SoundSync

Authors: Caleb Lille, Rohan Raavi, Sanjana Shriram
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— In music performances, distractions often disrupt the flow of musical expression. One prominent challenge is page flipping. Current solutions include Bluetooth foot pedal page turners and human assistants. However, performances can be disrupted by foot pedal devices, while human page turners can obstruct the musician's view and cause physical sheet music to fall. SoundSync uses an eye tracking camera and a microphone to capture user's gaze and audio input to autonomously turn a page. Through a decision logic program, these inputs determine the best window for turning pages with 95% accuracy. SoundSync uses state of the art alignment algorithms in combination with eye tracking to achieve hands-free operations and enhance the user experience.**

***Index Terms*— Audio Alignment, Score Following, Eye Tracking, Head Tracking, Music, Digital Music, Sheet Music.**

1 INTRODUCTION

Musicians face a variety of distractions that deduct from the beauty of the music they are making. The most common problem musicians face is turning a page in a manner that doesn't detract from the music. With the advent of digital music displays such as tablets and iPads, page turning technology is evolving.

Page turning technologies have been on the rise in the music industry. The most widely available options include a foot pedal page turner and a physical human page turner. A foot pedal page turner is a device where the musician presses the foot sensor to turn a digital page of music. It uses a Bluetooth connection to a digital tablet and can be a convenient hands-free page turning solution for musicians.

However, current technologies have their limitations. For example, operating a foot pedal results in a loud foot tap to signal the page turn. This can be distracting during a performance. In some cases, the foot pedal does not work and can require a second tap. A human page turner can obstruct the player's view and introduces the risk of dropping physical sheet music during a performance.

SoundSync is a digital page turning system that utilizes visual and audio inputs to determine when to flip the page. Both input streams are fed into different models that track where the user is located in the music. These results are fed into a decision logic program that decides when to flip the page. Digital sheet music is displayed on a laptop. Lastly, SoundSync uses a Windows laptop to handle all the data processing. With a focus on accessibility and inclusivity, SoundSync aims to provide an inclusive streamlined music making experience for all musicians.

2 USE-CASE REQUIREMENTS

SoundSync was designed with accessibility as a top priority. The social implication of SoundSync is that more people of varying backgrounds can enjoy playing hands-free. The resources that most people have access to have also been considered when designing SoundSync. As of 2019, 73% of adults in the United States owned laptops or personal computers[4]. With accessibility in mind, laptops come out ahead of alternatives like iPads and tablets.

SoundSync exclusively uses audio and visual inputs making it accessible to those who cannot operate a foot pedal page turner or similar technologies. Since it is fully digital, SoundSync is less disruptive than traditional methods such as loud foot pedals and physical page turning all while guaranteeing that the musical ambiance remains undisturbed.

2.1 Sensor Input Use Case

The system must be able to take in audio and visual inputs. The visual input comprises of tracking the user's eye gaze on the screen. Our algorithms must identify patterns where the user wants to turn the page by staring at the end of the page and/or head gestures. The audio model takes in the user's audio input to track the user's current position in the music, while also being aligned with the uploaded MIDI File. The system must be robust enough to handle the occasional wrong notes and uneven rhythms. Both sensors should fail to detect inputs less than 1% of their active time.

2.2 Frontend Use Case

The system must indicate to the user where they are in the score with a real time cursor that is constantly updating. The display should be easy and intuitive to use. After that, there needs to be a consent page to alert users that the system is tracking their eyes and recording their audio. The system must display page turns and turn the page accurately. Page flipping is the most important requirement; we're aiming for a page flip success rate of 95%.

This requirement is rooted in the principle of accessibility for social impact. The frontend is what a user directly interacts with, so it should be as easy to use as possible to encompass a wide range of people.

By prioritizing these features, SoundSync aims to create a user friendly, adaptable, and seamless platform for musicians while promoting accessibility.

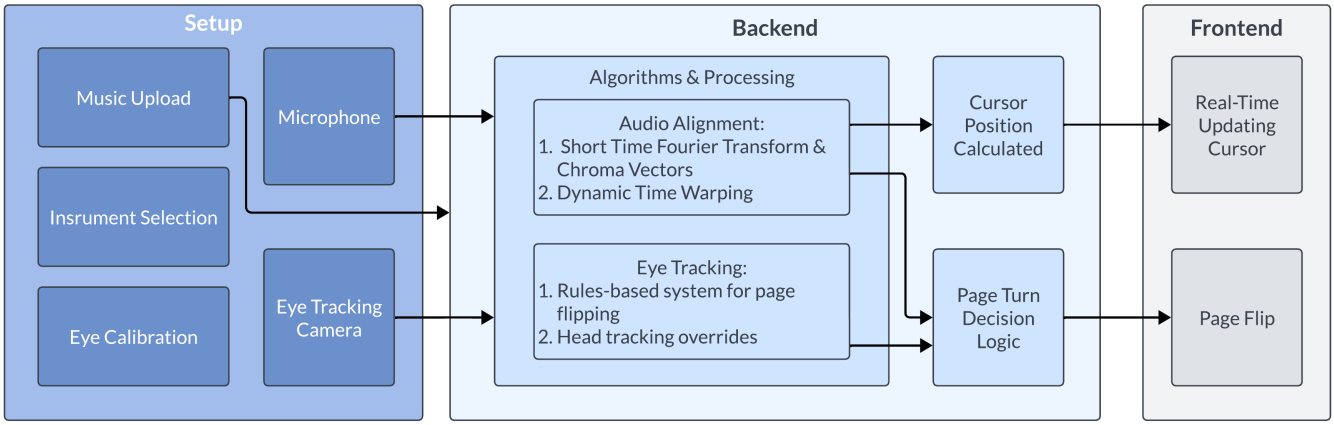

Figure 1: Block diagram of frontend, backend, and peripheral components interacting. The setup stage shows what happens before the user can begin playing music. The backend section notes the algorithms used in the backend and how they translate into outputs. All the backend processing occurs with data collected from the microphone and camera peripherals. Finally, the frontend demonstrates key features of the completed application and how they respond to real time user inputs.

3 ARCHITECTURE AND PRINCIPLE OF OPERATION

SoundSync's physical structure consists of a Tobii Eye Tracker 5 camera and microphone connected to a Windows laptop acting as a display from which to read digital music. A microphone sits on or close to the user's instrument to continuously record sound and plug into the laptop via the headphone jack. The eye tracking camera connects to the laptop through a USB-A port. The changes to our system since the Design Review are as follows. We are no longer hosting the backend on a Google Board, we are not training an ML model to implement eye tracking, the frontend is hosted using Django and JavaScript instead of Python, we're not implementing frequency filtering, and the audio alignment isn't solely relying on Dynamic Time Warping.

On a high level, the Windows laptop is connected to the Tobii Eye Tracker 5, microphone, and runs the program that displays the frontend. The system incorporates two models: a visual eye tracking model, and an audio alignment model. The visual model takes in the filtered eye tracking data and determines from a visual perspective, whether or not to turn the page. The audio alignment takes the processed audio signal and tries to align the stream with an uploaded MIDI file. Eye tracking uses a heuristic rules-based approach and linear extrapolation to classify or predict where the user is gazing. These actions are then sent to the frontend, where the page will either be flipped or not.

Refer to Figure 1 for a block diagram of the system architecture. Figures 2 and 3 expand upon the frontend and backend subsystems.

4 DESIGN REQUIREMENTS

4.1 Eye Tracking Requirements

Our specifications revolve around keeping the eye tracking component accurate and precise. To address the visual sensor input use case requirement, the system's margin of error is low and allows the system to outperform current solutions.

The first requirement is that the eye tracking field of view must register a human face. The Tobii Eye Tracker 5 meets this requirement with its 40 x 40 degrees field of view.

The second eye tracking requirement is the accuracy of the eye tracking data points. These points must be accurate and precise to one bar. Therefore, a 13" laptop display requires an accuracy of 1.65 cm to ensure we are within a single bar. A precision of 1.0 cm, which corresponds to the height of potential notes in a bar that could extend above or beyond the staff lines, keeps the eye tracking within the correct line on the sheet.

These two requirements warrant that the system accurately and precisely follows the player's gaze. Using a rules-based approach allows us to hard-code override identifiers on the page. We also need off-the-page gaze handling to identify when a user looks away from the page. In this situation, the system should not crash, and the algorithm should continue to predict the user's position. The purpose of implementing eye tracking in addition to audio alignment is to make the model more robust in cases where audio alone may not suffice such as nonlinear musical structures that go back in time or jump to different pages.

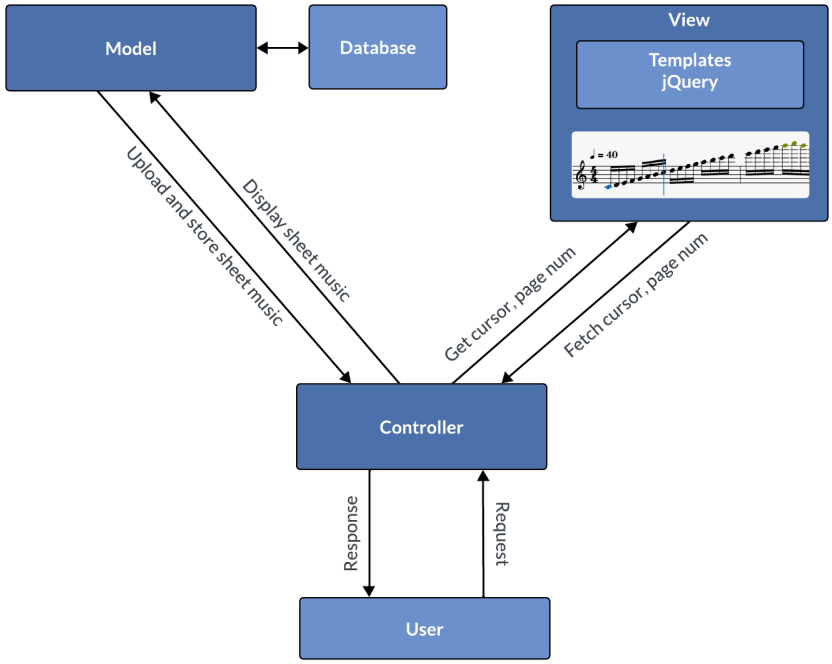

Figure 2: Frontend subsystem diagram. The frontend is structured using the Django Model View Controller Architecture. At the beginning, the user is directed to an "Upload Music" screen where they will upload both their sheet music and a MIDI file of the music. These files are stored in the database using Django models. After that, the user will begin calibrating their eyes. By focusing on each corner of a page of sheet music, the camera will map the location of the eyes to a location on the screen. In views, the processed backend data is updating the template using jQuery. Once these setup steps are complete, sheet music will be displayed, and the user can begin playing music. There must be a moving real time cursor and short page flip animations to simulate turning a real page while giving the user time to adjust to the new sheet.

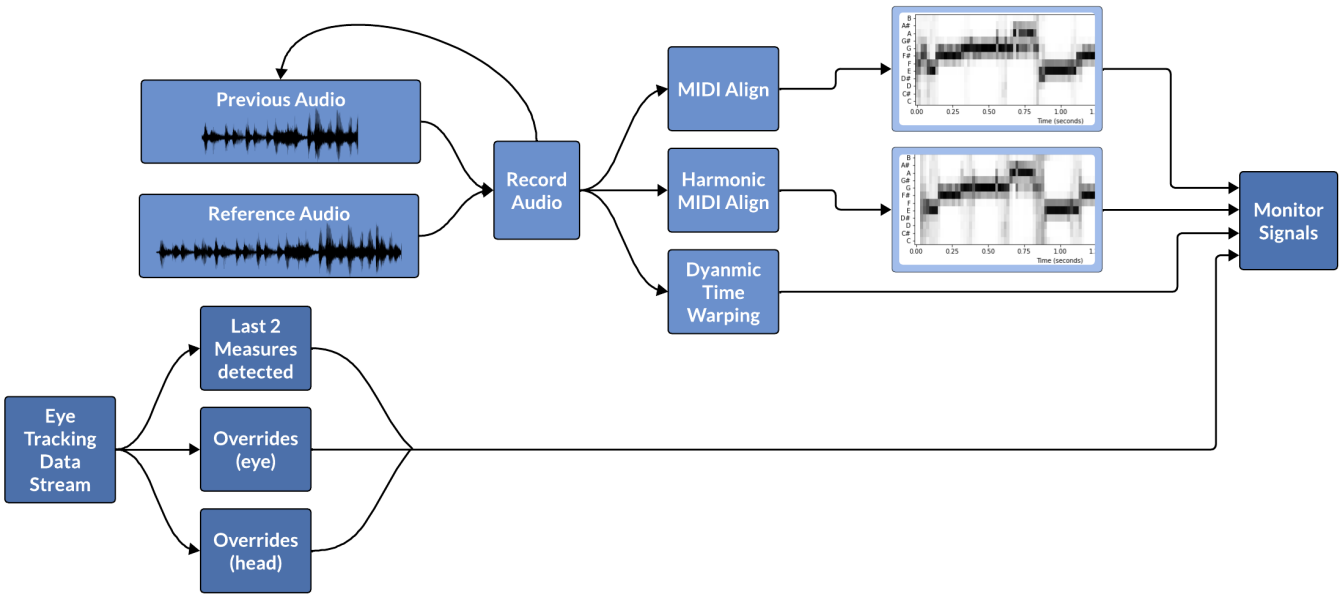

Figure 3: Backend subsystem diagram. A Windows laptop runs both audio and visual algorithms. On the audio end, an uploaded MIDI file is segmented into finite time sequences and is examined to extract information on tempo throughout the piece. The segmented parts of the MIDI file are aligned with the live audio using a two pointer matching algorithm. On the Visual end, data from an eye tracking camera is filtered and run through a logistic regression model to determine whether to flip the page.

4.2 Audio Requirements

In order to address use case requirements for sensor inputs and frontend, the backend must ensure accurate page turning. Specifically, we're looking at the time distance between where the music is and where the model thinks it is. The accuracy requirement is up to 1 beat in either direction, forward or backward. This requirement ensures that our audio accuracy is close enough to avoid large audio alignment and syncing issues.

To address the frontend use case requirement, segmentation is be deployed to reduce latency. Segmentation involves dividing an audio sequence into smaller pieces in order to process them individually. This technique is essential to guarantee real time page flipping.

To address the sensor input use case, the audio model must be robust enough to operate accurately despite 1 wrong beat per sequence of 10 beats. The user must perform 90% of the notes accurately within a segment of music.

Another requirement designed to address the frontend use case requirement is the SNR. The audio component ensures the SNR is greater than 25dB.

4.3 Hardware Requirements

All hardware used for the design must be simple, user-friendly, and intuitive to operate. First, the display device should last for the entirety of a long practice session; four hours should account for most cases. Therefore, a Windows laptop satisfies this requirement. Furthermore, the whole system must be portable and consist of small components: a lapel microphone, eye tracker, and a laptop. This is to ensure that the system can be setup and used anywhere with ease.

4.4 Latency Requirements

Processing delay for both these models must be short and accelerated to avoid unnecessary lags within the system. SoundSync's processing latency should be no longer than 500ms. 500ms refers to a quarter note at 120 BPM, which was the proposed fastest tempo that our system must work with. These design requirements address the frontend use case requirement of having a high speed processing system.

5 DESIGN TRADE STUDIES

5.1 Backend Compute

The Google Board was our board of choice at first because it had the best processor for machine learning. The edge TPU processor can run ML inferences quickly and efficiently. However, using a Google Board would also necessitate an additional battery pack to supply power.

During development, the Google Board OS ran Debian Linux which was incompatible with the Python packages PvRecorder, SyncToolbox, and Librosa which are integral for audio recording and aligning. Without supporting these packages, we could not reasonably use the Google Board for the audio backend. Furthermore, the eye tracking calibration steps need a display for setup - and the Google Board didn't support displaying calibration. With the fatal flaws detailed above, we pivoted to using a Windows laptop for the backend.

With the Windows laptop, we were able to integrate all the Python packages we needed for audio alignment. The eye tracker is not compatible with MacOS, which is why we opted for Windows. Directly connecting the frontend and backend was simplified when using a laptop for both systems.

Moreover, Django is compatible with Python subprocess calls, which means accessing the eye tracking data stream was more seamless when using a Windows laptop, since all of the programs were hosted on one machine. Additionally, we found that the Tobii SDK and API were not compatible with the Google Board's OS, Debian Linux, which meant that we could not run our eye-tracking program off the Google Board.

The Google Board also had difficulties displaying matplot graphs. Therefore, testing audio alignment off the Google Board proved to be inefficient. Lastly, having all of the code files on one machine decreased our system latency significantly because we lose the costly device to device latency if we used the Google Board and a laptop working together.

Ultimately, the decision to use the laptop for backend compute was a careful balance of performance, integration efforts, and compatibility with the frontend, ensuring optimal system functionality.

5.2 Camera

The Tobii Eye Tracker 5 camera is the foundation of retrieving high quality eye tracking data. This camera, designed for eye tracking, can retrieve pupil data within 0.5 degrees compared to 2-5 degrees for a conventional webcam[2]. The camera also has built-in head tracking which when used with the existing model can help account for non-linearity in the music by giving the user a manual override method. Furthermore, because pro models of Tobii Eye Trackers are outside of our budget, we developed using the more accessible Tobii Eye Tracker 5. This camera required development through the Steam Engine API on Windows OS. The Tobii Eye Tracker 5 SDK had head tracking capabilities, but we found that this feature was too sensitive and could not be used for a musician as they tend to move while playing an instrument.

Although conventional webcams can be programmed to have high quality eye tracking, adding a peripheral would result in a less compact design. Furthermore, Microsoft tested the accuracy and precision of the predecessor to the Tobii Eye Tracker 5 under different lighting environments. Because these cameras are similar, the baseline data extracted from Microsoft is likely comparable to the Tobii Eye Tracker 5 camera under artificial and natural lighting

conditions. We tested the camera's accuracy and precision by tracking the user's eyes as they played through a sheet of music.

5.3 Microphone

A lapel microphone works very well at picking up sounds at a short distance but doesn't pickup sounds from farther away. Although boom microphones and podcast microphones can lead to slightly more clear signals, the size and/or price of these microphones make them incompatible with our project. The lapel microphone also has the ability to be clipped onto either the user's collar or the instrument itself to get a clear harmonic sound from the instrument. On top of this, this high quality microphone had a signal-to-noise ratio far exceeding our use-case requirements. This was measured by inputting a pure sine wave and measuring the noise in the output using Audacity. The lapel microphone worked well for a violin, but other instruments may require slightly different types of microphones for audio recording. For instance, Dr. Roger Dannenberg placed a transducer on the mouthpiece of a brass instrument, since that is the region of the instrument where the most pure frequencies can be recorded from.

5.4 Frontend Choice

We contemplated using several options for the display: a React webpage, a locally run Python program, an iPad app, and a webpage with a Django backend.

Developing an iPad app proved to be difficult as nobody on the team had experience with Swift development. Integration with an iOS app also requires adhering to iOS SDK guidelines and managing resources to prevent the app from slowing down and failing to be real time.

While our frontend developer had experience with React web applications, several challenges prevented us from ultimately choosing React as our frontend framework. Because React is a client-side framework, it may be difficult to handle the continuous stream of data from the peripherals. Furthermore, integrating specialized hardware like the Tobii Eye Tracker 5 and Google Board would have required careful handling of low level interactions and device specific protocols. We additionally looked into implementing our frontend using Tkinter, but that proved to have less intuitive UI than a standard webpage with limited functionality. However, we definitively wanted to use Python for our frontend implementation, since Python allows for easier integration with peripherals. Python also has an extended set of libraries that are compatible with MIDI files, the Tobii Eye Tracker 5, and audio in general.

Ultimately, we used a webpage with Django in the backend to run our app. The Model-View-Controller architecture helped manage interactions across the frontend and backend. The pages were handled by templates and the urls and views helped manage and connect backend functionality and stream it to the frontend. We tried multiple approaches to send data to the frontend in real-time with the lowest latency possible. First, the web sockets approach failed to provide the kind of interaction we were hoping for. Eventually, we used jQuery and AJAX to send data from the backend to the frontend continuously every 50ms. We found that the Python views and controls in JavaScript helped us maximize functionality and focus more on connecting and integrating rather than building a usable framework for a frontend application.

5.5 Displays

The current design for the system has a 13" laptop screen to act as the display placed on top of a musical stand. Although laptops are more bulky than a tablet, 73% of Americans own a laptop which increases accessibility[4]. Furthermore, using a laptop as a display is much simpler as an iPad would require a developed app, while the laptop can simply display information from the backend.

5.6 Override Conditions

Override conditions were necessary to satisfy accurate page turning. With override conditions, users can communicate directly with the system and manually turn a page. If a user uses override conditions they can mitigate any risk associated with using the system's autonomous components. For example, if audio alignment turns the page too early or too late, users can use the eye tracking override to quickly turn the page where they need. In case of failure, there is still a way to mitigate the risk of not having the page turn at all. This approach provides the same functionality of existing apps today.

5.7 Rehearsal vs Performance vs Practice

The scope of whether the user is performing or practicing drastically changes the design of the system. Designing for performance requires the system to have extremely robust page turning with the ability to distinguish and separate polyphonic music. This is because a wrong or inappropriately timed page turn may cause the user to lose focus and can lead to mistakes that jeopardize the quality of the performance. Musicians are also playing in environments with several other musicians, so the scope of our project is a private rehearsal. This allows for a lower successful page turning rate as the stakes are not as high.

6 SYSTEM IMPLEMENTATION

6.1 Audio Alignment

Our first iteration of audio alignment used Dynamic Time Warping to align a current snippet of audio to the reference MIDI. DTW is an algorithm aimed at minimizing the Euclidean distance between two finite time sequences. Although DTW is robust, it imposes a set of conditions that are assumed to be met. The first is that DTW assumes finite time sequences are passed in with a start and

end point. This means DTW must be fed in a complete segment of recording and cannot be fed more data while computing. The second condition for DTW is that the audio is assumed to always be progressing forward, called the monotonicity requirement. This requirement of always moving forward in the music means DTW does not work for an audio segment where the user replays a section. This limits it's capabilities for a system that aims to align in a practice setting where the user may jump to anywhere in the music at any time. To run DTW, the segment of audio recorded was aligned to the next four bars from where the cursor position was. Looking at the next couple bars was added to reduce the latency of DTW. DTW was ultimately not enough to align audio on it's own and hence we worked on a seperate MIDI Align algorithm.

In this algorithm, we extracted the note sequence from the time sequence by taking a Short Time Fourier Transform and the chroma vector. Chroma vectors reveal what note bins a given frequency is sorted into and helps us extract notes from frequencies with confidence. This bin method means even if the user is slightly out of tune, because the bins encapsulate the note as well as the frequencies above and below it. After filtering the chroma vector data, we get a sequence of notes and their relative positions. After converting both the reference MIDI and the current audio into these arrays of notes and "positions", we began building a matching algorithm to rapidly place sequences of notes into the larger reference. We used a two pointer approach in order to traverse the list while adding the least amount of additional processing time.

As we built upon our design, we encountered several unforeseen challenges. A point of difficulty in audio alignment was dealing with latency at various steps in the system. For example, librosa is a Python library that processes the audio into audio frames and also computes chroma vectors. However, this function runs caching in the background on the first call that causes the delay to rise from 20ms to 900ms. Therefore, our first call to audio alignment lagged the system and lead to undefined behavior. We fixed this by calling librosa during setup.

Another latency bottleneck ended up being the constant AJAX calls to update the webpage every 20ms. Eventually, the system would lag, so we decreased the frequency of these calls to once every 50ms. This ensured that there was time for the backend to process while still keeping the frontend cursor moving smoothly. One of the biggest accuracy hurdles was overcome when we realized that running chroma vectors on raw recorded audio gave an output with several wrong notes. Upon inspection, these additional harmonic frequencies were being detected due to the resonance of the violin. We fixed this issue by using a mute to dampen the sound of the violin, which mitigates the effects of violin resonance for now.

6.2 Eye Tracking & Head Tracking

The Tobii Eye Tracker 5 came with the ability to track eye movement and head position. The major benefit of the Tobii camera was the high precision and high data rate. However, because the Tobii Eye Tracker 5 is not categorized as a pro model, it is incompatible with Tobii's Python SDK. This meant in order to use the camera, the code had to be run in C++ - the only language with SDK support. Therefore, we worked hard to correctly transfer retrieved data points from C++ to Python. We used a Python subprocess call to the C++ executable created by Visual Studio. This approach wrote the C++ subprocess to the stdout pipe and had Python read out the buffer to gather eye data. However, the readline call to read the buffer from the stdout pipe had a latency whenever the buffer was empty. Essentially, if the system attempted to retrieve eye data while the buffer was empty, the whole system had to wait until data was written to the buffer. The recurring cost of this operation was 30s. Fortunately, the delay did not significantly reduce the performance of the system.

Head tracking was another feature intended to be added to the system as a fail safe for musicians to turn the page in the case of a system failure. However, because musicians are rarely still when playing and could even turn their head to look at different parts of the room, we found this method extremely error prone. The head tracking tests we performed proved that it was too sensitive to movement. We instead discovered that having the user look at the turn page forward or turn page backward buttons proved much more reliable as it was unaffected by the user's movements.

6.3 Frontend

We used a locally hosted interactive webpage with a Django backend to display the sheet music. Django is a Python based web framework that follows the Model View Controller Architecture. The web application has 4 pages: start, calibration, music upload, and display. The start page contains a description of the system alongside the Terms and Conditions. Here we described how we will not be storing or selling recorded eye and audio data. The next page is the calibration page which details how to install and use Tobii Experience which is the application needed to calibrate the camera to a user's eyes.

The consecutive page is the Upload page. Here, the user uploads their standardized sheet music, as a PDF, and its corresponding MIDI file. Lastly, the user picks which instrument they are playing and must click submit. On submit, the last page is rendered with the uploaded sheet music and a cursor at the beginning of the piece. We included two override buttons above the sheet music to allow users to manually turn the page forwards or backwards using a mouse click or eye tracking. At the bottom of this page, there are buttons to toggle eye tracking and audio alignment, so the user can use one subsystem or both at the same time. This feature was implemented for testing purposes.

During development of the webpage, we struggled with getting data to display in real time without lagging the entire system. Initially, we attempted to use web sockets and Django channels to implement the cursor moving

as a function of the backend data, however the implementation of web sockets was not what we were looking for. After a lot of trial and error, we achieved full functionality in the frontend using Asynchronous JavaScript and XML (AJAX) to seamlessly interface with the backend and receive updates for the cursor. Using AJAX marked a pivotal breakthrough, because alternatives like web sockets proved to be incompatible with our code base.

7 TEST & VALIDATION

We conducted a series of tests to verify and validate our system. Please refer to Figure 7 for a table of the results.

7.1 Audio Tests

The first audio test was a signal integrity test that was performed using a pure 440 Hz audio input. Instruments such as violins have resonances at harmonics associated with a western tuning system. Therefore we fed a non-resonant pure tone into the microphone for this test. Once filtered and processed, the SNR of the signal picked up by the microphone was greater than 25 dB.

The second test was page flipping at multiple tempos with varying musical structures. Custom composed pieces encompassed a range of musical beats, structures, and notes. The beats covered note lengths ranging from an eighth note to a whole note and notes ranging from G3 to E6. The following tempos were tested: 60 BPM, 90 BPM, and 120 BPM. This range accommodated most beginner repertoire. For each of these variations, we tested whether the page flipped within the last 2 measures of each page.

Another test that was run was the system's robustness as the environment changed. Due to the scope of our project, we wanted to make sure our system worked in noisy rooms with background noise as well as quiet practice rooms. The system should meet all use case requirements under conditions where noise is voluntarily added by the user, such as via a metronome. Therefore, testing of the system was completed in a quiet environment as well as a loud environment with a metronome running.

For unexpected user edge cases such as stopping prematurely or repeating sections randomly, the system operated normally without any unexpected behaviors. Specifically, the user stopping prematurely caused the cursor to stop until the playing resumed.

7.2 Eye Tracking Tests

Eye tracking tests focused on verifying that the distribution of data points obtained by the Tobii Eye Tracker 5 stayed within one bar. The predicted size of this bar is 1.65 cm x 1.0 cm for a 13" display. If the data point distribution did not meet this requirement, then the size of the bar had to be adjusted.

To test this, Sanjana read music across one line at three different tempos, while recording her eye positions. The three tempos we used were 60, 90, and 120 BPM. We then plotted this distribution in Figure 6.

When standardizing the sheet music, we organized the lines such that there would be sufficient space where eyes wouldn't overlap. To test this, we recorded the range of the y-coordinates of Rohan's eye position as he read different lines. We looked at the variation in points and checked for no overlapping sections. The results of this test were plotted in Figure 5.

7.3 Integration Tests

Integration testing aimed to measure and verify the improvement that eye tracking adds to the existing system. We tested our system with multiple violinists, and had them test the system twice: once with only audio and once with both audio and eye tracking. The violinists gave us feedback of our system and told us what they preferred. All the users said that the audio alignment alone performed well, but in cases of wrong or missed page turns, the user had to manually click the next page or back page button. When both systems were used, all users said the experience was smoother. In rare cases of wrong/missed page turns here, the visual eye-override quickly helped solve the problem.

7.4 User and Frontend Tests

We tested the system on violin sheet music across 20 page flips. Our design goal was to have a page turning accuracy of 95%. What we measured was a page flipping accuracy of 90%, as we observed 2 wrong and/or missed page flips out of 20 during our tests. However, the override controls worked 100% of the time, when a wrong occurrence of a page flip did occur. Therefore, the eye tracking system helped bring our total system page flipping accuracy to 100%.

Lastly, we asked our users to give feedback about our front-end, specifically if it was intuitive and user-friendly. They all reported that the web-page was very easy to use and understand. They also said that the overall experience was smooth and very user-friendly.

7.5 Results for Design Specification: Eye Tracking

Eye Tracking Latency: When the eye tracking enable button first gets toggled, the system starts running the eye tracking heuristic model. The eye tracking heuristic model takes 250 ms to create the API and initialize the camera. The Tobii Eye Tracker 5 Camera operates at a sampling rate of 60 Hz, which means that each data sample of the user's eye position takes 16.67 ms to record. However, sending the data to the backend of the web application and then reading each sample from the backend takes 30 ms. Therefore, when the system is running, the latency is 30 ms.

The reason why the eye tracking program is not used as a helper function is because the initial startup phase

of the program costs 250 ms each time the function is being called. This is due to the function needing to create a object that finds and links itself to the eye tracker. As a result, a subprocess of the program was initialized to run the function once and continue running. This causes the 250 ms cost to occur once at the beginning of our system and continue with an average latency of 30 ms while the user is playing.

Figure 5 shows the plot of the range of y-values for each row in our standardized sheet music. Notice how the data points do not over lap for each row, as each row is spaced evenly from each other. This shows that our eye tracking is very reliable when identifying which specific row and bar the user is currently looking at. In Figure 5, however, we did observe some overlap between row 2 and row 3. Nonetheless, this is negligible because our eye tracking heuristic pays close attention when the user is looking at the last row, right before the page turn. Lastly, we found that the distribution of points was accurate and precise within 1 cm, which satisfies our dimensional design requirement.

Figure 6 shows Sanjana's eye distribution as she read one line of music at various tempos. We observed that the distribution of the user's eyes does not vary as a function of the tempo. This reinforced that our eye tracking data is very consistent across multiple tempos when working with a bar height of 1.0cm and a width of 1.65cm.

7.6 Results for Design Specification: Audio Alignment

We conducted a series of tests for Audio Alignment. For page flipping using audio alignment only, we tested 20 page flip boundaries at 120 BPM and found that the page turned within 2 bars of the end of the page 90% of the time.

For the latency of the audio alignment subsystem, we conducted a series of tests in the backend at several different tempos. We found that the latency across tempos ranging from 60 BPM - 240 BPM didn't exceed 100 ms. We also found that the introduction of a metronome didn't impact the computation time in the backend. This reinforced that the audio alignment algorithm we wrote didn't introduce a large slow down. We changed our audio alignment approach from using DTW to our algorithm in order to achieve this latency.

Our segmentation tests revealed that an average length of 50 frames at a frame length of 1024 with a sampling rate of 16000 Hz was ideal for the duration of a segment. We found this by testing a variety of lengths of frames according to the average length of a beat in a given piece. We found a good balance between the accuracy of the alignment and the duration of recording, when using 50 frames of audio recording.

7.7 Results for Design Specification: Hardware

To test if the hardware used was intuitive to operate, we had multiple violinists test our webpage. We provided the sheet music and corresponding MIDI files, however, the test users walked through the webpage themselves. Because the system is hosted on a laptop, the system can run for 4 hours, which satisfies another hardware requirement. The overall system was also fairly portable and consisted of a laptop, microphone, and small microphone.

8 PROJECT MANAGEMENT

8.1 Schedule

Refer to Figure 4 in the Appendix.

8.2 Team Member Responsibilities

The components for this project fall under 3 categories: frontend, visual, and audio. The audio tasks include testing the best placement for the microphone, ensuring the microphone signal is clear, processing the audio in real-time continuously, and writing an algorithm to align it. The visual tasks consist of writing the code that interfaces with the Tobii Eye Tracker 5, filtering the data points, and implementing a heuristic to determine how to evaluate eye data. The frontend tasks comprise of building a user interface that shows the eye tracking calibration, allows the user to upload their sheet music and a MIDI file, and displays the sheet music with a following cursor as well as page turns.

The tasks were divided with Sanjana tackling the frontend development and audio alignment, Rohan working on the hardware and integration with decision logic algorithms for eye tracking and audio as well as frontend debugging, and Caleb solving audio alignment problems and helping with eye tracking. We all worked on at least 2 subsystems and integration was done together.

8.3 Bill of Materials and Budget

Refer to Table 1 in the Appendix.

8.4 Risk Management

The major risks in this project ended up being integration, latency, and audio alignment.

Integrating multiple subsystems that all run in different environments meant that we needed to begin integration efforts much earlier than we did. Despite audio alignment being done in Python, once integrated, several days were spent debugging the algorithms and connecting to the frontend. Once we added the eye tracking C++ subprocess call into the mix, it became even more difficult to ensure the algorithms were collaborating with each other. Tens of packages and libraries as well as SDKs all needed to run in parallel in a Django backend. There were a plethora of issues with the Django backend and parallelization.

Throughout development, we struggled with the latency vs accuracy trade-off. Latency is extremely important to

us as the system needs to be real time and the cursor position should be updating and aligning with the most recent data as frequently as possible. With recording audio, while we could process about 50 frames in a matter of 20ms, the backend had to spend roughly 3 seconds to only record audio. This meant while collecting the 3 seconds of audio, the system has to assume the user is playing in time. This delay in audio alignment can hinder the user's practice experience. However, if we reduced the amount of frames recorded, and hence the recording time, the number of notes heard would be reduced and it would be harder to align. Our current audio alignment algorithm can accurately align audio in sub 500ms by assuming the user is always playing a note within the next couple bars and recording for only about 50ms. This means that within that time frame only one frequency is heard and is aligned to a nearby note. Our system allows for multi-page jumps which means one detected frequency is not enough to correctly identify where on the page the user is playing. Hence, we found 3 seconds to be roughly the amount of time needed for a recording to align well. It is also important to note that this time was a function of the average length of a note and would differ piece by piece. For faster pieces, where more notes are played in a shorter duration of time, the recording time could be as low as 1.5 seconds. For slower pieces, where more time is need to read multiple notes, the recording time could be as high as 4.5 seconds. This ultimately gave the system the ability to adapt to different pieces and prevent undefined behaviors across pieces.

Another major risk in our project was if we could write an audio alignment algorithm that aligned audio in real time. While this may sound feasible, audio alignment in real-time is an ongoing research area without quantitative results, especially when accounting for nonlinear musical structures that go backwards or jump forwards in time. Furthermore, we initially relied on Dynamic Time Warping to do the heavy lifting for us - locating similar sequences and aligning them in time. Due to the algorithms being run under the hood, DTW took too long to return a location. Due to this, we began working on inventing our audio alignment algorithm. We settled on a two pointer approach post chroma vector analysis to determine recorded sequences and align them with the corresponding reference audio. As we tested this algorithm, more and more edge cases were discovered, and we wanted to ensure that our algorithm would allow users to play music non-linearly. Therefore, we were not able to use the "arrow of time", which is the belief that time only moves forward.

9 ETHICAL ISSUES

The worst-case scenario for SoundSync is that it gets breached and user privacy is compromised in some manner. An attacker could exploit a vulnerability in the system, breaching SoundSync's security protocols. They would be able to gain unauthorized access to the user's data, capturing sensitive information such as sheet music, practice patterns, and possibly personal details. Our intended use case is that a user has the technology sync to their performance in real time and flip the page automatically. Walking through a less-than-ideal scenario, we begin with a user practicing, however their connection gets intercepted. While they were trying to practice, someone else captures their data and essentially hacks them. In this situation, the user's privacy is compromised. A musician's privacy and potentially a musician's intellectual property in the case where a musician is the composer.

In particular, people with visual disabilities or other impediments that impact their music performance may be more negatively impacted by a security breach. Individuals with visual disabilities may also create and modify sheet music or audio recordings for practice, which is their own intellectual property. Their sheet music or audio data could get stolen, which is more detrimental. In this scenario, the user's privacy is violated. Although this data may not seem like it gives away much personal information, companies could, in theory, use the eye tracking data to infer health data about the user. This could then be used for advertising and help companies create a profile of the user. Keeping this medical information confidential and safe is vital to prevent companies from collecting and generating complete profiles of people.

Systems that follow a player and flip the page automatically were first built by Dr. Roger Dannenberg at Carnegie Mellon University. These systems then went on to become a system known today as SmartMusic. Other people have built similar systems such as Andreas Arzt who showed that DTW could be run on segmented sequences of the MIDI file and live audio for real time audio processing and score following[2].

SoundSync differentiates itself from existing technologies by studying how eye tracking can be combined with audio alignment to provide extended functionality for users. Our system will also perform audio processing in real time, whereas many existing technologies perform post processing on completed audio streams to determine where a user is located in the music.

10 SUMMARY

Soundsync takes in visual and audio inputs to determine when to digitally flip a sheet of music. The system used the Tobii Eye Tracker 5 camera, a Lapel clip-on microphone, and a Windows laptop. Our system met most of the design specifications. Most notably, the page flipping accuracy was 90%, however, the combined accuracy of the system with both audio alignment and eye tracking was 100%. We are very pleased with the latency of the backend subsystem. The latency's of the audio alignment was roughly 100ms, the eye tracking latency was about 30ms, and the latency of the front end was about 50ms.

Some limitations of our system include overly repetitive music, musicians who are playing out of tune, and tempo. Due to the nature of our audio alignment algorithm, there

is low tolerance for blatantly wrong notes.

With more time, we would have liked to train an ML model for eye tracking to better predict the confidence levels of specific edge case eye movements such as looking up off the screen or glancing down. These specific glances are tied to when the user looks at the conductor or their instrument and would give extra information to work with. Additionally, we would have liked our audio alignment algorithm to include more support for edge cases and repetitive sections of music.

SoundSync is designed to be accessible to those who cannot operate existing page turning machines and for those who are looking for a seamless and non-distracting music experience.

10.1 Future work

The future of SoundSync involves better accounting for some edge cases and testing our product with a wider range of music with more nonlinear structures. We are excited to keep working on the product during our free time over winter break!

10.2 Lessons Learned

Our project was extremely ambitious and attempted to solve a novel problem that is an ongoing research area. Overall, integration is challenging. Our main takeaways are to integrate components earlier in development. Another takeaway is that work should constantly be saved. We ran into an issue where a team member's laptop got wiped. Thankfully, through the use of git and all of us being well-versed with the other subsystems, we worked together to resolve and rewrite any code we lost.

Glossary of Acronyms

- API - Application Programming Interface
- BPM - Beats Per Minute
- DTW – Dynamic Time Warping
- MIDI - Musical Instrument Digital Interface
- ML – Machine Learning
- OS - Operating System
- SDK - Software Development Kit
- SNR - Signal to Noise Ratio
- TOPS - Terra Operations Per Second

References

[1] Andreas Arzt, Gerhard Widmer, and Simon Dixon. "Automatic Page Turning for Musicians via Real-Time Machine Listening". In: Jan. 2008, pp. 241–245. DOI: 10.3233/978-1-58603-891-5-241.

[2] Anna Bánki et al. "Comparing online webcam- and laboratory-based eye-tracking for the assessment of infants' audio-visual synchrony perception". In: *Front. Psychol.* (2021).

[3] Google. 2023. URL: https://coral.ai/products/dev-board/#description.

[4] Pew Research Center. *The Demographics Of Device Ownership*. Tech. rep. 2022. URL: https://www.pewresearch.org/internet/2015/10/29/the-demographics-of-device-ownership/.

[5] Romain Tavenard. "Introduction to Dynamic Time Warping". In: (2021).

11 Appendix

Table 1: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Tobii Eye Tracker 5	N/A	Tobii	1	$298.53	$298.53
Shure MVL Microphone	N/A	Amazon	1	$73.83	$73.83
Microphone Adapter	N/A	UGREEN	1	$7.00	$7.00
Google Coral Dev Board	N/A	Coral	1	$144.24	$144.24
Total					$516.60

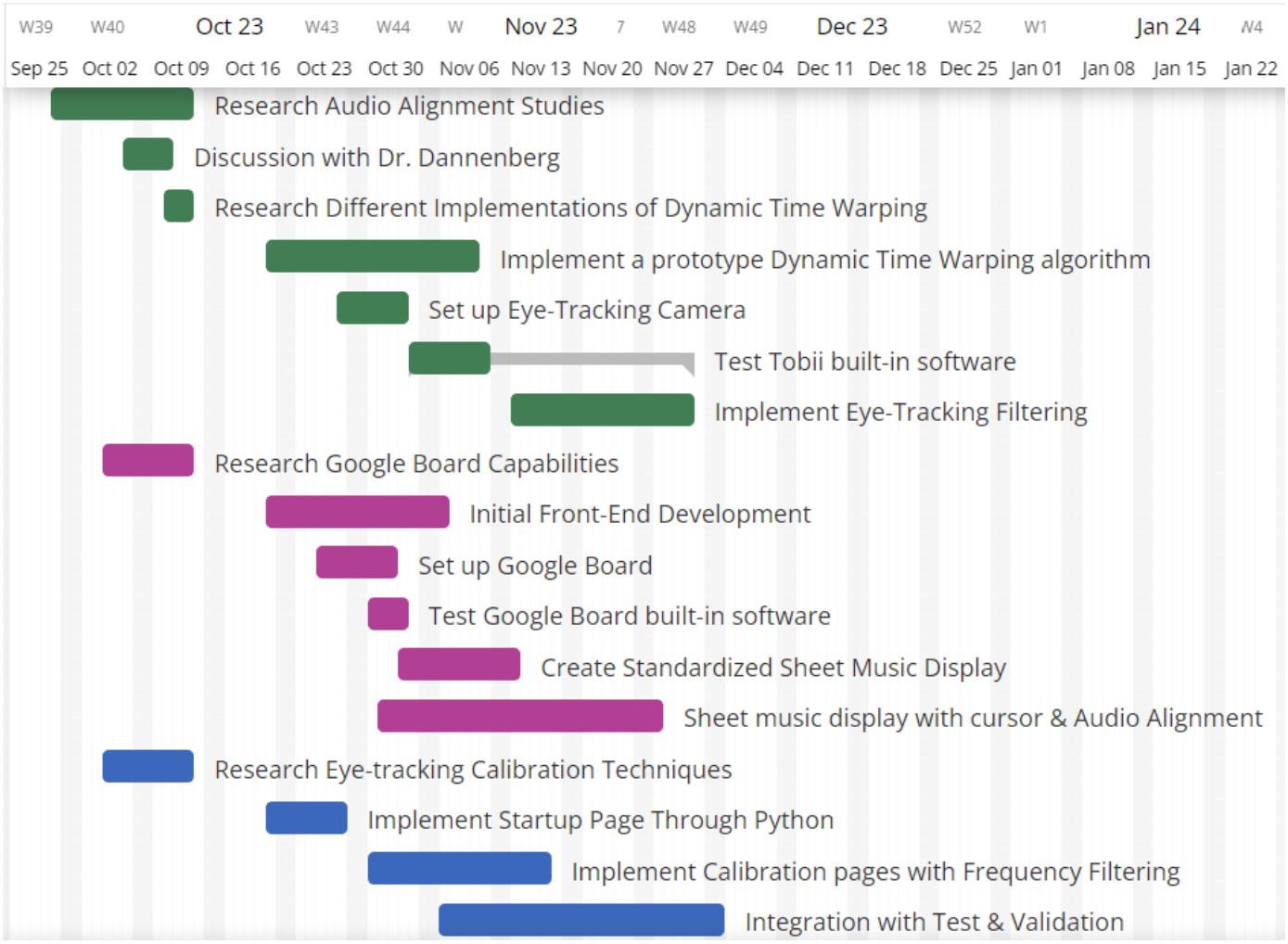

Figure 4: Gantt Chart Diagram. The green tasks (Caleb) are audio, the pink tasks (Rohan) are hardware, and the blue tasks (Sanjana) are display and eye tracking. The division of labor is divided to optimize parallel development and provide slack for integration of different components. The tasks were divided based on areas of expertise or interest and accounted for breaks and holidays.

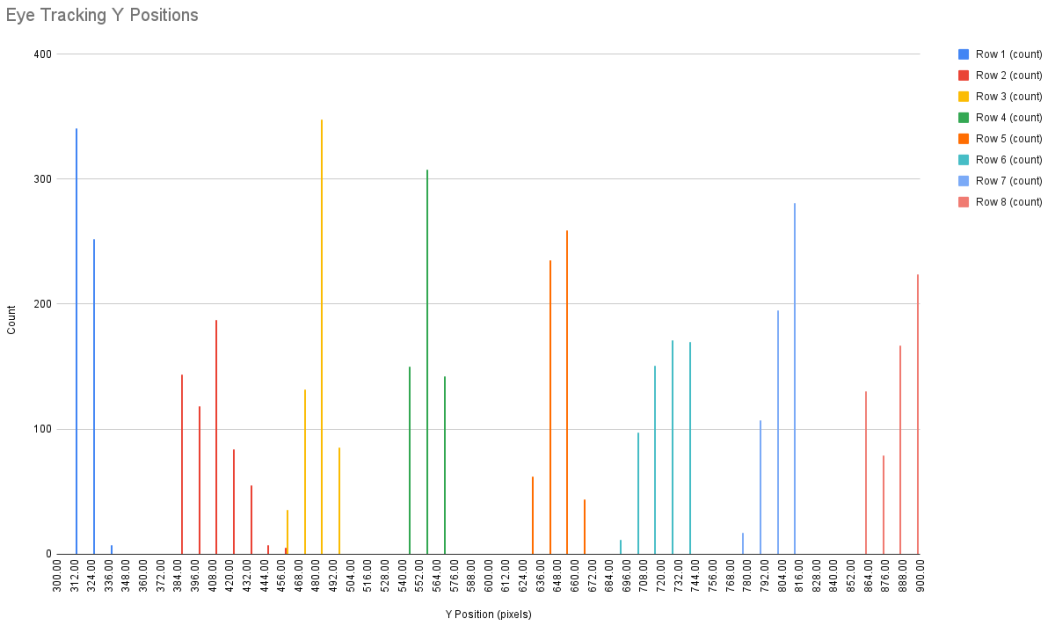

Figure 5: The measured coordinates of the Y coordinate of a user's gaze as they played through all consecutive lines on a page.

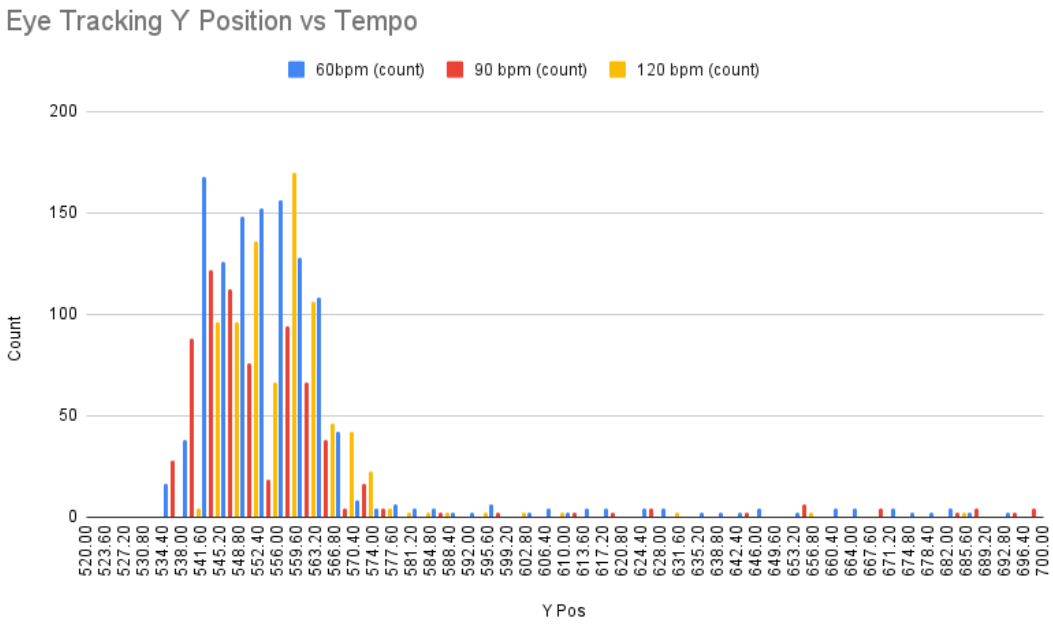

Figure 6: The measured Y coordinate of a user's gaze as they played through one line at various tempos.

Metric	Test	Target	Actual	Met
Backend (Eye) Latency	Measure how often the front end reads from the backend in order to update information	< 500 ms	30 ms	✔
Page Flipping Accuracy	Play through music and determine if page flips occur within 2 bars of the end of the page	> 95%	90%	✘
Visual Model Improvement	Measure how much page flipping accuracy improves with eye tracking and head tracking	> 5%	10%	✔
Eye Tracking Accuracy	Measure variation in data points	< 1.65 cm	< 1 cm	✔
Eye Tracking Precision	Measure variation in data points	< 1 cm	< 1 cm	✔
Backend (Audio) Latency	Play portions of two octave scale and ensure the audio aligns	< 500 ms	100 ms	✔
Audio Alignment	Measure if cursor is within 1 bar of current position in music	1 Bar	> 1 Bar	✗
Audio Robustness	Track alignment for intermediate missed notes played during piece given correct beginning and end notes.	10%	up to 50%	✔
Eye Tracking Robustness	Eyes moving off screen, eyes looking at least few measures randomly, erratic eye movements, etc	no false positives	no false positives	✔

Figure 7: Results from all major tests