Department of Electrical Engineering,
National Tsing Hua University
Special Topic on Implementation
Research Abstract

Extensive Tests on Various Implementations of a Real-time Audio to MIDI Converter

音樂演奏即時 MIDI 轉錄器之多方實作與完整測試

Major Category: **System**

Group Number: **A412**
Advisor: **Prof. Yi-Wen Liu**
Members: **Kuei-Yuan Tsai, Jia-Sin Wun**

Research Period: **From 2023/09/11 to 2024/05/07**.

Abstract

This research focuses on using a software approach to achieve real-time audio to MIDI conversion, which requires finding the correct strength, pitch, and event of an input audio efficiently. Considering the fact that reliable real-time audio to MIDI applications can hardly be found, and that a partner project of our lab is in need of one, we decided to dive in and implement one ourselves. After more than 7 months of research and development, we were able to come up with a system that reaches an optimal accuracy of 0.965 for a fast-paced song at 145 bpm.

The scope of our project covers an implementation phase, followed by a testing phase. In the implementation phase, there are 4 main issues we had to address: (1) energy to velocity conversion (2) pitch detection (3) note segmentation (4) computation speed. For pitch detection, we decided to select 3 different algorithms: YIN [2], PYIN [3], and CREPE [4] for comparison. Various fixes and optimization approaches were added successively throughout development to boost performance and reduce latency.

In the testing phase, a series of experiments was conducted to assess the performance of the system across a range of input conditions, including 3 music tracks and 7 instruments, each with distinct character. This comprehensive evaluation was done to help compare the 3 different algorithms (YIN [2], PYIN [3], and CREPE [4]), and give us insights on our system's strengths, limitations, and potential areas for improvement.

From our test results, we conclude that it's hard to pick a best pitch detection algorithm, as each of them favor differently in song and instrument type. Still, we can state that the more advanced algorithms, PYIN [3] and CREPE [4], do perform better than YIN [2] on the whole. Up to this stage, our program shows fairly good performances with piano, flute, and violin (accuracy > 0.8), while needing much improvement with brass or vocal. Future plans for code revision are stated in the context, please refer to our GitHub page for project updates.

GitHub:

https://github.com/RyanTheBigO8/Realtime-AtoM-Converter

Demo:

https://drive.google.com/file/d/1NsKVpZWwAnOvoWY3wjgZOfSfWkcBRywa/view?usp=drive_link

Keywords:

Real-time audio processing, MIDI conversion, YIN algorithm, PYIN algorithm, CREPE, music technology, performance analysis.

Fig. 1 Project Layout

Contents

1. Motivation

MIDI, or Musical Instrument Digital Interface, is a universal protocol that facilitates communication of musical data between electronic instruments, computers, and other digital devices. Like most other protocols, MIDI comes with its own data format, where each unit of data is called a MIDI message. Each MIDI message stores at least 4 pieces of information: the "event" (8-bits), "timestamp" (16-bits), "note number" (8-bits), and "velocity" (8-bits) of a musical note. "Event" marks the begin or end of a note (x90 for NOTEON, x80 for NOTEOFF); "Timestamp" records when the event is triggered; "Note number" represents the pitch of a note (e.g. 60 = C4); "Velocity" indicates the strength of a note. Hence, each musical note can be described by a pair of MIDI messages, one marking its NOTEON event, and the other marking its NOTEOFF event.

The conversion of real-time audio to MIDI represents a pivotal point of convergence between creativity and technology, offering musicians and producers a unique opportunity for unparalleled musical exploration. The seamless translation of live audio input into MIDI data enables a transformative process that opens up a realm of possibilities, from expressive live performances to streamlined production workflows. The capacity to instantaneously capture musical ideas, extract elements from existing recordings, and interact with virtual instruments in real-time enables artists to transcend traditional constraints and explore novel avenues of expression. The integration of real-time audio to MIDI conversion enriches musical experiences in both studio and live settings, fostering innovation and pushing the boundaries of what is possible in the realm of sound.

Despite the vast benefits of MIDI, converting music from audio to the MIDI format is a daunting task. For most of the time, this operation has to be done manually to ensure precision of note lengths and pitches. Thanks to advancements in software technology and deep learning models, the reliability of automated audio to MIDI conversion has increased significantly. However, most of these conversion programs require lots of processing power and time, making them only capable of running off-line.

While there seems to be a software solution for everything nowadays, we found out that there exists a huge gap in the market for real-time audio to MIDI applications. Though a handful of them can still be found [1][5], they run on external micro-controllers and are subjected to performance and input limitations. Hence, we decided to take the challenge and implement one ourselves, one that runs natively, and can achieve maximal accuracy while being computationally fast.

2. Research Methodology

Our project consists of 2 phases. The first phase is implementation, which involves incorporating 3 different pitch detection algorithms into a custom-built framework that handles real-time audio to midi stream. The second phase is testing, where we assess the performance of our program with 3 songs, each performed by 7 different instruments.

2-1. Implementation

2-1.1 Programming Language

We chose to implement our program using **Python** for the following reasons:

- High level language which significantly reduces coding complexity
- Many open-source audio processing libraries available for use
- Well documented and a large community of users
- Compatible with the existing accompaniment system from our lab, which is also written in Python.

2-1.2 Audio Input and Sample Buffer

Among the most popular Python audio libraries, such as **Aubio**, **Pyaudio**, and **Sounddevice**, **Sounddevice** has emerged as the most stable choice for handling audio streams. The **InputStream** class is responsible for reading audio data at the specified sample rate and storing the samples in a buffer.

Upon the filling of the buffer, a frame is created, which then triggers the **audio_callback()** function for the purposes of pitch, velocity, and event detection. The buffer size is contingent upon the pitch detection algorithm employed. For instance, the buffer size for YIN and PYIN is 2048, while for CREPE it is 1024, in order to align with the dimensions of the ML model.

In order to guarantee real-time processing, it is essential that the program completes the processing of each frame (comprising 1024 or 2048 samples) before the next arrives, thus preventing input overflow. For example, with a standard sample rate of 44,100 *Hz* and a buffer of 2048 samples, the processing time per frame should be less than 46 *ms* (2048/44100). This guarantees uninterrupted frame processing and prevents the occurrence of dropped frames due to input overflow.

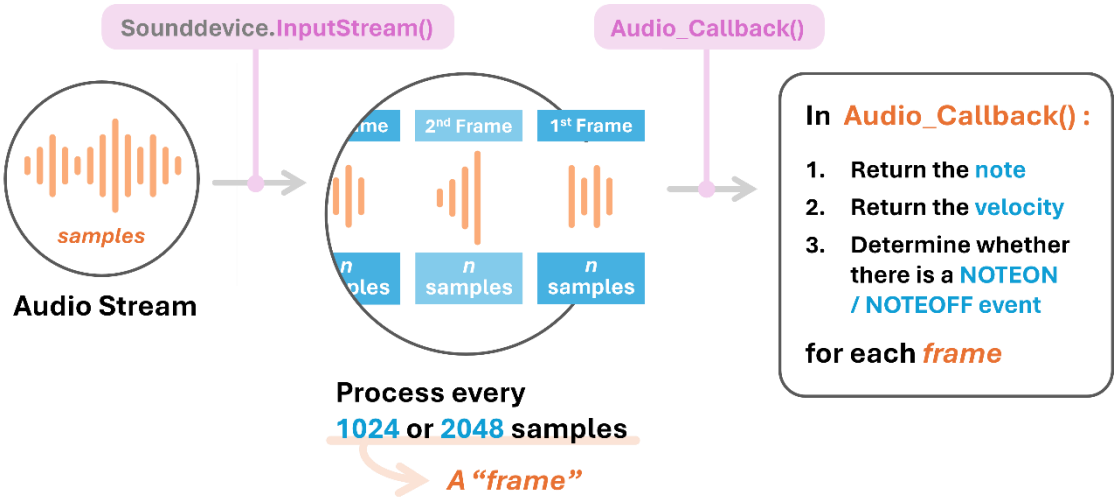

Fig. 2-1 Audio data flow control

2-1.3 The audio_callback() Function

As we have described earlier in advance, the **audio_callback()** function is the heart of our program. It is where all the output information: note, velocity, and event are generated. In the following context, we'll walk through the processes each frame goes through inside **audio_callback()**.

(1) Velocity

Recall that the *velocity* of a note can be derived from its strength, or energy. Based on this knowledge, our method is straightforward. First, we find the Root Mean Square (RMS) Energy of the frame, and then map the energy to a value between 0~127, which then represents the velocity.

(2) Note (Pitch Detection)

We utilize existing implementations for YIN and PYIN from Librosa functions yin() and pyin(). For CREPE, adjustments are made within the framework hosting pre-trained models. However, parameter tuning proved challenging, notably frame length for YIN/PYIN affecting accuracy and speed. Advanced algorithms like PYIN and CREPE offer confidence matrices to filter frequencies, necessitating meticulous threshold adjustments. Despite minimal coding for CREPE, parameter optimization for all algorithms demanded considerable effort.

(3) *Event*

Event detection is deciding whether a NOTEON or NOTEOFF occurs, or in simple words, note segmenting. This is the most difficult and yet to be improved part of our program. Segmenting notes properly means to not send additional notes and not miss notes. When it comes to implementation, we need to handle the following 5 scenarios as described in Table 2-1.

Table 2-1

Note Segmenting Scenarios

#	Scenario	Output Events
1	Currently muted, pitch detected	NOTEON(new note)
2	Currently pitched, mute detected	NOTEOFF(current note)
3	Current pitch ≠ new pitch	NOTEOFF(current note) NOTEON(new note)
4	Current pitch = new pitch, but they belong to the same note	Do nothing
5	Current pitch = new pitch, but they belong to two separate notes	NOTEOFF(current note) NOTEON(new note)

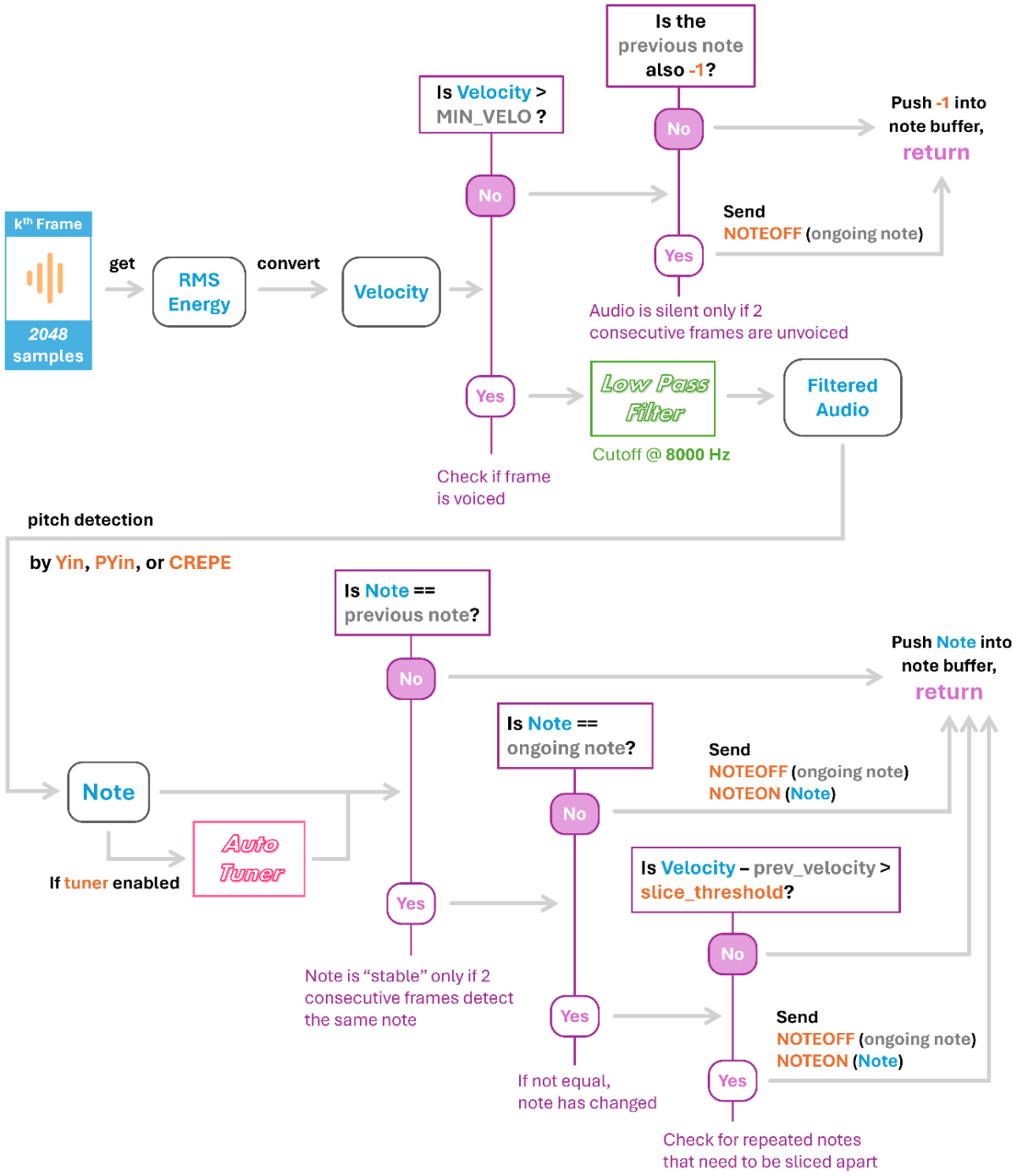

Fig. 2-2 Flow Diagram of **audio_callback()**

2-1.4 "Auto Tuner" Feature

The Auto Tuner is a supplementary function we added to support pitch detection. The way it works is that it auto-corrects MIDI notes out of key into the key specified by the user. For instance, if the specified key is C major, any detected pitch that falls out of key will be corrected into the nearest pitch in the C major scale.

2-2. Testing

In the testing phase, we evaluate and compare the performance of the 3 pitch detection

algorithms with 3 songs, each performed by 7 instruments. The songs and instruments were meticulously chosen such that they are characteristically unique, allowing our program to be examined comprehensively.

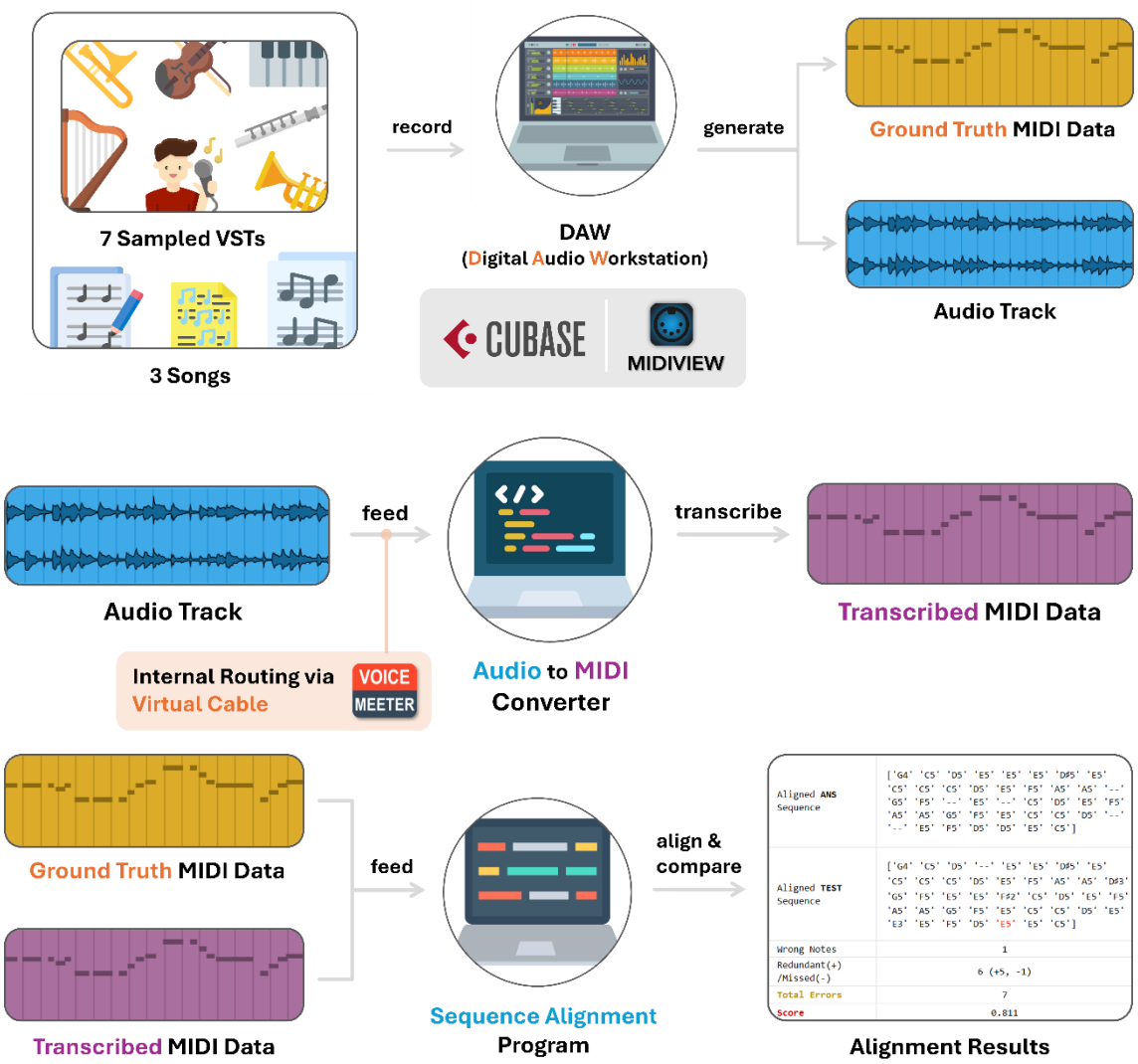

Fig. 2-3 Flow Diagram of Testing

3. Experiment Results

The accuracy of a performance is defined as the total number of errors, including those related to wrong pitch, redundancy, and missed notes, divided by the total number of notes in the sequence.

$$Accuracy = \frac{Errors}{Total\ Notes}$$

$$Errors = wrong\ pitch + redundent\ notes + missed\ notes$$

Table 3-1

The output format of a testing result

Song: You are my Sunshine (Total: 37 notes) **Instrument:** Piano **Algorithm:** YIN	
Aligned Ground Truth Sequence	[‘G4’ ‘C5’ ‘D5’ ‘E5’ ‘E5’ ‘E5’ ‘D♯5’ ‘E5’ ‘C5’ ‘C5’ ‘C5’ ‘D5’ ‘E5’ ‘F5’ ‘A5’ ‘A5’ ‘G5’ ‘F5’ ‘–' ‘–' ‘E5’ ‘–' ‘C5’ ‘D5’ ‘E5’ ‘F5’ ‘A5’ ‘A5’ ‘G5’ ‘F5’ ‘E5’ ‘C5’ ‘C5’ ‘–' ‘D5’ ‘E5’ ‘F5’ ‘D5’ ‘D5’ ‘E5’ ‘C5’]
Aligned Detected Sequence	[‘G4’ ‘C5’ ‘D5’ ‘E5’ ‘E5’ ‘E5’ ‘D♯5’ ‘E5’ ‘C5’ ‘C5’ ‘C5’ ‘D5’ ‘E5’ ‘F5’ ‘A5’ ‘A5’ ‘G5’ ‘F5’ ‘E5’ ‘E3’ ‘E5’ ‘A2’ ‘C5’ ‘D5’ ‘E5’ ‘F5’ ‘A5’ ‘A5’ ‘G5’ ‘F5’ ‘E5’ ‘C5’ ‘C5’ ‘D5’ ‘D5’ ‘E5’ ‘F5’ ‘–' ‘D5' 'E5' 'C5']
Wrong Notes	0
Redundant Notes	4
Missed Notes	1
Total Errors	5
Accuracy	0.865

Note that we set the score 0.7 as the lowest acceptable score. Any experiment that scores below 0.7 is considered a failed detection, thus, there is no point comparing those values. Due to space limitations, only a summary of our testing result is included below. For the full result, please refer to our *excel file*.

3-1. General Analysis

From the 63 test results, the very first thing we discovered was that most detection errors come from redundant or missed notes (note segmenting errors) rather than misdetection of pitch. This means that our program can easily determine the right pitch but has trouble outputting the correct number of notes for specific instruments.

It seems at first that these note segmentation errors are merely due to imperfections in our segmentation decision-making script and have little to do with the chosen pitch.

In the following two sub-sections, we’ll show that YIN is the fastest in computation while lacking a double-check mechanism, making it especially sensitive to any change in the audio. Hence, it is prone to outputting more notes than required. On the other hand, PYIN and CREPE are slower in computation while being able to discard unconfident pitches, making them more likely to miss notes.

3-2. Computation Speed

Computation speed of the 3 algorithms were measured by running them with equal buffer sizes and counting the number of callbacks in a fixed time length. More callbacks mean that more frames were processed in the same amount of time, implying a faster computation speed. We found that the number of callbacks were YIN: PYIN: CREPE ≈ 2 : 1 : 1.5, hence we arrive at YIN > CREPE > PYIN for speed.

3-3. Noise Rejection Ability

As mentioned before, PYIN and CREPE return confidence matrices that help filter out unsure detections while YIN does not. Therefore, we can already infer that PYIN and CREPE are less susceptible to noise from an implementation aspect. Still, we have to make sure that our test results also support this claim.

The song, “You are My Sunshine”, as described in Table 2-3, was purposely designed to test the noise resistance of the PDAs by adding a drum loop in the background. From Fig. 3-2(c), it is indeed true that PYIN and CREPE outperform YIN for nearly all instruments, very likely thanks to their noise-rejecting capabilities.

3-4. Pitch Detection Accuracy

Though all three algorithms are fairly accurate in identifying pitches, differences in accuracy can still be spotted when they are put under the stress test, namely the song “DoReMi”. “DoReMi” includes 10 different pitch types and contains parts with fast-changing pitches. Table 3-2 records the number of pitch misdetections in this song.

Table 3-2

Pitch Misdetections from “DoReMi”

Instrument	Algorithm		
	YIN	PYIN	CREPE
Piano	0	0	0
Flute	2 (half)	1 (octave)	1 (half)
Violin	0	0	0
Harp	1 (half)	1 (half)	0
Trumpet	1 (half)	0	0
Trombone	0	1 (octave)	0
Women	1 (half)	0	0

From Table 3-2, the superiority of CREPE in pitch detection accuracy over the other two is apparent. The term inside the brackets denotes the type of pitch error. “Half” stands for missing the pitch by a half-step (e.g. mistake C4 for C4#); “Octave” refers to missing the pitch by an octave (e.g. mistake C4 for C3), both of which are common mistakes in pitch detection.

3-5. Effect of Different Instruments

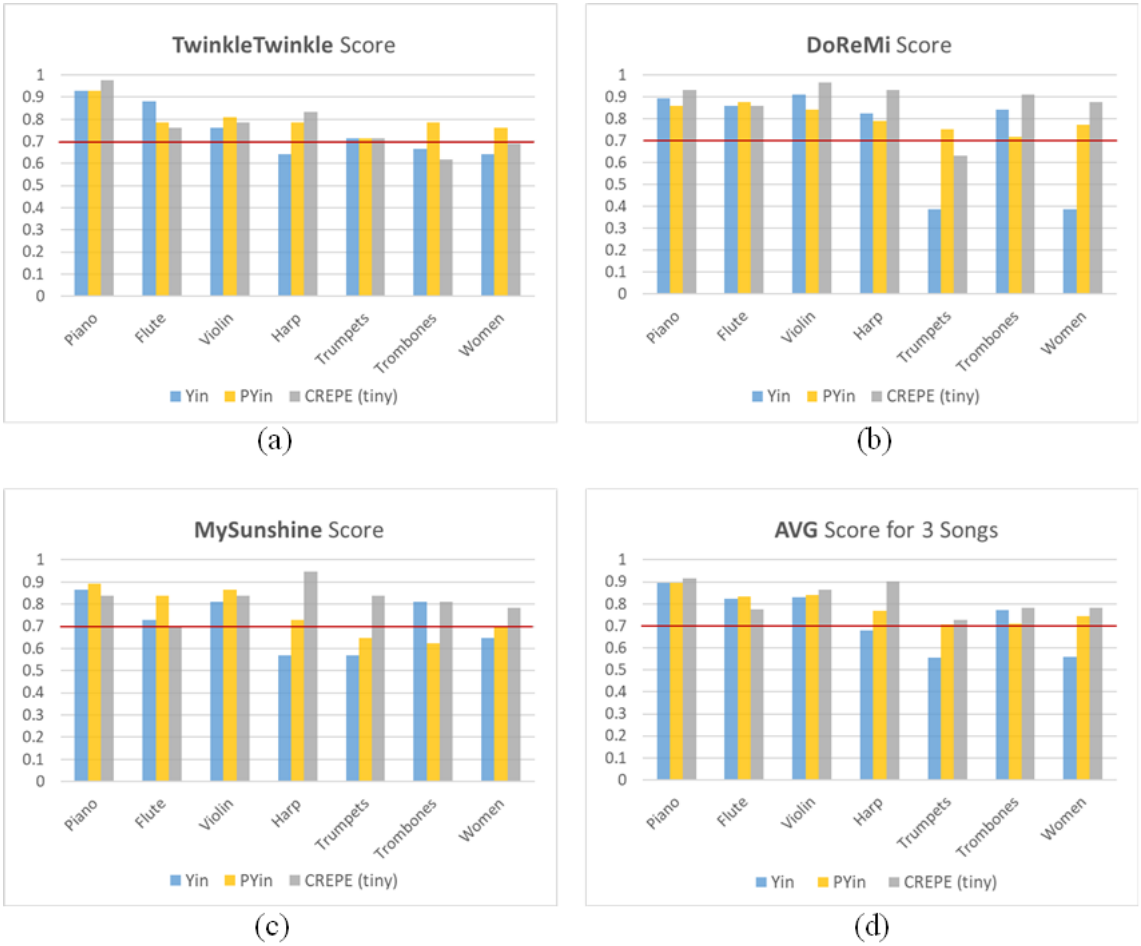

Fig. 3-1 Note Alignment Score with Different PDAs

Fig. 3-2(d) gives us an idea of how well our system performs when faced with different instruments. Generally, our system interprets the piano, flute, and violin fairly accurately, where our system stands out the most with the piano. The best result appears in "Twinkle Twinkle Little Star" at a near perfect score of 0.976, with CREPE selected as the PDA and piano being the instrument. The other 4 instruments, however, turned out to be rather difficult for our system to interpret accurately. Below are two interesting observations we collected.

4. Conclusion

- It's hard to pick the best pitch detection algorithm among YIN, PYIN, and CREPE, as there isn't one that leads in performance in all categories. However, it is still valid to claim that the more advanced algorithms, PYIN and CREPE, generally give rise to better performance compared to YIN.
- Regarding the characteristic comparison of the 3 PDAs, we have CREPE being the most pitch accurate and reverberance-resistant, CREPE and PYIN being less-sensitive to noise, and YIN being the most computationally efficient.

- Each instrument we tested is unique in timbre, with some of them compatible with our program and others not so much. We will further revise our system's framework, such that it becomes more robust to different input sound sources.
- This research employs a combination of empirical experimentation and theoretical analysis to contribute to the advancement of real-time audio-to-MIDI conversion technology. By elucidating the capabilities, limitations, and applications of this technology, our aim is to stimulate further research and innovation in this nascent field at the intersection of music, technology, and data.

5. Reference

[1] Balberchak, B. S. Real-Time Audio-MIDI Controller. https://digitalcommons.calpoly.edu/cpesp/285, April 2018.

[2] Alain de Cheveigne´, Hideki Kawahara, "YIN, a fundamental frequency estimator for speech and music," The Journal of the Acoustical Society of America 111(4):1917-30, May 2002.

[3] Matthias Mauch and Simon Dixon, "PYIN: A FUNDAMENTAL FREQUENCY ESTIMATOR USING PROBABILISTIC THRESHOLD DISTRIBUTIONS" 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Florence, Italy, May 2014.

[4] Jong Wook Kim, Justin Salamon, Peter Li, Juan Pablo Bello, CCREPE: A CONVOLUTIONAL REPRESENTATION FOR PITCH ESTIMATION" arXiv:1802.06182v1 "February 2018.

[5] Julián Markow, " AUDIO TO MIDI in real time " [Online], Available: jomarkow/Audio-to-MIDI: Real time Audio to MIDI converter. Convert your sound into an editable music document. (github.com)

[6] Kit ARMSTRONG, Tzu-Ching HUNG, Ji-Xuan HUANG, and Yi-Wen LIU, "Real-time piano accompaniment model trained on and evaluated according to human ensemble characteristics," accepted by Sound and Music Computing, Porto, Portugal, 2024.

6. Thought

Taking part in the two-semester implementation course was truly an extraordinary journey for us. Initially, when Prof. Liu invited us to join their regular meetings, we felt overwhelmed by the complexity of the content, often leaving us feeling lost and confused. It's astonishing to reflect on how, after two semesters, we found ourselves actively contributing to a project alongside master's and PhD students.

Throughout the course, the selection of topics underwent numerous revisions, and we encountered various challenges before finally settling on a topic that truly ignited our passion. This journey allowed us to gain a deeper understanding of our own limitations and weaknesses, prompting us to embark on a journey of self-improvement through dedicated study.

The success of our project is undoubtedly owed to the invaluable guidance provided by our advising professor, Yi-Wen Liu. We are also grateful for the support and contributions of all the other team members in our group, whose collective efforts were instrumental in achieving our goals.