Sing us a song, you're a piano pi!

Marco Acea, Angela Chang, John Martins

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—For centuries, pianos have been used to play music. "Player pianos", containing mechanisms that allow them to sound without human control, were first introduced in the late 19th century. This mechanism allows the instrument to be freed from its traditional constraints: the number of frequencies produced is no longer limited to a meager number of ten fingers.**

***Index Terms*—Signals and Systems, Embedded Systems, Internet of Things**

I. Introduction

This project seeks to take advantage of player pianos to simulate human speech. Human speech is made up of many frequencies, and enough keys - and their frequencies - combined can reproduce speech with high enough fidelity that a player piano can speak and be understood.

Different modules of this project will come together to translate the phonemes of human speech into timed key presses on a piano. The audio processing module will receive audio input. This input information will be sampled for different frequencies and amplitudes. A smooth average will be taken at each frequency that corresponds to a piano key. We will repeat this at each time period. The key scheduling module will thus take the output of the audio processing module and use the time and amplitude information to determine the keys that need to be played at different times.

II. Use-Case Requirements

There are four major requirements for our use-case scenario. The first thing we need access to is recordings of our users' voices. To achieve this, we'll build a user interface into our web application that allows users to record their voice and send it to the backend. For the user experience to be pleasurable, our UI needs to have a ~200ms end-to-end latency. This means that for any user interaction, there should be at most 200ms before any new actions can be taken, for example, listening to the post-processed audio produced from a user's voice recording. We derived this number based on conventional advice regarding user response. Usually, 100ms is a limit for the user's flow of thought to stay uninterrupted and for the user to feel that the system is instantaneous. Since our recording system is not real-time like a website, we had leeway in this regard; we decided that 200ms is a reasonable time for a non-real-time response to feel smooth and acceptable.

Next, with the recording audio of a user's voice, we need to extract the frequencies that make a user's speech. We need to gather information on how these frequencies change throughout time to generate a sequence of keys to be played. To achieve this, we divide the incoming audio into 'windows' which we can use to see how the frequencies of a user's voice change with time. A metric of success for this requirement involves being able to accurately estimate the frequencies and amplitude of each frequency within a user's speech for each of those time "windows". We decided that an 80% estimation accuracy for these frequencies and amplitudes would ensure our audio processing transformation introduces as little error as possible.

Once we have a description of the keys we need to play on the piano, those notes need to be scheduled over time, onto a physical device so that the piano keys are played correctly. Errors introduced in our implementation of this requirement can affect the intelligibility of our piano's output. Inaccurate timing in our scheduler will affect the timing of syllables the piano needs to produce, by delaying, elongating, or speeding up syllables. To mitigate these errors, we've decided that our scheduler should miss less than 5% of timed syllables. This will ensure that any errors in timing will not affect the entirety of our system's output.

Lastly, we need to implement a physical device that will actuate the keys on a piano and produce the sounds we aim to recreate. The success and accuracy of this requirement encapsulate the overall performance of our system's pipeline. Since there is a much smaller range of notes possible with a piano, we understand that the output of our piano will never be an exact copy of a user's speech. Therefore, we're requiring our physical device to have an 80% fidelity rate. In other words, the output speech of our piano should be intelligible to a user 80% of the time.

III. Architecture and/or Principle of Operation

While planning our overall system architecture, we realized the importance of modularity within the different subsystems. Allowing each subsystem to treat the others as black-boxes creates an opportunity to isolate subsystem tests and debugging with simulated inputs/outputs. We've grouped our subsystems into 4 main areas: user interface, audio processing, note scheduling, and physical performance. Figure 1 illustrates how the subsystems interact with each other.

Our system begins in the user interface, where users will interact with our web server via a browser on their personal devices. The user will be able to record audio files in-browser or choose amongst previously recorded files to play on the

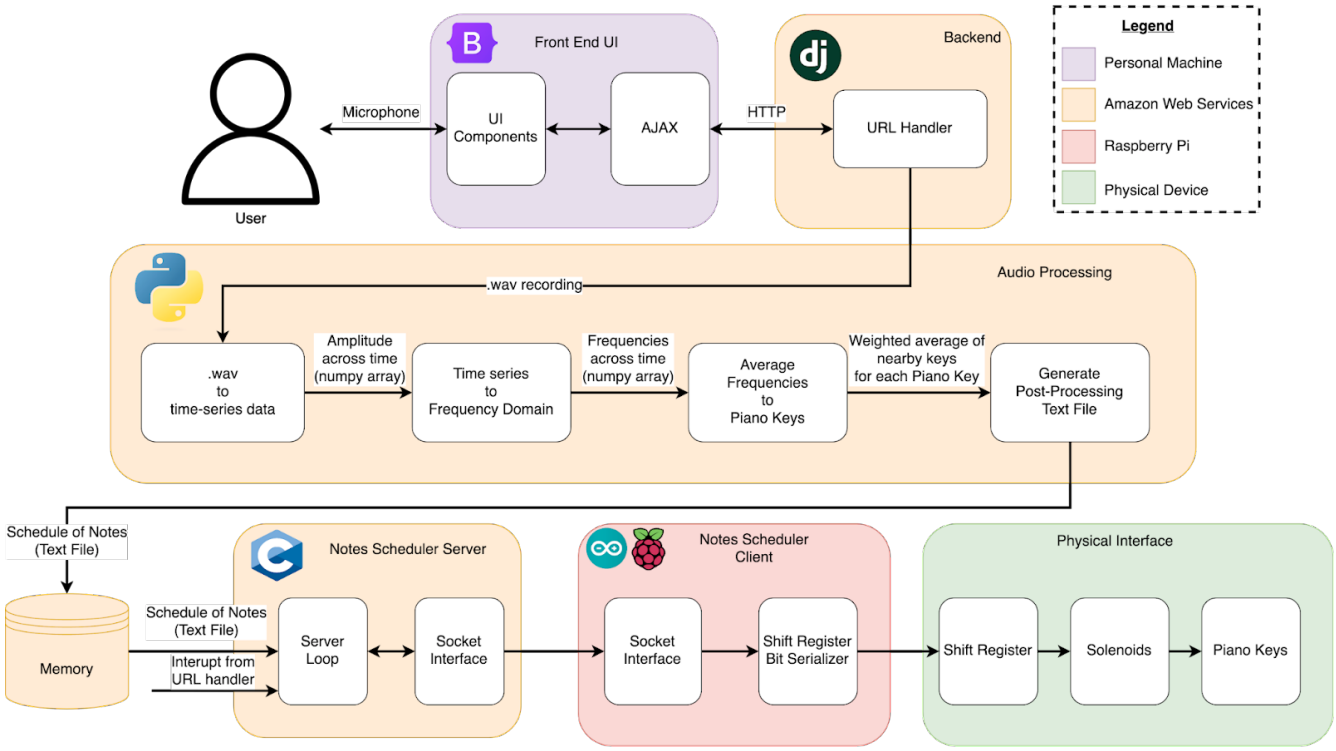

Figure 1 Overall Architecture

piano. The user interface handles obtaining the audio recording file, as a .wav file, and passes it along to the audio processing subsystem hosted on our backend server.
The audio processing subsystem converts the input audio into the collection of frequencies that make up these sounds across time. The audio processor then maps the frequencies at each timestamp seen in the audio recording to the frequencies playable by each key of the piano and encodes this data in a text file to be sent to the note scheduler hosted on the backend server.

The notes scheduler uses a socket interface to send the Raspberry Pi data on which piano keys to play at any given time. The Raspberry Pi then serializes data on which keys to be pressed and sends it bit by bit onto a shift register.
On the physical interface, the shift register outputs these control signals in parallel to the solenoids that actuate the piano keys.

IV. Design Requirements

As noted in our use-case requirements, we have 4 main requirements that each influenced the choices we made when designing our system.

A. Web App Latency

We are requiring that there is a <200ms latency period between interacting with our web application and noticing a change in the playing of audio on the piano. To ensure this, we are hosting our audio processing subsystem on the backend of our web application on AWS. This will allow us to use the powerful computers of the AWS EC2 instance to process the audio very rapidly. This leaves most of our 200 millisecond latency time for the transmission of data from AWS to the Raspberry Pi via sockets. We are estimating that this should be plenty of time since we are condensing the scheduled note information being sent into simple text files of bits describing which keys need to be played. To improve our transmission speeds, we will be testing an optimal balance between the size and frequency of data packets to minimize sending times. Lastly, the Raspberry Pi will immediately begin sending the received stream of bits to the shift registers to get played by the solenoids.

B. Frequency and Amplitude Extraction Accuracy

We understand that there will be some inevitable loss when translating the frequencies of the human voice to the finite and relatively small number of keys on a piano. To make sure we are maximizing the ability to play the voice as close to the original recording as possible, we need to ensure that we are extracting the frequencies and amplitudes from the recording as correctly as possible. We have chosen an 80% accuracy rating for the extraction of frequencies and their amplitudes as described in section N. This information is extracted from the voice recording by first splitting the recording into several windows of time and then processing each window through a Fourier transform, which converts the recorded data from amplitude across time to amplitude across frequency. We will be running tests to figure out the most optimal time window

for our Fourier transform such that the accuracy of the frequencies and amplitudes extracted is maximized.

C. Note Scheduling Accuracy

Next, we want to design our system to schedule notes to miss or incorrectly play less than 5% of the syllables spoken in the voice recording. Human speech, in English, is typically spoken at 10 to 15 phonemes (distinct sounds), or 4 to 7 syllables, per second [3][4]. As our system seeks to emulate human speech, our rate of producing sounds should not differ much from this rate. Typically, a piano key on an analog upright piano can be pressed up to 15 times per second. This limitation comes from the mechanism that moves the piano's hammers to its strings, known as the action. We will be using a digital piano, which does not have an action. However, digital pianos are programmed to resemble analog pianos as much as possible, and thus we will still use 15 times per second as a limitation for our piano. We have tested the maximum actuation rates of some test 5N solenoids and found that they can actuate at rates up to 16 Hz. Since the solenoids can fire up to 16 times per second, we will be able to create up to 16 distinct sounds in 1 second, allowing us to capture the upper bound speeds of typical English speech.

D. Fidelity Rate of Final Piano Playback

Lastly, we are aiming to achieve an 80% fidelity rate, meaning that 80% of the words spoken in the original voice recording will be able to be identified when listening to the piano playback. To ensure this, we must be able to play the correct frequencies at the right volume levels at the right times. To ensure correct frequencies can be played in the first place, we start with the frequency range of human voices. We found that the fundamental frequency for human voice can be as low as 85Hz and that telephone communication captures at frequencies up to 4000Hz. This gives us a good approximation of the range of human voices. By applying the equation on figure 2. We found that to capture the ranges of the human voice, we need all the keys from 20 (82Hz) to 88 (4186Hz).

$$n = 12 \log_2\left(\frac{f}{440\,\text{Hz}}\right) + 49$$

Figure 2: n is the nth key of the piano starting from the leftmost key of an 88-key piano

Next, we need to make sure that we can most accurately map the frequencies obtained from our Fourier transform to the frequencies playable by the piano keys so that we can best determine which piano keys are needed to play a particular sample of the voice recording. To do this, we will be implementing audio blurring. This involves taking surround frequencies around the frequencies of each piano key, averaging their amplitudes and weighting them by how distant they are from the frequency of the piano key, and determining if this average passes a threshold determining that the key should be played. For example, in the case where the voice audio file contains weak amplitudes at exactly the frequency of note A4 of the piano (440Hz), but contains strong amplitudes at frequencies 437, 439, and 445, we will be able to capture these surrounding frequencies and determine that the note A4 should indeed be played to capture the slightly different frequencies. On top of that, humans are only able to discern a > 1% frequency difference between two tones.
To best reproduce the amplitude of each frequency that makes up any given sample of the voice recording, we will be encoding the bits sent to the solenoids with a PWM signal, allowing us to control the volume at which the keys are pressed.

V. Design Trade Studies

A. Virtual vs. Physical Piano Performance

One of the biggest challenges when initially designing the scope and specifications of our project was determining whether we would be recreating the processed speech virtually through a virtual piano interface or physically with solenoids pressing keys on a real piano. We understood that building a physical key pressing system would introduce a lot of complexity and potentially cause us to focus on mechanical systems that were out of the scope of our ECE-focused design requirements. After deliberate research and preliminary design of the circuitry and hardware needed to create the physical system, we decided that we would attempt to create the physical performance interface, and fall back to implementing the virtual performance interface if the physical system presented too many issues or troubles focusing on mechanical areas. This choice influenced several other design choices (as in the following section), and to plan around this fallback, we also derived a proof-of-concept design that we would build and test to help determine the feasibility of building the entire physical interface. Our main drive for developing this physical interface lay in providing the user with the most engaging experience possible in seeing how the piano can be used, outside the possibility of human players, to recreate speech and extend the realm of the typical use of pianos. We also coordinated efforts with Benjamin Opie, an electronic music professor here at CMU, to be able to have access to the digital piano practice room for the testing of our designs.
Just recently, we implemented our proof-of-concept design consisting of wiring up 3 5N solenoids connected to shift registers and MOSFETs for actuation. We were successfully able to actuate the solenoids in customized patterns based on serial bit streams inputted to the shift registers and outputted, in parallel, to the solenoids. This test helped us determine our power consumption and strategy when scaling this up to the full, 69-key system.

B. Web-Hosted vs. Hardware

When we were deciding where we would host and store our programs, we identified we needed 2 main components: a strong computational unit capable of performing complex signal processing on incoming voice recording quickly, and a way to host a fast-responding user interface that could be used to record audio and interact with the system like pausing/playing and loading past recordings.

We first looked at using a Raspberry Pi for both the audio processing and notes scheduling portion of the system. We found that what we gained in low data transfer times and

portability, we lost heavily in computing power. We quickly determined this would not work.

We next looked at using a Jetson Nano to do the audio processing, while keeping the Raspberry Pi for the notes scheduling. In this scenario, we could probably achieve performance with the stronger computer on the Jetson, however, we needed to take into account our virtual piano backup plan in case things didn't go as we wanted. If we needed to implement the virtual piano instead, we would need to transfer the processed audio data and key schedules to AWS, where the virtual piano would be hosted. On top of that, this would introduce uncertainty as to where to host our user interface. If we hosted it locally, users would only be able to record audio and access the piano system on our local machines, but we would be able to run commands hosted on our local Jetson much more quickly. If we hosted our user interface on AWS, this would allow any user to access the interface on their personal device for recording, however, we would need to transfer the entire audio file to the Jetson for processing, which could introduce a large lag time. These trade-offs brought us to our currently agreed-upon design choice of hosting the user interface and audio processing on an AWS EC2 instance and transferring the data to a Raspberry Pi for playback on the physical piano interface. With these choices, any user can access the web application on their personal device for recording. These recordings are seamlessly sent to the audio processing subsystem also hosted on AWS. By hosting our audio processing subsystem on AWS, we can take advantage of the powerful computers of the EC2 instance to perform the audio processing very rapidly. Lastly, if we must implement our virtual piano instead of our physical performance interface, the rest of the subsystems will be already hosted on AWS, therefore transferring data to the virtual piano would be optimal. Ideally, if we get this system toolchain working with our physical piano interface and we still have time in our project, we will work to migrate the web app and audio processing to a Jetson to remove any data transfer lag times and take advantage of the portability of having a fully hardware-based system.

C. Audio Time Sample Window

Next, we examined the trade-offs associated with choosing the window of time in which we sample our audio recording for processing. This time window represents the duration of each snippet of the voice recording that the Fourier transform gets performed on. We determine the total number of time windows, and thus Fourier transform operations, for each recording by the simple equation:

$$N = \frac{T_r}{T_w}$$

N is # of windows, Tr is time of recording, Tw is time of the window

Our goal with selecting an optimal time sample window is to minimize the Fourier transform error and the time needed to compute each transform as well as the whole recording. To do this, we will be performing tests on a range of time window values by inputting the same simple audio, with a range of different windows, through our Fourier transform. The time taken to perform the transform and its accuracy will be analyzed against a control Fourier transform with a control time window from a well-established python library. We hypothesize that a shorter time window will allow for much shorter audio processing times as we can run more transforms and send smaller packets of data to the notes scheduler more frequently. However, we would be sacrificing the accuracy of the Fourier transform in extracting the frequencies and amplitudes, which hinders our use case requirement. On the other hand, choosing larger time windows may slow down audio processing times since we would be computing the Fourier transform and sending data to the notes scheduler in larger packets less frequently. However, due to the larger time window, the Fourier transform has a larger sample of data to work with, thus improving the accuracy of the frequency and amplitude extraction.

D. Comparing Different Averaging Techniques

To propagate as much information about the frequencies that make up a user's voice throughout time, we want to average those frequencies around the discrete points that correspond to piano keys. We considered four options, each proposing more added benefit than the last. The first was to filter information at the frequencies that correspond to the piano keys. This approach loses the most amount of data on what the original frequencies of a user's voice were. However, this approach offers the benefit of being able to be implemented in hardware since the number of filters needed would be small.

Our second approach was to compute a moving average of all the frequencies before the point where a piano key lies. This improves on our initial approach but lacks information about the frequencies that lie after the frequency of a piano key.

Our third potential solution was to compute an average using the neighboring points before and after the frequency for our piano key. We think this is our best bet since this averaging technique is capable of propagating all of the original features from the original audio, albeit without the same resolution.

We considered a weighted neighbors average, where again we average the neighboring points around the frequency of a piano key but weigh each neighbor's contribution to the final average by their distance from the piano key's frequency. Although a valid way of averaging our input, it would cause frequencies lying between any two piano keys to not be propagated. Therefore, we think that our third approach is our best attempt at propagating all the information regarding a user's voice onto the piano keys.

E. Solenoid Choice

Originally, we were deciding between using 5 Newton solenoids and 25 Newton solenoids for actuating the piano keys. Looking at the solenoids available to us, the 5N solenoids were typically much less expensive and used less power to run. For our proof-of-concept design of the physical interface, we ordered 5N solenoids that needed 12V and 1A to run. We found they got quite hot when left activated for extended periods of time and that 5N is just barely enough to actuate a key at an audible level. Another problem with using 5N solenoids is that since they need the full power to actuate the key, there will be no room for adjusting the volume of the

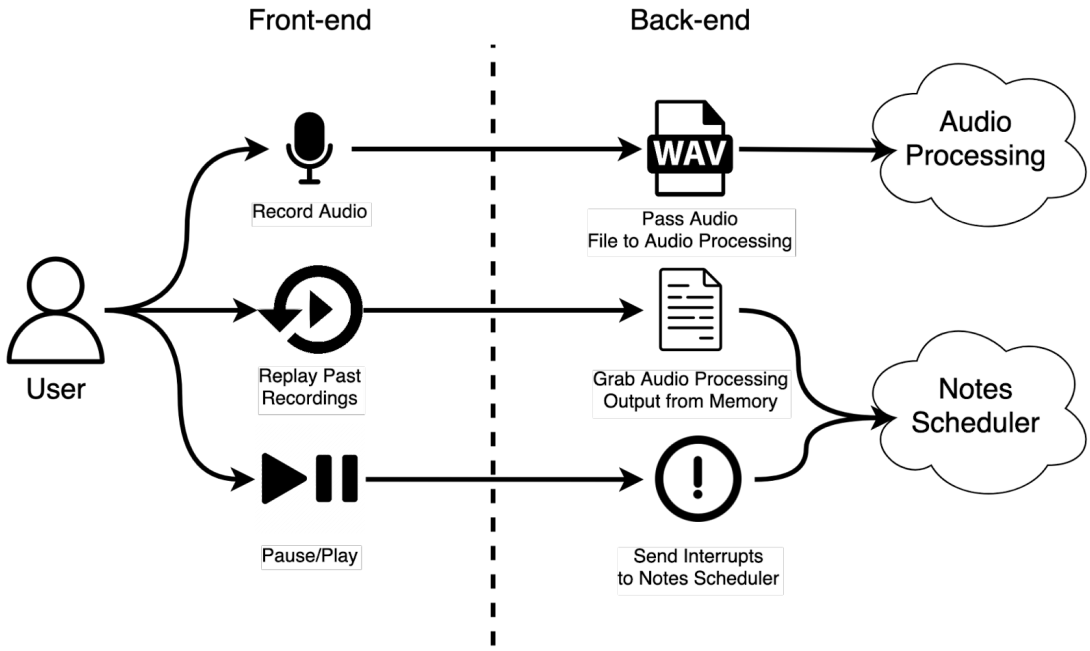

Figure 3. A diagram of the web application interface design and data flow

key press to represent the amplitude differences of each frequency. Upon further research, we found 25N solenoids that were at a good price and used a reasonable amount of power (12V, 1.5A). We will be getting the 25N solenoids as they will be able to press the keys with ease and have much more room for adjusting the speed of the key press such that we can adjust the volume of the key to emulate the amplitude of each needed frequency.

VI. System Implementation

A. Web Application User Interface

Our web application user interface is the subsystem responsible for supporting controls the user can use to interact with our system. [Figure 3] On the web app, users will be able to record audio, upload past recordings, or pause/play the piano playback. We will be hosting our web app with Django, a python-based web framework, on an AWS EC2 instance. Using Django allows us to conveniently integrate our audio processing and note scheduling into our backend since those will be written in python as well. On the front-end user interface, we will be using Bootstrap for a pleasing and simple design and Ajax to support asynchronous calls to our audio processing and notes scheduler functions while maintaining responsive interactions. Audio recordings will be done using built-in Javascript libraries that will save the file in the .wav format, a lossless audio format that stores the audio as amplitude across time. This .wav file will be then sent to the audio processing subsystem. The pause/play functionality will be implemented by sending interrupts to the notes scheduler whenever the user toggles the pause/play of a recording. The interrupts will contain information that causes the notes scheduler to either continue or stop sending information to the Raspberry Pi.

B. Audio Processing

The following implementation details are also illustrated in figure 5. The audio processing module takes in as input a recording of our user's voice.

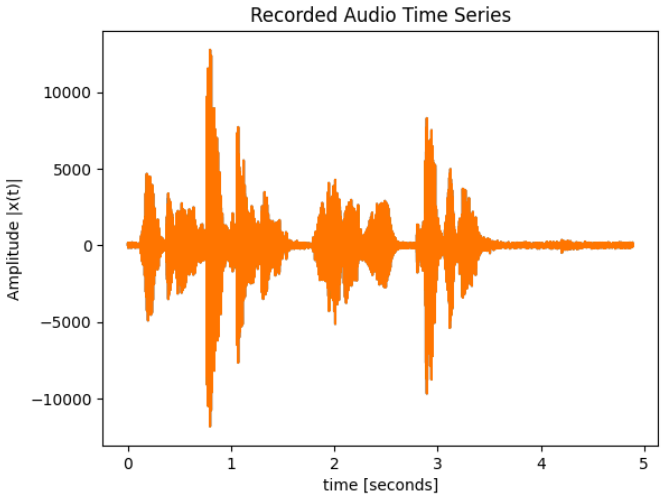

Figure 4. Incoming Audio Plotted as Time Series Data.

The incoming .wav file is converted into an array containing time-series data, as in figure 3. To collect information on how the frequencies of our audio change over time, we divide the incoming array into a series of sample windows containing a collection of samples from the time series array. For more information on how we determined the size of this window, refer to the third section of our design trade studies.
Then, for each sample window array, we calculate the Fast Fourier Transform (FFT) of that window's time series data, which returns an array containing information about the sample's spectral density. This process is repeated for every sample window within the original time series array, and the

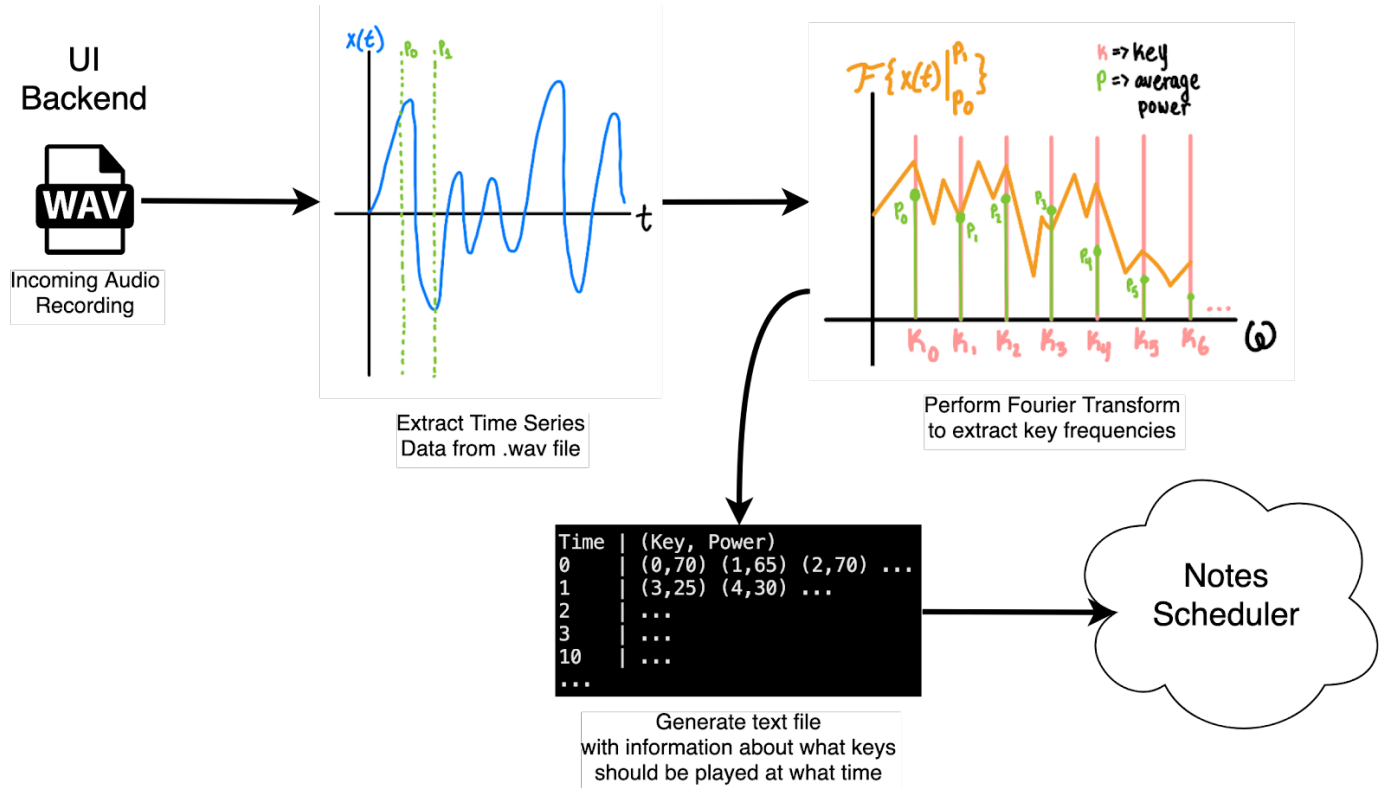

Figure 5. An overview of the audio processing module.

result is an array describing how the frequencies of our original audio recording change over time. This result is illustrated on the spectrogram in figure 5.

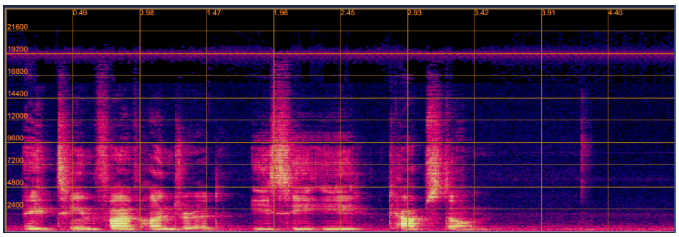

Figure 6. Spectrogram displaying input audio frequencies changing with time. X-axis, time in seconds. Y-axis, power.

Replicating these frequencies throughout time using a piano is a 'lossy' transformation, meaning that it is impossible to reproduce all of the frequencies we're able to extract using only the keys on a piano. Therefore, to propagate this information from our original recording through a piano we average the surrounding frequency magnitudes available to us at the points that correspond to keys on the piano. There are several ways of doing this, all of which effectively blur the incoming audio recording. We use the term blur to reference the notion that averaging the surrounding pixels of every pixel in an image creates a blurred version of the original image. We are still considering the effectiveness of different blurring techniques. For more information, please refer to the fourth section of our design trade studies.

Averaging the incoming frequency information onto the points in the frequency domain that correspond to piano keys, results in an array that describes what piano notes should be played as well as with what strength, across time. This information is condensed into a tab-separated value (.tsv) file, containing information about the moments in time where non-zero (i.e moments where at least one key should be pressed) keys are being played and at what strength. This text file is stored to memory to be used by the note scheduler process.

C. Notes Scheduler

We define the notes scheduler as the module that takes input from the audio processing module and converts it into a format our physical piano interface can use to create the replication of human speech. By definition, this requires the notes scheduler to translate time-indexed arrays of frequencies and their corresponding amplitudes into presses of piano keys at different time intervals with an appropriate level of force. In more specific terms, this module will translate the output of the audio processing system in the form of a text file with tab-separated integers. Each line of the file represents a timestamp, and each column represents a frequency. The integer at line x and column y represent the volume of frequency y at time x.

A consideration is that even though an arbitrary frequency, a, may be active at time t, the piano key does not necessarily need to be pressed at that time. It may have been pressed during a previous timestamp and still producing sound. Therefore, the note scheduler must account for which keys are already sounding while pressing new ones to produce a smooth sound that evokes human speech, as opposed to stuttering, choppy noise. The module will accomplish this task by computing, for each frequency, the difference between amplitudes for the current timestamp compared to the previous timestamp, as well as keeping track of which keys are

currently pressed and for how long. If the frequency is significantly louder from one timestamp to the next, or if it has been long enough such that the sound has already faded, the key will be pressed again. Conversely, if the key is currently pressed but its amplitude is insignificant, the solenoid will lift.

D. Physical Performance Scheduler

For the physical performance interface, we will be implementing a circuit containing a Raspberry Pi as the signal driving unit, 9 serial-in parallel-out (SIPO) shift registers, 69 MOSFETs, and 69 cylindrical 25N solenoids. Data from the note scheduler, in the form of a 69-bit value where the nth bit represents if the nth key will be played or not. This 69-bit value will be split into 9 Bytes, where each byte will be loaded into one of the 8-bit shift registers. We will be using 9 GPIO pins of the Raspberry Pi to serially load the 69 bits into the 9 shift registers in the correct order such that the LSB is the lowest frequency note and the MSB is the highest frequency. As the data is being serially loaded into each shift register, a GPIO pin of the Raspberry Pi drives a shift clock signal to all of the shift registers such that for each bit sent to the shift register, the clock outputs a positive edge signal indicating to the shift registers to intake a bit and shift the values. Once all shift registers are filled with data (8 clock cycles), a latch signal is sent from another GPIO pin telling all the shift registers to output their values in parallel to the 69 MOSFETs. Each MOSFET acts as a transistor controlling the flow of current from our 12V power source to the 25N solenoid.

Some specifics: We need to actuate solenoids and produce a sound with a period of the time window we select for our audio processing since each of the 69-bit values represents the notes comprising a particular time window sample of the audio recording. This means we have to be able to play audio at a frequency of 1/Tw (Tw = window of time). Since we need to load 8 bits of our data into the shift registers within each of those solenoid actuation periods, our bit streams and clock signals need to be 8 times faster than the solenoid actuation frequency. This value is given by 8 * (1/Tw). Lastly, to determine the total power consumption of our system, we must calculate the power of a single solenoid and multiply it by 69 since each solenoid will be wired in parallel. For one 25N solenoid, we will need 12V and a maximum of 1.5 A of current. With this, we will need a 12V supply capable of outputting ~(1.5 * 69) = ~103 A of current. This would require a 1200 Watt power supply since power (Watts) is V*I = ~12*100.

VII. Test, Verification and Validation

As previously stated, our goal for the web app-to-physical system latency is 200ms or less. We will test this by measuring the time between pressing a button on our frontend UI, waiting for the system to interact and output to the UX, and measuring the response time. This will be done with the Selenium browser automation tool.

The fast Fourier transform accuracy will also be tested. We will create input audio with randomly generated frequencies and amplitudes. This input will be fed through the fast Fourier and the output accuracy measured in terms of the number of correct frequencies and corresponding amplitudes. By definition, FFTs are more accurate with a longer chosen window, but this will incur higher latency. We will adjust the latency to achieve an accuracy of 80%.

We also have standards for syllable timing. We wish to record the start times and end times for various syllables when spoken and compare those to the start times and end times when sounded by the piano. We do not need a very stringent requirement for this, as speech with “incorrectly” timed syllables is often still intelligible - immigrants who speak second languages with non-L1 accents can still be understood; British and American anglophones can still understand each other, and many machine-generated voices sound “robotic” due to their timing. However, we still wish to somewhat faithfully replicate the speech of the specific user. Thus, we will aim for a syllable timing fidelity rate of 75%, but this will be more of a goal than a requirement.

Finally, as the overarching goal of this project is to evoke human speech, we wish for our piano to be understood as it speaks. We will test this by surveying groups of volunteers who will listen to prompts given by the piano and try to decipher what it says. We aim for a high success rate of 95% here, as it is the main goal of the project.

VIII. Project Management

A. Schedule

Given that the physical device is our biggest single-point failure, we’re frontloading our efforts into confirming that it is possible to build this device. To do this, we’ve spent the first 3-4 weeks of our project working on a series of “proof of concept” milestones that demonstrate we can control the physical interface in the way we expect to. These milestones, if not met, mean we should pivot towards the virtual interface and use the rest of our allotted time for the physical interface on the virtual alternative. If the proof-of-concept milestones are met, by our fifth week of development we will decide whether or not to pivot. In the case where we do not pivot, each of us has a portion of the physical device that we will contribute to. Marco will build the frame and printed circuit board. John will design and test the circuit. Angela will implement the firmware that controls the solenoids. In the case where we do pivot, the virtual piano becomes an additional feature to the web application, with development led by John and the help of others if needed.

Concurrently, the three of us will lead the work being done in our respective areas. Marco will develop the audio processing, John will implement the web application, and Angela will implement the notes scheduler. Four weeks before the end of the semester, we’ll spend two weeks regrouping and assembling the physical device, integrating our individual workloads, and running tests on the effectiveness of our resulting system. We’ve allocated two weeks of “slack time” for any issues that may come up along the way.

B. Bill of Materials and Budget

Please refer to Table 1 for more information.

IX. Related Work

Player pianos have been around for many years, however only recently have they begun to be built using electromechanical devices. Edelweiss Pianos sells player pianos within the range of $20,000 USD. Mark Rober, a former NASA and Apple engineer, now Youtuber, posted a video introducing his self-talking piano which inspired this project.

X. Summary

Our goal is to create a self-talking piano. In order to achieve this, we'll record a user's voice via a web-app based user interface. Once we have this data, we will extract the frequencies that make up a user's voice across time. We will average these frequencies onto the frequencies that correspond to piano keys and output that data onto a tab separate file. A notes scheduler will parse this file and control a series of solenoids that can actuate the keys on a piano. The result should be series of notes being played on the piano that mimic the sound of a human voice.

Glossary of Acronyms

- Play rate, the rate at which the keys on the piano are being pressed.
- Phoneme, the atomic unit of speech, can be typically represented by a letter in English.

References

[1] World Leaders in Research-Based User Experience. "Response Time Limits: Article by Jakob Nielsen." Nielsen Norman Group, https://www.nngroup.com/articles/response-times-3-important-limits/.

[2]"Piano Key Frequencies." Wikipedia, Wikimedia Foundation, 29 Aug. 2022, https://en.wikipedia.org/wiki/Piano_key_frequencies.

[3]Haskins Laboratories. (n.d.). Alvin M. Liberman, 82, Speech and Reading Scientist. Retrieved December 19, 2011, from http://www.haskins.yale.edu/staff/amlmsk.html

[4]Peelle, Jonathan E., and Matthew H. Davis. "Neural Oscillations Carry Speech Rhythm through to Comprehension." Frontiers in Psychology, vol. 3, 2012, https://doi.org/10.3389/fpsyg.2012.00320.

Table 1

Bill of Materials						
Description	Manufacturer	Model Number	Quantity	Cost [$USD]	Shipping [$USD]	Total [$USD]
25N 12V Solenoids	MannHwa Smart Home Electrical	TAU-1039B	70	$2.52	$161.61	$338.01
Shift Register Kit	BOJACK	SN74HC595N	1	$6.99	$0.00	$6.99
5.6A 100V MOSFET Transistor Kit	BOJACK	IRF510N	1	$7.99	$0.00	$7.99
24V 50A Power Supply	Weiattle Trading Company Ltd. Store	N/A	1	$65.99	$3.01	$69.00
8020 T-Slot Extrusion	80/20	N/A	N/A	N/A	N/A	$50
PLA 3D Printing	N/A	N/A	N/A	N/A	N/A	$50
					Grand Total	$521.99