Absolute Pitch: Automatic Sheet Music Navigation Based on Real-time Music Recognition

Nolan H. Woo

Saratoga High School
Saratoga, California, USA
nolan.h.woo@gmail.com

2024 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD) | 979-8-3503-8790-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICABCD62167.2024.10645276

***Abstract*—Music has been the universal language of the world for centuries. However, for all these years, musicians have faced a challenge that remains unresolved: the task of turning pages while playing their instruments. The sound of page turning during performances often disturbs the music and can cause abrupt pauses, and on some occasions, pages can fall out, ruining entire performances.**

The goal of this research is to solve this issue and to develop an app that digitizes sheet music and automatically adjusts the displayed music at the rate of the musician's playing. Using the user's MusicXML file, the music is displayed on the device with the Android graphics API. After activating the microphone, the algorithm uses the properties of sound to analyze the audio signal, employing tools like the Fourier Transform. By converting audio signals into frequency components, the algorithm identifies harmonics in order to calculate the dominant pitch. Instead of just computing the frequency with the highest intensity, we calculate the average power of each note across all octaves to find the fundamental frequency. Once the pitch of the note is found, the app scrolls through the music and adjusts to the performer's playing. After testing the algorithm on different examples of sheet music, we find that the program is accurate in detecting pitch and displaying music in real-time.

Keywords—music recognition, pitch detection, real-time processing

I. Introduction

As the principal cellist in my school's orchestra, I was assigned a crucial solo part in the performance of Jessie Montgomery's "Banner." I prepared extensively, rehearsing it numerous times. However, during our performance, an unexpected hiccup occurred—I completely missed the page turn right before my prominent solo. The flimsy pages had slipped from my fingers and I completely botched my solo. This catastrophe made me realize how problematic page turns could be.

During a concert, the performer usually has to pause momentarily, lifting their hand off their instrument and flipping the pages. This causes the flow of the notes to stop abruptly. Although composers sometimes try to coordinate the page turns with rests in the music, this is not always plausible, and the sound of the page turning can ruin the beauty of the music. Additionally, loose pages can fall out onto the ground and cause a mess.

A solution to this problem is using a tablet and a page turning foot pedal [1]. However, when using this tool during a concert, the pedal often refuses to work properly, requiring a couple aggressive stomps. As documented in numerous online articles [2], these devices are infamous for becoming stuck or malfunctioning at the worst time.

The goal of this paper is to develop an app to digitize sheet music and automatically adjust the displayed music at the rate of the musician's playing. The algorithm will determine when the music is being played by the sequence of notes it detects. When the musician plays the notes that are shown, the app will automatically scroll through each measure at the speed of the performer. By freeing musicians from the stress of manually flipping pages on time, the musicians will be able to keep their hands on the instrument and focus on producing high quality music.

II. Related Works

There are several approaches to address page turning problems for musicians. Page turning foot pedal [1] is one approach, but it often suffers from mechanical issues. Furthermore, musicians hire dedicated page turners, but often a mistake leads to disastrous impact on music performance [3].

In the past, people tried several approaches to build a score-following system. Dorfer et al. tried to address the problem by applying a multimodal convolutional neural network on the spectrogram of a given audio signal [4]. To address various temporal variations such as ornaments or chord embellishments, Dorfer et al. also introduced dynamic time warping [5]. On the other hand, Chen and Jang proposed an audio-score alignment process [6]. Noto et al. tried to build a score-following system using eye-gaze and keying information [7]. Terasaki et al. took a similar approach using a Hidden Markov Model [8]. Tabone et al. proposed a gaze prediction model that uses Kalman filtering to predict when to turn pages [9].

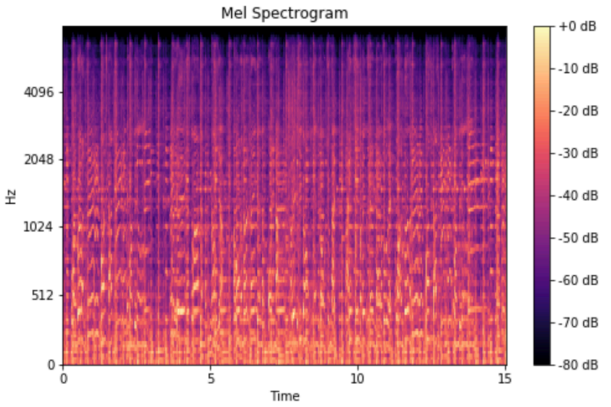

Fig. 1. Illustration of a Mel Spectrogram

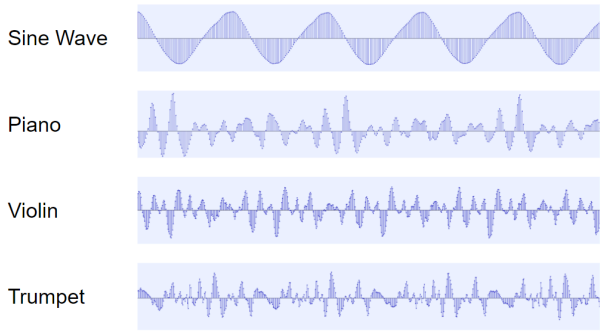

Fig. 2. Illustration of the Audio Signals of a Sine Wave, Piano, Violin, and Trumpet

III. Methodology

A. Harmonic Detection

To achieve this goal, our research delves into the fundamental properties of sound. The three main properties of sound, which are wavelength, amplitude, and frequency, are integral to analyze any audio signal. Furthermore, A4 at 440 hertz, the international pitch standard, is used as the basis to determine which note is being played [10]. To record the audio, we use microphones as transducers, converting sound into electrical signals, which can then be recorded in formats such as WAV and MP3.

Furthermore, we use the Fourier Transform, a mathematical operation that converts a time domain function into its frequency domain counterpart. This transformation is crucial for analyzing signals like audio, initially represented in the time domain, as it allows us to identify and extract frequency information.

To investigate deeper into the algorithmic aspects, we leverage tools such as the mel spectrogram (as shown in Fig. 1) to break down the audio signal into frequency components and map them onto the mel scale, providing a representation we can easily understand. In our algorithm designs, we specifically focus on tasks like harmonic detection, where we identify and analyze the harmonics present in the signal. Additionally, dominant pitch detection is employed to pinpoint the primary pitch in the audio, offering insights into the overall tonal structure. Lastly, we use high-pass and low-pass equalization (EQ) curves to filter out non-informative segments and enhance the precision of our analyses. This approach enables us to extract meaningful information from audio signals.

B. Challenges Overcome

Numerous challenges were experienced in this project. One of the biggest difficulties was that originally, the app was not able to detect the pitch very accurately. Some of the tones had a low frequency, which required collecting many samples to capture a wavelength reliably for the Fourier Transform. Furthermore, the original implementation was not accurate in calculating the octave of the note. Through a literature search, we discovered that the frequency characteristics, called the timbre, of different types of instruments are all different because they have distinct natural overtones or harmonics [11]. As shown in Fig. 2, when a note is played, its sound wave is actually made up of multiple frequencies, not just a single frequency.

To overcome this challenge, we decided to leverage the overtone itself in the detection algorithm. Instead of computing the frequency with the highest intensity, we calculated the average power of each note first by summing up the power of the frequency bank of each note for all octaves. With this, we were able to detect a dominating note more reliably. Once we detected the dominating note, we calculated the correct octave by finding the fundamental frequency. With these improvements, the app detected the pitch more precisely.

Additionally, if the same notes were played consecutively, the app was not able to detect if there was a change in note. As a result, a new array was made which contained only notes with different tones. By using this method, the app would be able to function properly, since consecutive notes with the same pitch could be differentiated.

Furthermore, many random notes were being detected by the app. It was detecting many lower-pitched and softer sounds in the surrounding environment. To solve this problem, a function that detects the amplitude was made. By setting the amplitude to be only in a specific range above the arbitrary notes, the audio was filtered out, and the background noises were not detected.

Lastly, the music displayed on the screen was not moving properly. The app's intended purpose was to smoothly scroll through the music along with the performer. However, there were many large, abrupt jumps from measure to measure. The cause of this problem was that the distance between each measure was not correlated with the distance between each note. As a solution, the distance between each measure was changed, and the function to calculate the distance to travel was updated.

IV. Experiments and Results

A. Experimental Setup

To perform this research, we utilized a range of hardware and software components to achieve the study's objectives. For developing the app, we used a desktop computer, and we used an Android tablet as a target mobile device to perform evaluations. Accordingly, we used Android Studio, an integrated development environment (IDE) designed for Android app development, to write the code. Furthermore, we utilized a USB-C cable to download data from the desktop to the tablet. Beyond computing resources, the

research involved using the Mini Piano Lite app to evaluate the accuracy of the algorithm by emulating different musical instruments. Moreover, MusicXML files, the digital format for musical notations, were used to import information about the sheet music from the user. These files allowed the algorithm to display the notes and rhythms on the screen and function properly.

B. Design and Development

Using the aforementioned materials, we began to develop this application using the following steps, eventually building an application flow illustrated in Fig. 3.

1. Creating a User Interface:
   a. Use the Android Button API to integrate a button into the app.
   b. Configure the button to prompt the user to upload a MusicXML file.
2. Parsing and Storing Music Data:
   a. Develop a class capable of storing information about the music, including the notes, pitch, duration, time signature, and more.
   b. Extract and analyze data from the MusicXML file using the created class.
3. Visual Representation:
   a. Create a function to determine note positions based on octave and pitch.
   b. Display notes according to their pitch and duration.
   c. Use the Android graphics API to draw music elements such as clefs and notes.
4. Dynamic Music Flow:
   a. Create a mechanism to render the music horizontally at the tempo at which it is played.
   b. Calculate the time required to play a measure and adjust the music flow accordingly, adjusting to the speed of the performer.
5. Microphone Integration:
   a. Utilize the AudioRecord interface to capture audio input.
6. Frequency Domain Analysis through Short-Time Fourier Transform:
   a. Create a Fast Fourier Transform (FFT) class [12] to convert audio signals from the time domain to the frequency domain [13].
   b. Implement a class for handling complex numbers, a crucial component in FFT.
   c. Create a threshold to filter out background noise.
7. Harmonic Detection and Dominant Pitch Detection:
   a. Delve into tools like the mel spectrogram to break down the audio signal and map them onto the mel scale.
   b. Identify and analyze the harmonics present in the signal.
   c. Utilize dominant pitch detection to pinpoint the primary pitch in the audio, offering insights into the overall tonal structure.
      i. To calculate average intensity, sum up the power of the frequency bank for each note across all octaves to enhance reliability in detecting dominating notes.
      ii. Use the calculated average power to identify the note with the highest dominance for improved pitch detection.

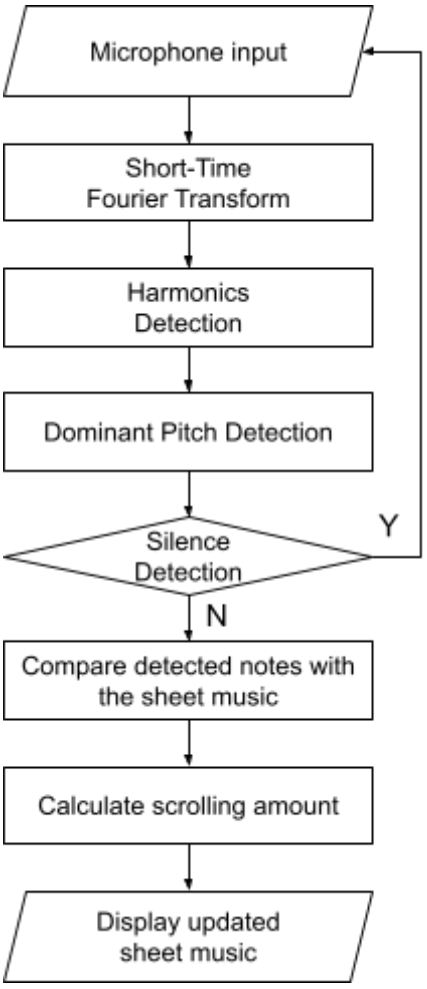

Fig. 3. Overall execution flow

      iii. Find the fundamental frequency associated with the detected dominating note for accurate calculation of the correct octave.
8. Silence Detection
   a. Employ high-pass and low-pass equalization (EQ) curves to filter out non-informative segments.
9. User Interface Display:
   a. Integrate a TextView element to present real-time information to the user, displaying pitch, octave, frequency, and amplitude of detected notes.
10. Note Sequencing:
   a. Track note sequences by maintaining a list of played notes.
   b. Identify changes in note tones and create a new array excluding consecutively repeated notes.

C. Testing

To validate the accuracy of our algorithm, we ran tests on the app using the following steps.

1. Find examples of sheet music, where some are simple and others are complicated.

2. Enable Developer Options on the tablet by clicking the build number seven times.
3. Launch the app on the Samsung tablet by connecting the computer and the tablet using the USB cable.
4. Run the program and first test what tone is being played.
5. Start playing an entire octave on the Mini Piano Lite app from C4 to B4 using the sine wave, piano, violin, trumpet options.
6. Record the data of the correct frequency played and the frequency that was detected, which is shown in the top left corner of the screen.
7. Afterwards, import each MusicXML file that is to be tested.
8. Make sure the app shows the correct notes and their corresponding duration.
9. Once the app displays the music on the screen, start playing the music.
10. Record data about whether the music displays the correct notes, durations, and page turning.
11. If it satisfies all the requirements, the app successfully finishes its task.
12. If the app fails, further improvements and debugging should be done to the code to make sure that it works flawlessly.

D. Results

The app successfully performed its intended task by displaying and recognizing the notes correctly and moving the music at the same speed as the performer. Furthermore, the detected frequency of the notes was accurate, even for different instruments that we used, including a sine wave, piano, violin, and trumpet. Under a well-controlled environment, our algorithm was able to detect the correct frequency for all of the test cases. There were no outliers in this stage of testing.

Table 1, 2, 3, and 4 show how the data for how the algorithm worked with a sine wave, a piano, a violin, and a trumpet, respectively, which we generated from the Mini Piano Lite app. The third column represents our original algorithm where we tried to find the frequency bank with the highest intensity, and the fourth column represents the final algorithm where we considered the fundamental frequency. As seen in Figure 4, 5, 6, and 7, the blue dots, which represent the original algorithm (frequency (Hz) with the highest intensity), do not match up with the correct frequency, while the red dots, representing the current algorithm, match up very well.

Table 2 clarifies the previous point that just calculating the frequency with the highest intensity would lead to inaccurate calculations when we use a real-world instrument, as shown in the third column.

Throughout the testing, the program worked flawlessly with many examples of music. To test the app, we used “Twinkle, Twinkle Little Star,” “Old MacDonald,” “Itsy Bitsy Spider,” “Bach Cello Suite No. 1 Prelude,” and “Eine Kleine Nachtmusik” as our MusicXML files. In all five examples of music, they each displayed the correct notes, duration, and flowed smoothly when the notes were played on the Mini Piano Lite app. The algorithm operated at real time with accurate frequency detection, even while switching between the various instruments. If the musician stopped playing, the app functioned properly and did not continue to scroll.

Figure 8 shows what happens when the app is first opened. The display shows a clef, a five-line staff, and a button that is used to choose what music to import. The app correctly displayed these components as intended by the code.

Figure 9 shows what happens when the “Choose MusicXML” button is clicked. A new display is shown where MusicXML files can be chosen. The app takes the user to this screen, so that they can choose a MusicXML file of their choice. When a file is selected, the app takes data from this file and reverts back to the main screen.

Figure 10 shows when TwinkleLittleStar.xml was chosen from the previous image. This file contains the song “Twinkle, Twinkle Little Star,” one example of music that was used in this project. During the testing, the program was able to follow along with the performer very well and function properly. The code detected the audio signals very well and the notes that were shown correctly corresponded to what the musician was playing.

TABLE 1. Generated Frequency Versus Detected Frequency (Sine Wave)

Correct Note	Correct Frequency (Hz)	Frequency (Hz) With the Highest Intensity	Fundamental Frequency (Hz) of Note With Highest Intensity
C4	261.63	263.78	261.63
C#4	277.18	279.93	277.18
D4	293.66	296.08	293.66
D#4	311.13	312.23	311.13
E4	329.63	328.38	329.63
F4	349.23	349.91	349.23
F#4	369.99	371.45	369.99
G4	392.00	392.98	392.00
G#4	415.30	414.51	415.30
A4	440.00	441.43	440.00
A#4	466.16	468.35	466.16
B4	493.88	495.26	493.88

TABLE 2. Generated Frequency Versus Detected Frequency (Piano)

Correct Note	Correct Frequency (Hz)	Frequency (Hz) With the Highest Intensity	Fundamental Frequency (Hz) of Note With Highest Intensity
C4	261.63	1706.51	261.63
C#4	277.18	1948.75	277.18
D4	293.66	1916.46	293.66
D#4	311.13	2029.50	311.13
E4	329.63	3962.11	329.63
F4	349.23	699.83	349.23
F#4	369.99	2040.27	369.99
G4	392.00	1378.13	392.00
G#4	415.30	414.51	415.30
A4	440.00	1545.01	440.00
A#4	466.16	1636.52	466.16
B4	493.88	1733.42	493.88

TABLE 3. Generated Frequency Versus Detected Frequency (Violin)

Correct Note	Correct Frequency (Hz)	Frequency (Hz) With the Highest Intensity	Fundamental Frequency (Hz) of Note With Highest Intensity
C4	261.63	1566.54	261.63
C#4	277.18	274.55	277.18
D4	293.66	1749.57	293.66
D#4	311.13	1243.54	311.13
E4	329.63	328.38	329.63
F4	349.23	349.91	349.23
F#4	369.99	925.93	369.99
G4	392.00	387.60	392.00
G#4	415.30	414.51	415.30
A4	440.00	1103.58	440.00
A#4	466.16	925.93	466.16
B4	493.88	1232.78	493.88

TABLE 4. Generated Frequency Versus Detected Frequency (Trumpet)

Correct Note	Correct Frequency (Hz)	Frequency (Hz) With the Highest Intensity	Fundamental Frequency (Hz) of Note With Highest Intensity
C4	261.63	527.56	261.63
C#4	277.18	559.86	277.18
D4	293.66	586.78	293.66
D#4	311.13	942.08	311.13
E4	329.63	667.53	329.63
F4	349.23	1582.69	349.23
F#4	369.99	1480.41	369.99
G4	392.00	791.35	392.00
G#4	415.30	1469.64	415.30
A4	440.00	667.53	440.00
A#4	466.16	936.69	466.16
B4	493.88	995.91	493.88

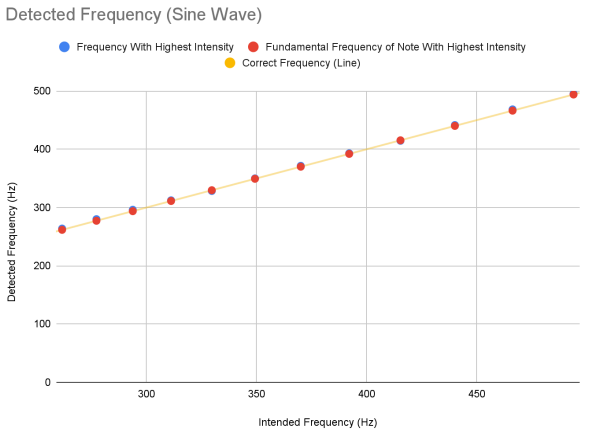

Fig. 4. Scatter Chart of Table 1

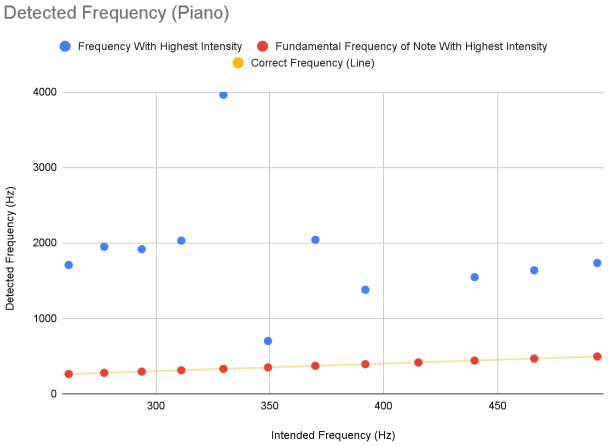

Fig. 5. Scatter Chart of Table 2

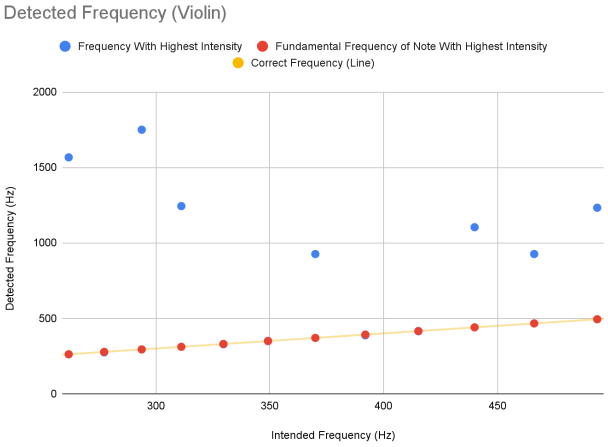

Fig. 6. Scatter Chart of Table 3

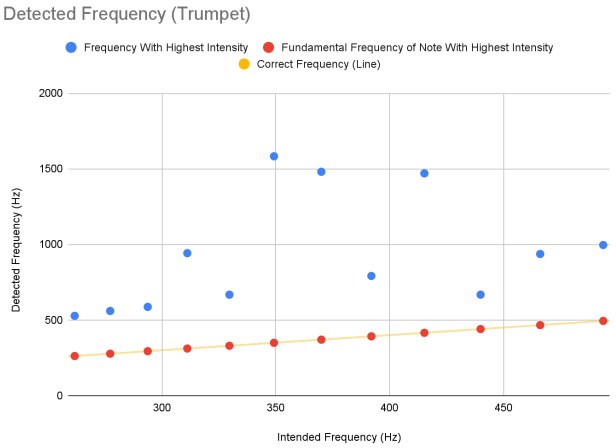

Fig. 7. Scatter Chart of Table 4

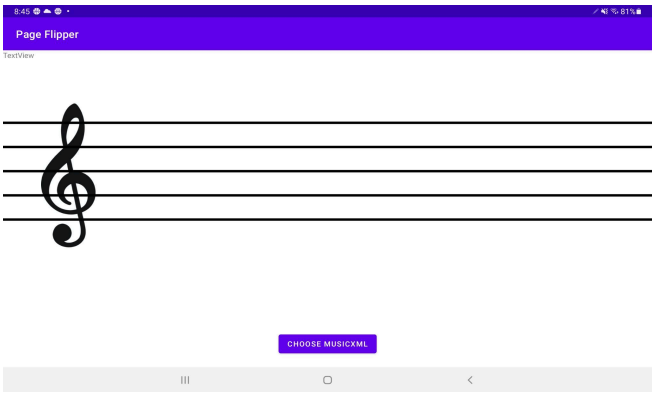

Fig. 8. Screenshot of App When First Opened

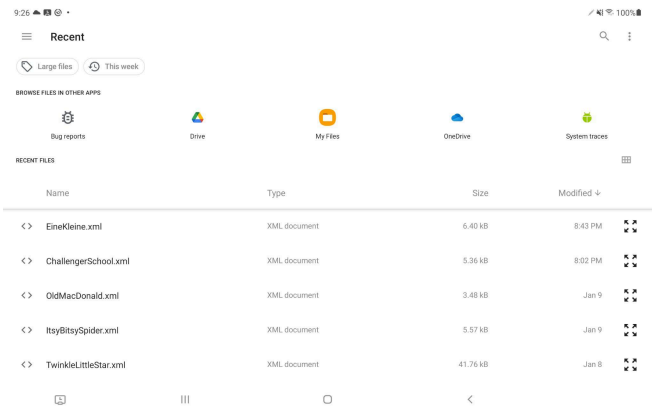

Fig. 9. Screenshot of App When Selecting a MusicXML file

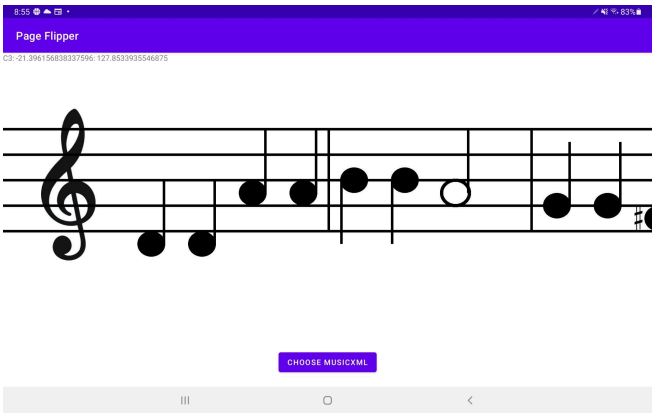

Fig. 10. Screenshot of App When Playing “Twinkle, Twinkle Little Star”

V. Further Research

The current app is limited to finding a single dominant pitch at a given time. To make the application more widely useful, the following improvements may need to be made.

One possible improvement to this project is to incorporate more complex elements of music [14]. Musical ornaments, which include trills, glissandos, and turns, can be added, so that more complicated music could be used in the display. Furthermore, chords, in which multiple tones are played at the same time, can be included to improve this project [15-17]. This may require more sophisticated signal processing algorithms or machine learning algorithms. Additionally, creating multiple parts can be another improvement. For example, piano music has two hands, and the app can display both of the hands.

Further research includes optimizing this algorithm for orchestras. In many orchestras, the musicians always have a problem with flipping pages. Instead of detecting the notes of a single player, the program can recognize the entire playing of the orchestra. In the future, the app can be altered, allowing the sound of multiple instruments played simultaneously to be interpreted by the code. Other further research includes creating and implementing a precise machine learning model. On the other hand, optical music recognition [18] can be used so that the program can import sheet music by simply taking a picture of printed sheet music [19-20]. The program can become more flexible, convenient, and user-friendly.

VI. Conclusion

During a performance, musicians have to pause momentarily, lifting their hand off their instrument and flipping the pages. The difficulties of page turning have plagued musicians for decades, causing loud noises and falling sheets. The goal of this paper was to develop an app to digitize sheet music and automatically adjust the displayed music at the rate of the musician’s playing.

Using the Fast Fourier Transform, harmonic detection, and dominant pitch detection, this program was able to accurately calculate the frequency of the notes and convert it to the pitch and octave. Furthermore, the application proficiently tracked note sequences, ensuring the music was properly displayed on the screen. The app demonstrated its functionality with various examples of music that had varied notes, time signatures, and key signatures. For instance, during the testing of "Bach Cello Suite No. 1 Prelude," the displayed notes, including G3, D4, and B4, were accurate. Lastly, corresponding durations were appropriately displayed, and the page smoothly turned in sync with the performer.

Acknowledgment

We express our gratitude to the reviewers for dedicating their time and effort to reviewing the manuscript. We value all the comments and suggestions provided, as they have contributed to enhancing the quality of our work.

References

[1] PageFlip, “Bluetooth Page Turner Pedal,” https://www.pageflip.com/, n.d.
[2] G. Knowles, “Page turner pedal on iPad isn't turning the pages,” https://www.dropboxforum.com/t5/Delete-edit-and-organize/Page-turner-pedal-on-iPad-isn-t-turning-the-pages/td-p/588889/page/3, April 2022
[3] A. Todorova, “Pianist page turning TOP 1 fail,” https://www.youtube.com/watch?v=xn8pmz8JJoA, June 2021.
[4] M. Dorfer, A. Arzt, and G. Widmer, “Towards score following in sheet music images,” In Proceedings of the International Society for Music Information Retrieval Conference (ISMIR) (New York, NY), 2016.
[5] M. Dorfer, A. Arzt, and G. Widmer, “Learning audio-sheet music correspondences for score identification and offline alignment,” In International Society for Music Information Retrieval (Suzhou). doi: 10.5334/tismir.12, 2017
[6] C. Chen and J. S. R. Jang, “An effective method for audio-to-score alignment using onsets and modified constant Q spectra,” Multimedia Tools Appl. 78, 2017–2044. doi: 10.1007/s11042-018-6349-y, 2019
[7] K. Noto, and Y. K H., Takegawa, “Adaptive score-following system by integrating gaze information,” In Proceedings of the 16th Sound and Music Computing Conference (Málaga), 2019
[8] S. Terasaki, Y. Takegawa, and K. Hirata, “Proposal of score-following reflecting gaze information on cost of DP matching,” In International Computer Music Conference Proceedings (Shanghai), 2017.
[9] T. André, B. Alexandra, and C. Stefania, “Automated page turner for musicians,” Frontiers in Artificial Intelligence, vol 3., 2020.
[10] R. M. Mottola, “Table of Musical Notes and Their Frequencies and Wavelengths,” https://www.liutaiomottola.com/formulae/freqtab.htm, January 2020
[11] N. Scoggin, “Barron's AP Music theory,” Hauppauge, NY: Barron's., 2018
[12] K. Wayne, “FFT.Java,” https://introcs.cs.princeton.edu/java/97data/FFT.java, February 2011
[13] A. Orobator, “How to recognize a musical tone in Android?” https://stackoverflow.com/questions/27528031/how-to-recognize-a-musical-tone-in-android. , December 2014.
[14] D. Farrant, “Types of musical notes: Hello Music theory,” https://hellomusictheory.com/learn/types-of-musical-notes/, February 2021.
[15] Y. Ueda, Y. Uchiyama, T. Nishimoto, N. Ono, and S. Sagayama, “HMM-based approach for automatic chord detection using refined acoustic features,” In 2010 IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 5518-5521).
[16] R. Hernández, A. Guerrero, and J.E. Macías-Díaz, “A template-based algorithm by geometric means for the automatic and efficient recognition of music chords,” Evolutionary Intelligence 17, 2024.
[17] W.M. Szeto, and M.H. Wong, “A graph-theoretical approach for pattern matching in post-tonal music analysis,” Journal of New Music Research, 35(4), pp.307-321, 2006
[18] J. Calvo-Zaragoza, J. Hajič Jr, and A. Pacha, "Understanding optical music recognition," ACM Computing Surveys (CSUR) 53.4 (2020): 1-35.
[19] Yoyo, C. Liebhardt, and S. Samuel, “BreezeWhite/oemer: v0.1.7,” Zenodo, October 2023.
[20] A. Hemdan, A. Mohamed, K. Mohamed, and A. Mahboub, “Orchestra,” GitHub, 2020. https://github.com/AbdallahHemdan/Orchestra