated with digital instruments (without major effects). • Clean frequencies recorded with microphone (containing noise). • Recorded voice (containing noise). • Pitched frequencies.
	Ad-hoc tests, providing additional tests after checking if everything is working fine.

Non-functional testing	
Performance testing:	Iterating through the size of the audio file and checking the time that is required to make complete analysis.
Installation testing:	Creating a short guide on setting up the program and (if there will be enough time) testing clarity of GUI.

Cracow, January 2021

Report from conducted tests:

Functional testing:

Tests with positive result:

- Positive testing:
  - Arduino plays the previously declared sequence correctly. Declared sequence is the same format as the sequence generated with the Python script.
  - Python script created in order to send processed MIDI notes to Arduino works properly on a declared data.
  - Uploading files with the wrong format results in “Enter valid file!” comment in Python console
  - Uploading silent file generates no files/arrays with midi notes and gain
- Negative Testing:
  - Python script analyzes frequencies between 0 - 22 [kHz] but for spectrogram it shows freq scale between 0 - 1k [kHz] for the clarity
  - Audio volume is neutral for program performance
  - Audio files can be relatively long, e.g 6 min
  - The script works properly for multiple clean computer-generated frequencies and clean digital instruments

Tests with negative result:

- Positive testing:
  - Arduino script fails at processing received data with MIDI notes, therefore, it cannot be played instantly after executing Python code.
  - Blurred audio files aren’t analyzed properly (Python script pull out whole range frequencies with highest gain
- Negative testing:
  - Script doesn’t work properly on pitch-bended frequencies and recorded audio files containing larger amounts of noise.

Non-functional testing:

- Long audio files don't slow down Python script in any noticeable way. Time required for creating a complete analysis is short. For 6 min piano audiofile it takes about 1.5 min to analyze.
- Frequency resolution is 5.38 [Hz] so it is satisfying for all notes above 82 [Hz] E2. Time resolution is 0.05 [s] and it is enough.

Cracow, January 2021

6. Result

Audio analysis works fine when the input .wav files are simple, that means they don’t contain lots of noises, reverbs or delays.

Due to lack of time and problems with communication between Python script and Arduino IDE we weren't able to overcome this. The biggest obstacle was collecting data with Arduino. Despite that device has been picking up received data properly, we didn't manage to process that data into an array and play them in real time in an audio workstation.

Despite this, our team managed to program an Arduino UNO for the MIDI controller that can play previously declared arrays with MIDI notes.

As we still wanted to improve project functionality, we decided to change project assumptions and add functions responsible for creating MIDI files.
The final project contains previously audio analysis and generation .mid files that can be further processed with audio workstations.

Cracow, January 2021