Sonic Score Saxophonics

Lin Zhan, Junrui Zhao, Jordan Li

Department of Electrical and Computer Engineering, Carnegie Mellon University

Abstract— **Sonic Score Saxophonics is a system designed to enhance the learning experience by providing constructive feedback. The system collects the user's fingering and audio during the saxophone playing, compares the fingering and note with built-in music, and provides visual feedback. The user can select from a range of beginner-friendly music scale exercise sheets in the web application.**

***Index Terms*—Saxophone, Web Application, Sensor, Audio, Note, Pitch, Signal**

I. Introduction

Learning the saxophone is a venture that is fun and rewarding, but it also comes with considerable financial challenges. While the instruments themselves are cheap, the availability of rental and rent-to-own programs allow aspiring musicians to obtain instruments at relatively affordable prices, but what is not available is affordable instructors. Saxophone instructors can cost upwards of $100 per hour, which is not feasible for a lot of families. Self-learning the instrument, without an instructor, is difficult, due to the development of bad playing habits, which, once formed, are difficult to fix. Even for well-off players, it is not practical to always practice with an instructor, due to both cost and time constraints, such as travel time for the player to the instructor.

Sonic Score Saxophonics is designed to be an instructor at home for beginners, aiming to provide an affordable way for students to avoid bad habits when practicing at home. The system mainly focuses on tone matching along with fingering detection, as one common mistake beginners make is overblowing, which causes overtones, which are pitches that are above the correct pitch. The project is not aimed to completely replace instructors, as their subjective inputs on factors such as tone quality and articulation are difficult for computers to analyze, but rather to be an assistant to help students practice at home.

There are no products currently on the market that directly address this concern. Other workaround solutions all have their disadvantages. Hiring online practice assistants is not cost effective, and playing on a MIDI-based electronic wind controller can help players with fingerings, but doesn't help players with their embouchure, which affects tones.

The main goal of the project is to develop a system that helps aspiring saxophone players develop their skills at home without also developing bad playing habits.

II. Use-Case Requirements

A. Accuracy

Fingering Accuracy

For saxophone players, regardless of their skill level, precise finger placement is essential for producing clear and melodious tones. Correct fingering is particularly crucial for saxophone beginners, as any incorrect key press can cause the note to go out of tune and may foster bad habits. Hence, our system is designed to distinguish between correct and nearly correct finger positions, and detect the mistakes with an accuracy rate of at least 90%. This feature is crucial to those new to the saxophone, as it allows for precise corrections and advancements in their playing skills.

Sound Accuracy

In addition to finger positioning, the ability of our system to accurately identify and evaluate the pitch and rhythm of notes is essential to our users. With a sound detection accuracy of at least 90%, our system caters to the users' need for improving their sound quality. This level of precision assists users to control pitch and tone as well as the length of each note, enabling them to produce sounds that are more in tune and match their intentions.

Acknowledging that users, especially beginners, cannot achieve perfection in their play, our system is designed to tolerate variations within an acceptable range. This approach ensures that minor deviations in pitch will not be recognized as mistakes, aligning with the users' needs.

Feedback Accuracy

The provision of accurate feedback is a fundamental requirement from the user's perspective. Our system targets a feedback accuracy rate of 80%, ensuring that users receive reliable guidance on their performance. With this accuracy rate, we minimize the chances of the system marking incorrect actions as correct and marking correct actions as incorrect. This is crucial for learning since it helps users pinpoint areas for improvement and adjust their playing accordingly. By highlighting both correct actions and areas needing refinement, our system encourages a balanced approach to practice. Moreover, by minimizing the chances of erroneous feedback, our system prevents users from developing incorrect habits, ultimately leading to more efficient and sustainable progress in their saxophone learning journey.

B. Latency

For an optimal learning experience with our saxophone add-on system, the responsiveness of the feedback mechanism is critical. Our system integrates and compiles users' data after the users finish playing, and displays the result when users click on the replay button. To meet users' expectations for immediate feedback, the latency between the user clicking on the button and the results being displayed should be within 5 seconds. This feedback should include incorrect fingerings, out-of-tune nodes and practical suggestions for improvement, allowing the users to reflect on their play.

III. Architecture and/or Principle of Operation

Please see the appendix for the system diagram.

Our whole system can be divided into three subsystems: the fingering collection, audio processor and user interface.

The reason why we chose to combine both fingering and audio information is that saxophone as a woodwind instrument requires more than fingering information to determine whether the player is playing notes correctly. By integrating audio information with matching fingering data, our system can achieve a comprehensive understanding of the user's playing. Additionally, it allows for the identification of some specific fingering issues that can be hard to detect just from audio information.

When a user is playing the saxophone, its original audio will go into the audio processor, which includes pre-processing, pitch detection and rhythm detection. For the pre-processing, we will perform a SNR check to the audio to make sure it's above 20dB so that it has enough quality for further processing. If the audio quality is below this threshold, it will be rejected and the user will be notified through the web app. We also filter out input signals that have a frequency below 70 Hz and above 800Hz. This frequency range is obtained from the playable range of tenor saxophone.

Then the audio will go into the pitch processor, which analyzes the signal in the frequency domain by performing Discrete Fourier Transform (DFT). We have set the tempo to be 60 bpm, which corresponds to one beat per second. The processor iterates through the signal using a sliding window with a size of 1/8th of a beat and extracts the dominant frequency in that window. Then it matches the dominant frequency of this interval to notes' frequency to determine what note is detected.

Aside from the pitch processor, the audio also goes through the rhythm processor, which determines the length of one note. We use Scipy's find_peak() function to find the onset of each sliding window. The output of the rhythm processor is a binary array that represents if there's notes detected in each 1/8th of a beat interval. Finally, we combined the output from the pitch processor and rhythm processor to generate an array of the detected notes and their lengths, which will then be sent into the web app backend.

Fingering collection happens at the same time as audio processing. During the saxophone players' playing, the fingering data will be collected by sensors installed on the saxophone's air doors. All of the sensors are wired to a PCB containing Sparkfun ESP32 controller. The controller will process the data and send note packages to the web app backend.

The web app contains a frontend user interface and a backend integration system. The user interface will have a main page including three sections: practice, learn and user statistics. The practice page allows the user to record their practice songs and receive feedback. The learning page will include a list of introduction videos and articles about saxophone learning, as well as some practice sheet music. The user statistics page includes history feedback for each practice song, which helps the user to improve based on past performance. The backend system takes input from both the fingering collector and the audio processor and matches the input with built-in sheet music. It will generate visual feedback and messages of how to improve, and display them on the practice page.

In the development of our system, several fundamental engineering principles were employed to ensure that the final product was both functional and user-friendly. One key engineering principle applied was the modular design approach, which involves breaking down the system into separate and manageable modules that can be developed and tested independently by each team member before integration. Another crucial engineering principle we utilized was the application of signal processing techniques, specifically the Discrete Fourier Transform (DFT). The DFT is essential for converting the time-domain saxophone audio signals into the frequency domain. It allows for accurate pitch detection, which is central to the functionality of our system.

The scientific principles in our project primarily include sound and digital signal processing. Understanding sound waves and how they propagate through the environment is fundamental for accurately placing sensors on the saxophone to capture data effectively. We apply this knowledge to use hall effect sensors to determine which saxophone keys have been pressed.

The mathematical principles applied in our system are centered around the DFT and sliding window technique used in signal processing and integration. DFT is a mathematical transform that decomposes a time-domain signal into its constituent frequencies. And the sliding window technique is based on principles of convolution and windowing in mathematics. This technique ensures that our system can track changes in pitch over time, providing dynamic and responsive feedback to the user.

IV. Design Requirements

A. Accuracy

The accuracy of the system is integral to the user experience of the system. Our use case states that the fingering collection system is 90% accurate, and this can be divided into key accuracy and transmission accuracy.

The key accuracy is the larger source of uncertainty due to the large number of keys in the system. Each key's accuracy can be described as in Equation 1 below:

$$key\ accuracy = \frac{\#\ of\ sensor\ output\ changes}{\#\ of\ times\ key\ is\ pressed/released} \quad (1)$$

There are 23 keys in the system, and each of them will have its own key accuracy. The overall system will have its accuracy based on the accuracy of each individual key. Some keys are determined by multiple sensors, since the saxophone key system involves several keys moving the same valve, or vice versa.

The transmission accuracy is a smaller source of uncertainty, due to the reliability of serial communication through USB. The wired connection used in the system allows for dependable data transmission between the fingering collection controller and the computer.

The pitch detection system should have an accuracy rate of 90%. This means that the combination of the audio collection and audio analysis should be 90%. Audio collection via a microphone should be close to 100% accurate, which leaves the burden on the algorithm. Thus, the pitch detection algorithm should have an accuracy of at least 90%.

The feedback accuracy should be at 80%, and this is a combination of the two above-mentioned accuracies. Since the subsystems both have at least 90% accuracy, the feedback system would have an accuracy of 81% approximately, which is within the use case limit.

B. Latency

The latency between the feedback being displayed and the user clicking on the replay button is also crucial to the user experience. For music sheets of all lengths, the latency should roughly equal, since the audio output and fingering output have already been generated as text files when the user stops playing.

There are several components to the latency: determining the sliding window, matching notes with reference notes within the sliding window, matching fingering with reference fingering within the sliding window, and ultimately comparing and presenting the information required for the UI.

Determining the sliding window should provide minimal latency, as it only involves simple calculations. We estimate this process takes mere milliseconds.

The primary sources of latency lie in matching notes and fingering within the sliding window to their respective reference data. This involves iterating through each data entry and accumulating time. Additionally, sorting the results is necessary to determine the most probable fingering and pitch if the reference fingering and audio are not within the sliding window. Particularly for fingering data, with its fast baud rate, the sliding window may contain numerous entries, lengthening processing time. This step should be completed within 3 seconds.

Comparing and presenting information for the UI does not become a bottleneck. However, due to the need to tolerate slight note pitch deviations and consider similar-sounding data as matching, this step may take longer than the first step, with 1 being second sufficient.

This allows approximately 0.8 seconds for the web app to update and generate feedback based on the received information, including any network latencies. This timeframe should suffice for most network situations.

Following the initial request, only slight adjustments to the sliding window are necessary, avoiding the need to recalculate large amounts of data. Thus, this latency minimally affects the update for each individual note.

V. Design Trade Studies

A. Choice of Sensor

The sensor is arguably the most important part of the fingering collection system. We chose to use hall effect sensors, since they don't rely on the detection of finger forces, and are small enough to be installed in the tight environment of a saxophone.

We also considered using film pressure sensors, installed on each individual key on the saxophone to detect finger movement, but film pressure sensors cannot fit onto some special-shaped keys of the saxophone, such as the palm keys on the left hand. It also could produce errors if the player's fingers rest on certain keys even when they are not pressing the key. The natural resting position of several keys is touching the key, which a film pressure sensor can detect as playing when in reality it is just resting. However, the film pressure sensor may be necessary on certain keys that do not allow for reasonable installation of hall effect sensors and magnets.

B. Audio Processing System

1) Pre-processing

We first plan to only use band-pass filtering to filter out signals with too high or too low frequencies. But we found out that it's not enough to cancel out noises. Therefore, we added an SNR check to check the input signal's dB level and reject any input that has unsatisfying quality. We decided to set the threshold to 20dB based on a previous ECE capstone project "Musician's Scribe", in which they tested that signals that have below 20dB are hard to perform a pitch detection algorithm. We have performed several tests on different audio inputs with various noise levels, and found that the 20dB threshold has the best performance in determining the audio quality.

2) Pitch detection

We chose to use a sliding window approach to iterate through the input signal. For now, we're using a window size of 1/8 of a beat. The reason that we chose 1/8 instead of 1/16 is that it can be less computationally demanding to detect the notes to the nearest of 1/8 of a beat. What's more, a window size of 1/16 of a beat is more oftenly used for songs with faster rhythms. Since our app focuses mainly on instructing saxophone beginners, our built-in practice songs have moderate pace, which makes a window size of 1/8 of a beat more appropriate.

The window size is dependent on the tempo of the input audio. We have tried to directly extract the tempo of a given audio using functions from python library Librosa, but the result wasn't accurate at all. Therefore, we switched to a fixed tempo of 60 bpm. This approach enhances the reliability of pitch detection, especially for beginners who should benefit from a consistent tempo during practice sessions.

3) Rhythm detection

Initially, we planned to only focus on the pitch detection and avoid developing a rhythm detection algorithm. However, through our progress we decided to deal with music that is approximately 1 minute long. That means a rhythm processor is needed since we need to determine how long each note lasts.

We first attempted to find the peaks of the whole audio input at once. But we realized that we still need to match the output array with the notes detected in the pitch detector, which is extra work. Therefore, we decided to apply a sliding window with the same size of the pitch detector, and thus the outputs can be aligned easily.

C. Web Server

1) Cloud Servers

Initially, we considered cloud servers like Heroku as our hosting option, because of their scalability and robust infrastructure. From previous experience and research, we know that these servers are capable of effortlessly managing user traffic, which is particularly advantageous for projects with significant fluctuations in usage. However, our project only involves some basic saxophone learning functionalities, and targets at saxophone beginners, a relatively small group. Hence, we don't expect our web application to have significant traffic that can only be handled by cloud servers. Therefore, the expansive capabilities offered by the cloud servers are much more than we require.

The decision against cloud servers was influenced by three main factors, the potential for ongoing costs that could be reduced through local hosting, the complexity associated with cloud infrastructure management, and concerns about latency. Compared to local hosting, cloud hosting has higher latency. Our project aims to provide quick feedback after users attempt to view the results, so that it's more convenient for them to look at the results multiple times and adjust their fingering and pitch based on that. High latency could hinder the user's experience in using the web app.

Moreover, our project handles some user statistics and audio data, raising data privacy concerns. Given these considerations, we steered away from cloud hosting. Local hosting emerged as the more suitable option, aligning with our project's modest size and specific needs while ensuring greater control over user data and minimizing operational costs.

2) Hosting Locally

Hosting locally is ideal for our saxophone learning project, especially since it effectively meets the demands of a specialized, beginner-oriented user base without the need for extensive scalability.

Firstly, it provides a controlled environment for development and testing, allowing for immediate iterations and updates without the risk of affecting a live audience. This is particularly valuable for a project targeting beginner saxophonists, as we can finely tune the learning experience based on direct feedback and observations.

Additionally, local hosting minimizes operational costs and simplifies compliance with data privacy, which is crucial since our application handles user statistics and audio data. By running the application on a local port, we also ensure that data security is tightly managed within our network, avoiding potential vulnerabilities associated with external hosting.

Finally, the simplicity of setup and maintenance with local hosting aligns with our project's scale, making it an economically and technically feasible choice for our specific needs.

While local hosting limits accessibility from outside our immediate network and requires hands-on maintenance, these limitations are manageable for our project. Given our targeted user base and the scope of our application, these constraints are acceptable and do not hinder our ability to deliver a quality learning experience.

D. Switching from real-time to offline

We transitioned from real-time to offline processing due to the challenges that come with real-time audio processing. Though real-time processing allows for easier synchronization, it leads to noticeable accuracy issues. To address this, we now save real-time fingering data and later merge it offline, allowing users to replay their practice sessions and review any errors and feedback. This method is also more practical because it's often difficult for users to focus on immediate feedback while practicing.

VI. System Implementation

A. Fingering Collection System

The fingering collection system consists of a series of hall effect sensors and magnets attached to different air doors on the saxophone and an ESP32 controller that collects and processes the data. The hardware design is based on a MIDI saxophone controller called JazzHands developed by AndrewChi, with several changes.

Hall effect sensors can detect the presence of magnetic fields and change its output voltage based on whether there is a presence of a magnetic field or not. Hall effect sensors are installed on the body of the saxophone, and magnets are attached on the valves of the saxophone. When a key is pressed or unpressed, the corresponding valve will either go up or down, depending on the key, causing the magnet to move, which causes a change in the hall effect sensor's voltage output. There are a total of 21 hall effect sensors, corresponding to all keys on a typical saxophone. Some keys do not need their own valves, as they can rely on a combination of existing sensors to determine whether the keys are closed or not. Thus, 21 sensors can account for all 23 keys.

The input from the hall effect sensors are wired to a PCB, which acts as the central control of the fingering collecting system. The PCB contains the SparkFun ESP32 Thing controller, pull-down resistors for the hall effect sensors, and shift registers. The inputs go to the shift registers, which are controlled by the ESP32 controller, which, after each read, will read from its data pin, one signal at a time, until all 21 inputs are read.

The wiring between the sensors and the PCB went through several iterations. The first design involved directly soldering cables to the hall effect sensors, which was both difficult to solder and would regularly fail after the soldering was completed, necessitating repairs. The second iteration was to use a strip board that would connect the sensor and the cable that goes to the PCB. One end of the strip board was the sensor, the other end was the cable that goes to the PCB. This design also did not work, because the strip board was too large to fit on a saxophone without modifications that could potentially cause shorts, since the saxophone is made of metal. The final design was to use ribbon cables that allowed the sensor to be directly plugged into the cable and the PCB. This design worked best due to its simplicity and ease of repair, but the cost of the system went up due to the ribbon cables being more expensive.

The plan was to use a film pressure sensor for the octave key, due to the octave key having two valves that do not open all the time, and that one of the valves is attached to the neck, which means if a sensor is attached there, the saxophone cannot be taken apart for transportation. However, there were several challenges regarding that: the film pressure sensor was difficult to get working, as the pressure sensor responded to very low levels of pressure, even including when a finger was simply resting on the key. The pressure sensor is also difficult to install onto the key itself, as permanent adhesive would have to be used to attach it, which would prevent easy disassembly of the system and cause permanent damage to the lacquer of the saxophone. A workaround was developed, in that a hall effect sensor is installed under the octave key itself. The design of the octave key allowed a magnet and a hall effect sensor to fit underneath, thus sensing whether the key is pressed or not.

Data was to be sent from the system to the web app using the MIDI standard, as it was an established standard that is used to communicate music data between different peripherals. However, MIDI could not account for fingering data transmission, so the system switched to using USB serial output to send fingering data to the web app.

B. Audio Processing System

The audio processing system includes three sub-systems: pre-processing, pitch detection and rhythm detection.

The pre-processing part includes checking the Signal-to-Noise ratio (SNR) of the input audio signal to make sure it has enough quality. We will reject input audio with SNR lower than 20dB to ensure the performance of the pitch detection algorithm in later parts. The formula we used is:

$$SNR \;=\; 10 * log_{10}(PowerSignal \,/\, PowerNoise) \tag{3}$$

The pitch detector takes the input signal from the preprocessor and performs pitch to note conversion. The audio data is divided into segments corresponding to 1/8th of a beat, based on the specified tempo. This segmentation involves calculating the duration of a full beat in seconds and subsequently determining the duration for an eighth note. The number of samples per 1/8th beat is computed by multiplying the sample rate by the eighth note duration. Then, each data segment is multiplied by a Hanning window. The windowed data is then subjected to the Fast Fourier Transform (FFT) to convert it from the time domain to the frequency domain. The formula we used is:

$$X(m, \omega) \;=\; \sum_{n=0}^{N-1} \mathrm{x}[\mathrm{n}]e^{-i2\pi kn/N} \tag{4}$$

After we gain the frequency of the interval with 1/8 of a beat, we convert the frequency into Musical Instrument Digital Interface(MIDI) notes, which is a standardized representation of musical notes. Once we get the MIDI note number, we can map it to a musical note. Equation 5 is the equation we used to calculate the MIDI note value, where f_n is the frequency we detected, and 440Hz represents an A4 note, which is the internationally recognized standard for musical pitch:

$$n \;=\; 12 * log_2(f_n \,/\, 440Hz) \tag{5}$$

Finally, the detected notes are transposed by 14 semitones to adjust the notes to the correct pitch. This is because the saxophone is a transposing instrument, which means that the pitch it produced does not match the concert pitch read on standard sheet music. Therefore, we want to transpose the notes to align the detected notes with the standard notation in sheet music.

For the rhythm processor, we apply a sliding window to the signal with 1/8 of a beat and extract peaks of it. We use the find_peaks function from Scipy library, which is a music and audio analysis library in python. This function takes a 1-D array and finds all local maxima by simple comparison of neighboring values (SciPy, n.d). We set the required parameter to be 1/8 of a beat of distance between the peaks detected.

C. Web App

The primary interface for users is through a web application, for which we choose Django for our web framework. It enables us to develop 3 distinct functionalities within our system, ‘practice’, ‘learn’, ‘user statistics’, as shown in Fig. 1.

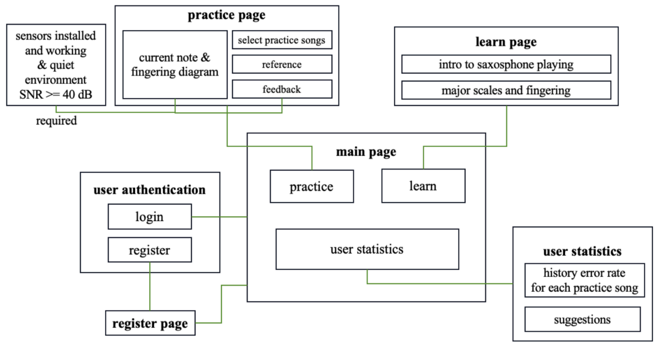

sensors installed and working & quiet environment SNR >= 40 dB
required
practice page
current note & fingering diagram
select practice songs
reference
feedback
learn page
intro to saxophone playing
major scales and fingering
main page
practice
learn
user statistics
user authentication
login
register
register page
user statistics
history error rate for each practice song
suggestions

Fig 1: the structure of the web application

The learning page is similar to a wikipedia page, with information about saxophone, fingering charts, articles and videos about saxophone, together with some common problems that the beginners may have. Users will be able to view some simple scripts and listen to the reference playings on this page. There are also external links to resources that are suitable for beginners.

The aim of the user statistics page is to provide users with a comprehensive overview of their progress, featuring historical error rates for each song practiced. To support this functionality, we utilize the builtin database SQLite3 in Django to store user information. This data will then be fetched and displayed on the user statistics page, ensuring that users have access to up-to-date insights on their experience with our web application.

The practicing page serves as the core of our application. It is designed to replicate the experience of a lesson, providing feedback after a user plays a practice song. Users must ensure that the sensors are active and functioning, and that the environment is relatively quiet with a Signal-to-Noise Ratio (SNR) of at least 20 dB before they can begin.

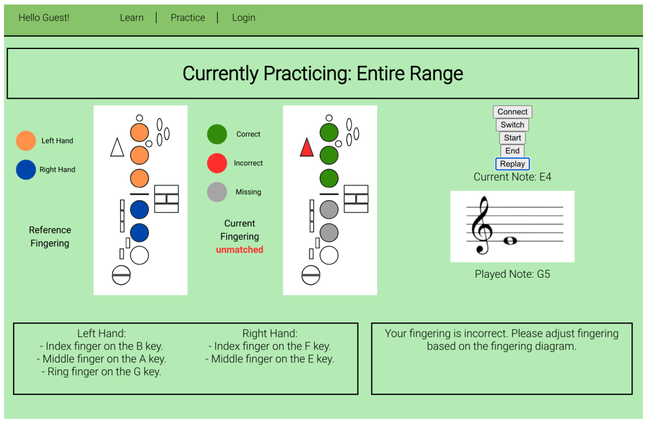
The structure of the practice page is organized so that both diagrams and texts can serve as tools to assist users in practicing. On the left, The UI displays a visual representation of the correct saxophone fingering using SVG components, enabling partially coloring to indicate correctness. To aid beginners who may not understand the diagram, a textual explanation for the reference fingering chart is attached below it. On the right, the current note and next note are displayed under the five buttons that the user can interact with.

By clicking on the connect button, the user can connect the web app with the fingering serial port. And then the user can switch between Entire Range, B flat scale and Mary had a little lamb and choose a music sheet to practice with. Once the user is ready to play, she should click on the start button, and the system will automatically start recording and reading from the serial port to get fingering data. After finishing the practice, she clicks on the end button to trigger the web app to download the recording, run the audio detection system on the recording to generate audio output, write the stored fingering data gained from serial port to fingering output file.

In order to integrate the audio output with the fingering output, we iterate through every reference fingering and note, record the current time in reference, calculate the sliding window to be (offset + curr_time - tolerance, offset + interval + curr_time + tolerance), and then apply the sliding window on user fingering and audio data to find whether there is a match between the reference data and user data, lasting for long enough time(at least 60% of the original length). If there is no matching, we will output the user fingering and audio as the data with longest duration in the sliding window.

After the integration is done, the replay button will enable the user to replay the fingerings, notes and feedback as many times as she wants. The UI provides visual feedback by highlighting the keys in red for incorrect fingerings, green for correct ones and gray for missing ones so that the user can quickly understand how to change the fingering. On the right, the current note in script and the detected note are shown, in case the user presses fingers at the correct positions but produces incorrect sounds, as in Fig 2.

Besides the visual cues, the system also offers detailed textual feedback to users' performance to guide users through the practicing process. If the fingering is unmatched, the feedback will be instructing the user to change the fingering, and if the fingering is matched and the detected note is unmatched with what the user intends to play, the feedback will provide possible reasons, for example, issues with embouchure, breath, or the instrument to help the user correct the playing.



Left Hand:
- Index finger on the B key.
- Middle finger on the A key.
- Ring finger on the G key.

Right Hand:
- Index finger on the F key.
- Middle finger on the E key.

Your fingering is incorrect. Please adjust fingering based on the fingering diagram.

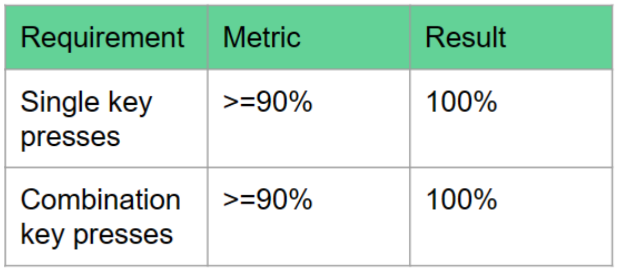
Fig 2: Practicing Page

VII. Test, Verification and Validation

A.Results for Fingering System

The fingering system test includes testing both single keys and a combination of keys for accuracy. Each of the 23 keys on the saxophone was tested for the single key test, and a chromatic scale was played for testing the combination keys. The chromatic scale would cover every key as well as most reasonable combinations of keys pressed on the saxophone. Each test was repeated for 10 times, and the average was taken.

Figure 3 shows the results of the accuracy tests.



Requirement	Metric	Result
Single key presses	>=90%	100%
Combination key presses	>=90%	100%

Fig 3: Testing result of the fingering system

The fingering system was 100% accurate on all tests performed, which was expected, given the robustness of hall effect sensors and the ESP32 controller. These results meet both the use case and design requirements, as both dictated that the fingering collection system should be at least 90% accurate, and the result was able to exceed that.

B.Results for Audio Processor

To assess the system's accuracy in note detection, we tested its performance using various inputs. Our tested inputs include: single notes that cover the entire range of tenor saxophone(from low B flat to high F), C major scale, B flat scale, and Mary had a little lamb with fast and low tempo.

Below are the results of them:

	Tempo (bpm)	Accuracy of Pitch detection	Accuracy of Rhythm detection	Latency
Single Notes	-	100%	100%	< 1s
C major scale	60	93% (14/15)	100% (Error within 0.2s)	<=2s
B flat scale	60	93% (14/15)	100% (Error within 0.2s)	1.5s
Mary had a Little Lamb v.1	150	34%	70%	1.5s
Mary had a Little Lamb v.2	60	75%	90%	1.3s

Fig 4: Testing result for audio processor

The pitch accuracy is detected by matching the notes from the output array with the reference notes we know. The accuracy of single played notes reaches 100% with less than one second of latency. The accuracy of both C major scale and B flat scale reaches 93%, which means that there is one note being detected incorrectly out of 15 notes in total. We tested Mary had a Little Lamb with two different versions, one with a tempo of 150 bpm and another with 60 bpm. We noticed that the result accuracy for the latter version is much higher, which is due to the way we implement the sliding window.

The rhythm accuracy is detected by matching the note length detected with reference note length in the sheet. We will tolerate an error within 0.5s since human players aren't as accurate as machines. The accuracy for single notes and music scales reaches 100%, and 90% for Mary had a Little Lamb with a tempo of 60 bpm.

Overall, our testing result for the audio processor reaches our metrics. The ability of our system to detect notes from music scales is above our expectations, while the accuracy for detecting more complex songs can still be improved.

C.Results for web app

We tested the web app performance after integration by playing according to it like a real user, and observing the replayed results. Our tested inputs include: single notes that cover the entire range of tenor saxophone(from low B flat to high F), B flat scale, and Mary had a little lamb. Fig 7 is a table of the results.

	Accuracy	Latency
Single Notes	100%	4s
B flat Scale	87%	4s
Mary Had a Little Lamb	69%	4s

Fig 7: Testing result for feedback accuracy after integration

For single notes, the system demonstrated perfect accuracy at 100%, maintaining a latency of 4 seconds. Performance slightly declined with the B flat scale, achieving an 87% accuracy rate with the same latency. Most of the unmatching in the feedback comes from the detection of audio rhythm, since a note is not always kept in the same pitch in a real-world playing. All those tests meet with our 80% accuracy metrics.

The most challenging task was Mary Had a Little Lamb where accuracy further dropped to 69%, again with a 4-second latency. Because Mary Had a Little Lamb is more similar to a real music piece, and each note's pitch and rhythm is less stable than previous ones. The ability to handle complicated music pieces remains a limitation in our system.

Above results indicate that while the app performs exceptionally well with simpler inputs, its accuracy decreases as the complexity of the musical pieces increases. Latency remains consistent across all tests, ensuring a smooth user experience.

VIII. Project Management

A. Schedule

Our schedule was created based on individual team member responsibilities and their knowledge areas. The web app and the fingering collection system are separate systems that can be developed separately in parallel, and then integrated together. Both systems can also be tested separately before integration, as the web app can be tested using tools such as tone generators and dummy inputs, and the fingering collection system can be tested by observing its outputs independently. The schedule will also leave time for final integration of the two components. Due to the more extensive time spent on our respective components, our final schedule allocated less time for integration and integration testing compared to the schedule outlined in the design report. Apart from this adjustment, we didn't make many other changes to the schedule.

Please see Appendix B for our Gantt chart.

B.Team Member Responsibilities

Jordan Li is responsible for the fingering collection system. His main responsibility is to build out the system on a saxophone, which includes installing the sensors, developing software for the controller, and testing of the system before integration.

Junrui Zhao is responsible for the design and construction of the web app. Her responsibilities include designing the layout of the website, building both the frontend and backend of the web app and displaying related diagrams and texts.

Lin Zhan is responsible for the audio processing algorithms. Her main responsibilities include implementing code for the audio processor to convert the audio signal input to notes with rhythms.

Integration is the responsibility of all three members. Each person will make changes to their subsystem according to integration requirements. Junrui is responsible for initiating integration in the web app backend and designing the integration logic to handle the received data, ultimately presenting it on the web app user interface.

C.Bill of Materials and Budget

Please see Appendix C for BOM.

D.Risk Management

In our saxophone learning project, we tackled several critical risks to ensure the system performs effectively and is user-friendly. The primary concern was the overall response time of the system, encompassing both the sensors and the web application.

For the hardware section, we bought one set of redundancy parts for everything, from the sensors to the PCB itself. This came in handy two days before the final presentation, when the ESP32 controller's USB port fell off in an accident, and could not be soldered back on. The extra set of parts allowed rapid fabrication of another PCB without the need to overnight the parts, which could risk budget overuse and unfinished product.

For the web application, we utilized predetermined test inputs to evaluate its functionality before full integration. This testing allowed us to identify issues promptly. Encountering performance delays, we enhanced the application's speed through refined algorithms and more efficient data processing methods. We also stored results in local files, a strategy that accelerates the web app when users access replay features.

Initially, we aimed to process audio inputs in real-time, but test results indicated insufficient accuracy. Consequently, we shifted to post-session audio processing as a fallback design. We had anticipated this adjustment, so the transition didn't take much time.

By implementing these strategies, we effectively managed project risks, ensuring smooth and rapid functionality.

IX. Ethical Issues

One edge case of operation is the aspiring saxophone player using this system exclusively, without the guidance of a more experienced instructor. While self-learning the saxophone is possible, especially if the player already has a music background, not practicing with an instructor can lead to bad habits developing and potentially misguided efforts on unnecessary practices. This would hurt both the player and saxophone instructors, since the player would not get the musical experience he or she desires, hampered by bad habits and wasted time, while the instructor would lose potential income from having one less student. If this happens on a scale, the market for saxophone instructors would shrink, causing lower availability of instructors, especially in lower-income areas.

This can be mitigated by explicitly asking the players to visit an instructor on a regular basis to check in, or to limit the sale of such systems to a network of music shops or saxophone instructors.

Another edge case is related to health. Given that both the hand and mouth touch the saxophone, the materials used in the system must be safe for human usage. Saxophones are also expensive and fragile instruments, and sticking a network of sensors onto it can cause damage if not done properly. If the manufacturer of such a system uses toxic materials that can harm human health, diseases can happen to the saxophone player. For example, if the system includes the use of toxic glue as the sticking agent of the hall effect sensor, the player could accidentally ingest it and cause physical harm. If the glue used is difficult to remove, removing the system from the saxophone when selling it or if the player no longer needs it can lead to permanent damage to the saxophone.

This can be mitigated by stringent quality control by the manufacturer to ensure that materials used would not cause such harm.

X. Related Work

There are currently no online saxophone instructing apps. There are some previous projects that work as music transcriber, which takes audio as input and generates sheet music as output. The idea behind these transcribers is similar to what we want to achieve in the audio processor part. Also, there is a past ECE capstone project "WoodwindMania" that works on a simulated learning tool that allows users to learn note fingerings for woodwind instruments. The project's idea is similar to what we want to achieve in the fingering collection part.

XI. Summary

To summarize, our system is quite successful since our system is able to provide accurate feedback for single notes input and scale inputs. The fingering detection and audio processor, which are two core functions of our app, are synchronized seamlessly. This ensures that saxophone beginners can receive immediate and precise feedback for fundamental exercises. However, for more complex songs such as Mary had a little lamb, the accuracy drops to 70%, which needs further improvement.

A. Future work

Our system currently still has many restrictions that need optimization. We plan to first add a function that allows the users to switch between different tempo ranges, instead of a fixed tempo. This will provide them with more flexibility in choosing which song to practice. We also want to improve the accuracy. This can be done by refining our current algorithm to better handle variations in rhythm complexity, perhaps through advanced machine learning models or more sophisticated signal processing techniques.

B. Lessons Learned

The most important lesson we learned through the course of the project is that early and frequent testing across a wide range of music inputs is crucial. Through testing, we noticed that the quality of the input audio file will directly affect the accuracy of pitch detection, sometimes making the error rate increase by 50%. We also learned that incorporating feedback from real musicians can help provide insights that are not immediately obvious from a technical standpoint.

Glossary of Acronyms

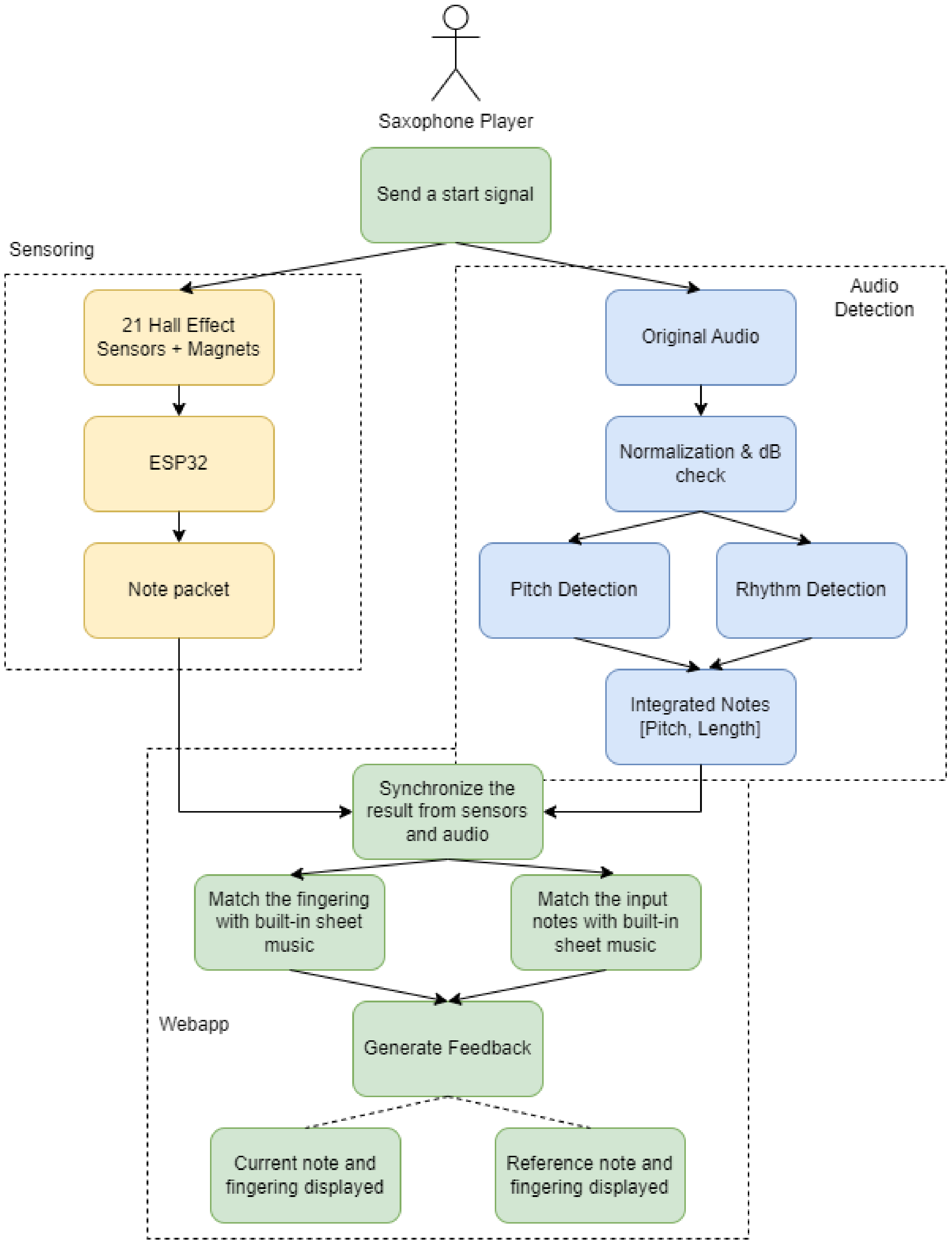
bpm - beats per minute
MIDI - Musical Instrument Digital Interface
MQTT - Message Queuing Telemetry Transport
OBD - On-Board Diagnostics

RPi - Raspberry Pi
SNR - Signal-to-Noise Ratio
Short-time Fourier Transform - STFT
UI - User Interface

References

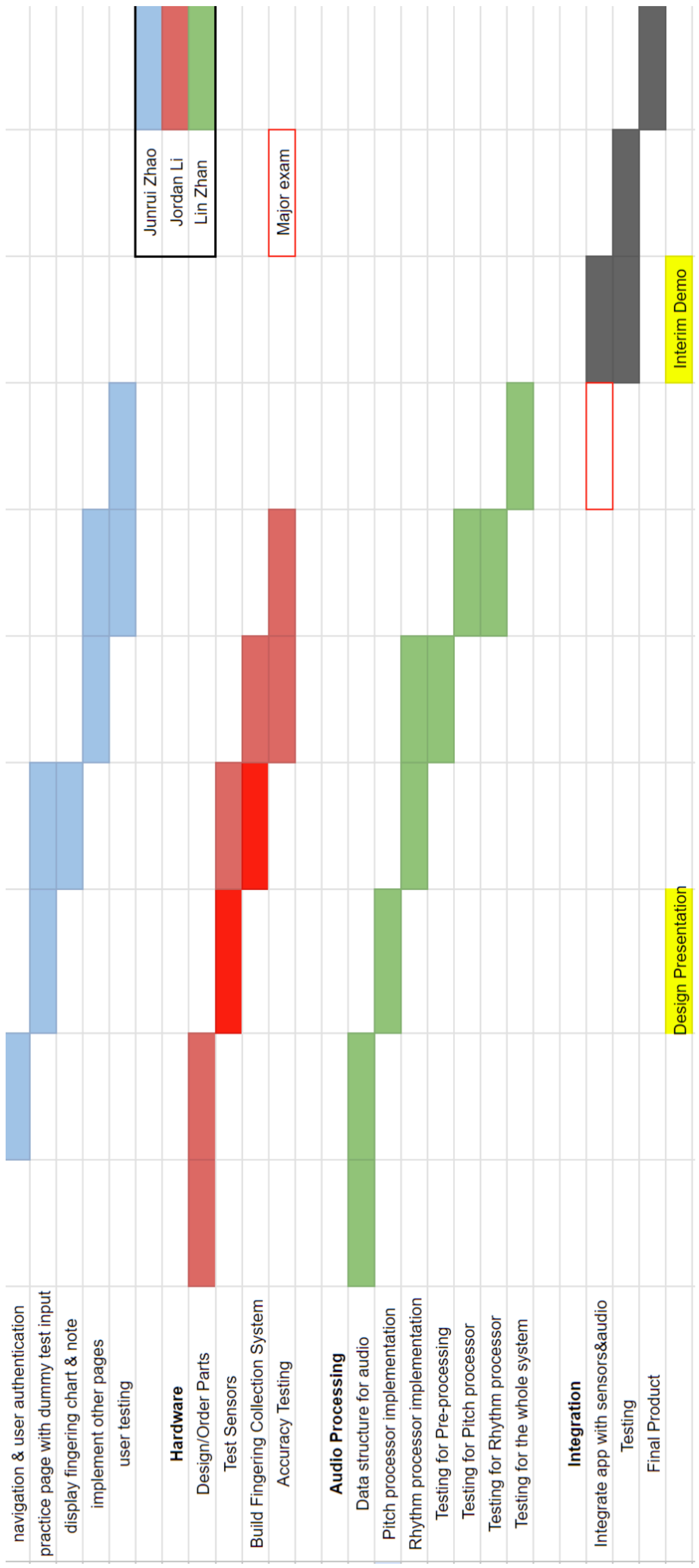
[1] AndrewChi, *Instructables*, Accessed on February 29, 2024, [Online]. Available: https://www.instructables.com/Jazz-Hands-Hybrid-Saxophone/

[2] Scipy User Guide v1.12.0, https://docs.scipy.org/doc/scipy/tutorial/index.html


Saxophone Player
Send a start signal
Sensoring
21 Hall Effect Sensors + Magnets
ESP32
Note packet
Audio Detection
Original Audio
Normalization & dB check
Pitch Detection
Rhythm Detection
Integrated Notes [Pitch, Length]
Synchronize the result from sensors and audio
Match the fingering with built-in sheet music
Match the input notes with built-in sheet music
Webapp
Generate Feedback
Current note and fingering displayed
Reference note and fingering displayed

Appendix A: System Diagram

Appendix B: Gantt Chart



Appendix C: Bill of Materials

Description	Model Number	Mfr.	Qty.	Unit Price ($)	Total Price ($)	Notes
Hall Effect Sensor	DRV5023AJQLPGM	Texas Instrument	60	0.708	42.48	30 more purchased to account for solder damage and change in cable
Magnet for Hall Effect Sensor	8015	Radial Magnets, Inc	40	0.299	11.96	10 more purchased to account for magnets that fell
Female MIDI DIN Connector	KCDX-5S-N	Kycon, Inc	3	1.63	4.89	Not used
100K Ohm Resistor Array	4116R-1-104	Bourns, Inc	6	2.5	15	Used 3 in final project
10K Ohm Resistor	CFR-12JR-52-10K	Yageo	5	0.1	0.5	Not used
8-bit Shift Register	SN74HC165N	Texas Instrument	6	1.35	8.1	Used 3 in final project
IC DIP Socket	1-2199298-4	TE Connectivity	10	0.199	1.99	Used 6 in final project
SparkFun ESP32 Thing Microcontroller	DEV-13907	SparkFun	2	23.46	46.92	One was a backup, USB port broke during testing
Sugru	I000953	Tesa	2	19.99	39.98	One more purchased due to higher than expected usage
40 Pin Male Pin Header Connector	1	Hotop	1	7.95	7.95	
Velcro Strips	N/A	Aniced	1	4.51	4.51	
Adhesive Dots	PGP55	Pritt	1	6.1	6.1	
MIDI Cable	108533	Monoprice	1	6.73	6.73	Not used
PCB	N/A	JLCPCB	1	37	37	
Tenor Saxophone	N/A	Bundy	1	1,000.00	Borrowed	
Clip on microphone	XLR	Todlinkoc	1	12.99	12.99	Not used, bought after design report

	2000					
Ribbon Cable	PRT-10363	SparkFun	20	1.06	21.2	Bought after design report
Ribbon Cable	PRT-10373	SparkFun	11	2.1	23.1	Bought after design report
Ribbon Cable	ED-DP_L100_F-F_120pcs	EDGELEC	1	19.99	19.99	Bought after design report
Total					**311.39**	