# accompanyBot

Aden Fiol, Rahul Khandelwal, and Nora Wan

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—For musicians looking to play along with piano harmony, finding piano accompanists can sometimes be a difficult and costly task. While musicians in these situations might be able to turn to cheap MIDI recordings, the digital sound does not match the acoustic quality of a real piano. On the other hand, high-end player pianos are too expensive for the average person's budget. With our accompanyBot, we have created a portable system capable of reading sheet music and playing the piano parts.**

***Index Terms*—microprocessor, MOSFET, Optical Music Recognition, piano, power, Raspberry Pi, robot, solenoid, tempo, time signature, XML.**

## I. Introduction

THIS project seeks to improve the playing experience for musicians who are looking for accompaniment while practicing and performing alone. Musicians and singers often rehearse music in ensembles with a pianist to emulate a performance-like setting. This enables the musicians to harmonize with each other and get a better sense of group dynamics and tempo. However, different individuals naturally operate on different schedules and cannot meet with others every time they practice. In these cases, individuals practicing alone would still desire piano backing to keep time in preparation for group rehearsals. Additionally, soloists who are performing a piece that involves piano accompaniment may not always be able to find a piano accompanist who can practice and play with them for their big performance. Additionally, the cost to hire a professional piano accompanist at the average hourly wage of $38.68/hour [1] may be a barrier for amateur musicians who are just starting out their solo careers.

The accompanyBot provides a comprehensive solution to service these goals. From a simple control interface on the user's computer, users are able to upload a file of sheet music and control our custom hardware that is capable of playing the piano part on a piano. Our portable physical interface that mounts to the piano will allow users to pick up and place accompanyBot on various pianos in different locations. With the overall cost of our entire implementation being only $226.51, the accompanyBot is more affordable than hiring piano accompanists after just 7 hours of use.

While there are existing technologies that users may be able to utilize to meet similar needs, they are not without their downsides. On the low-budget side, users might turn to MIDI recordings to play along to during practice. However, the sound quality of digital recordings cannot match the acoustics of physical pianos. For soloists who wish to perform at recitals and in professional settings, MIDI recordings will not suffice for the formal atmosphere. On the other side of the spectrum, high-end player pianos that have actuators embedded within the piano may have the desired sound quality but are also very expensive, with prices that can reach above $100,000 USD [2]. Moreover, these player pianos are upright and grand pianos. Thus the size and weight of these instruments make it impractical for musicians to move them every time they practice in a different place or perform at different venues.

## II. Use-Case Requirements

There are several use-case requirements that our design must meet to ensure user satisfaction.

### *A. Note Playing Accuracy*

Since the users will be utilizing our system to accompany them in performances, the accompanyBot must play the correct notes. Our note playing accuracy relies on many different factors, such as the sheet music parser, the note scheduler, and the circuitry being built properly. Since each area can introduce some small error that propagates down to the piano, we cannot guarantee complete accuracy. However, integrated together, these three major components of our project must still be able to maintain a degree of high note playing accuracy such that there are no mistakes noticed by the average listener.

### *B. Tempo Variability*

Since the accompanyBot is used for both practice and performance, the user should be able to adjust the tempo of the piano player so that they can start practicing a piece at a slower pace and work their way up for the final performance.

### *C. Tempo Accuracy*

Due to the importance of timing in music, the user will demand high accuracy with respect to the accompanyBot's playing tempo. This allows the musician to keep time with the piece and stay synchronized if their goal is to eventually play along with others. Thus we have set the requirement for the tempo accuracy to be 100% accurate to the tempo specified by the sheet music or that the user inputs. The exception to this requirement is the case where the written tempo exceeds the physical limitations of the solenoid, in which case tempo accuracy will be guaranteed up until the max tempo limit derived from the max switching rate of the solenoids.

### *D. Low Latency*

A pleasant user experience is a priority consideration for the design of our product. Ensuring a fast response between the user interface and the physical system is very important. We have decided to guarantee a response time within 150 ms. According to the NIH, the average human response time to auditory stimuli is around 140-160 ms [3]. Guaranteeing 150 ms will allow for a seamless transition from pressing play to hearing the piano being played. Another area of latency we had to address was the time between music upload and when the system is ready to play. Since the OMR time is variable with the number of pages and the result of the computation can be cached for reuse, this was not a factor in this latency metric. Rather, after the OMR has processed sheet music into XML, we sought to reduce the file transfer time of the XML from the computer running the application to the RPi. Since this transfer is being done via internet packets, we constrained the file latency to the maximum reasonable time a human typically will wait for a webpage to launch. This limit is approximately 2 seconds [4].

## III. Architecture and/or Principle of Operation

In order to operate efficiently and accurately, our system is divided into three distinct subsystems:

1. Local Application
2. Signal Coordination
3. Physical Interface

The block diagram in Fig. 1 depicts the three subsystems and their interconnects.

The Local Application (1) provides the user interface where users insert pdf or png files of high resolution[1] music scores. Additionally, user actions, such as playing or pausing the accompanyBot, modifying the music playing tempo, and jumping to an alternative region or measure of the piece, are received via the local application and sent to the rest of the system. Once users insert their music score, the app is directly responsible for making use of Audiveris, an open source Optical Music Recognition software, to read the notes from the score into an XML file that is then sent downstream through secure file copy over the internet.

The Signal Coordination stage (2) is composed of an Arduino Uno and Raspberry Pi 4. The Arduino acts as a proxy to facilitate USB-to-UART data communication between the GUI and Raspberry Pi. The Pi acts as a responsive piano-playing controller, taking in raw or transformed data from the user and responding by transmitting signals to control the hardware within the physical interface. The necessity for this separation is for a couple of reasons. Firstly, the Audiveris software is best built and run in a Windows environment which is incompatible with Raspberry Pi's native operating system. The RPi 4 is responsible for running the note scheduling algorithm and ON/OFF signal delivery to the hardware solenoids. Note scheduling is conducted via the manipulation of data constructed by the music21 library. Specifically, music21 is used to preprocess the XML received so that the Raspberry Pi can easily extract the note pitches and octaves. Alongside handling music notes, the RPi 4 accepts requests from the local application corresponding to specific dynamic actions the user makes. These actions flow through the main control function to the GPIO pins of the RPi 4.

The electrical signals sent across the GPIO pins are received by 12 transistors from within the physical interface (3). Low pin signals at the gates of the transistors put them in cut-off, while high pin signals above the threshold voltages turn the transistors on. Based on which transistors are on or off, current will flow from the power supply to the corresponding solenoids, thereby inducing a magnetic field

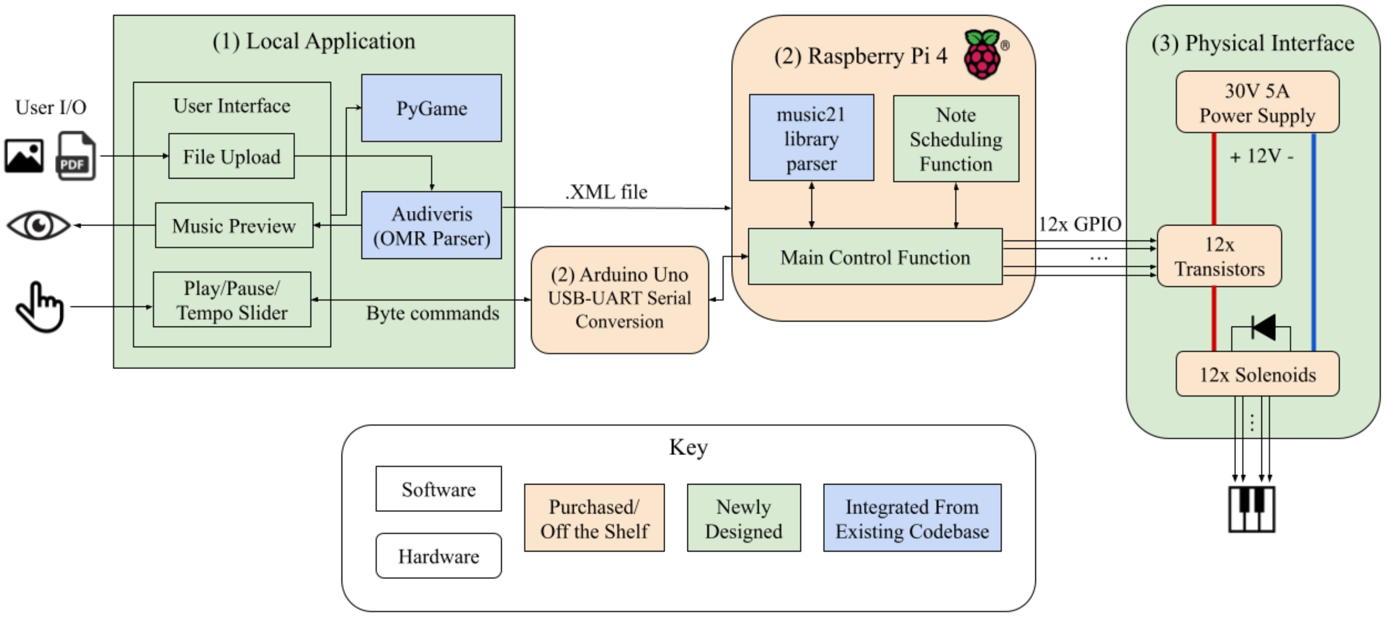


Fig. 1. Block diagram drawing of overall system

that actuates a shaft to press down notes on the piano.

Notable changes to the architecture from our design report include the addition of the Arduino Uno, modification to the XML transmission method, modification to the chassis design, and elimination of one solenoid and transistor from the physical interface. The Arduino Uno was a necessary component as the RPi does not transmit and receive data over USB easily. By connecting the Windows computer to the Arduino over USB and connecting the UART TX and RX pins on the Arduino to the RX and TX pins respectively on the RPi, end to end communication is achievable. While we can transmit small byte messages like start/stop/tempo change commands from the app or measure changes from the RPi in a few milliseconds, it takes much too long to transmit full XML files of songs that are hundreds of kilobytes in size. We decided to transmit the XML instead wirelessly via scp, which drastically reduced the number of seconds taken. Furthermore, we initially planned on having the chassis suspended over the piano keys using two metal rods. Unfortunately there were too many variables to account for, so instead we used the characteristics of the keyboard as support rather than adding our own support beams. This caused us to change the shape of the chassis which was not too much of a hassle fortunately. Lastly, we chose to restrict the number of solenoids to cover only twelve keys in a fixed arrangement over one octave as our keyboard did not have enough space between black keys such that an additional solenoid could fit. This change is the main reason that we had to limit our playing range to one octave that specifically starts on a C.

Fig. 2 below shows the entire system with all the subsystems integrated together.

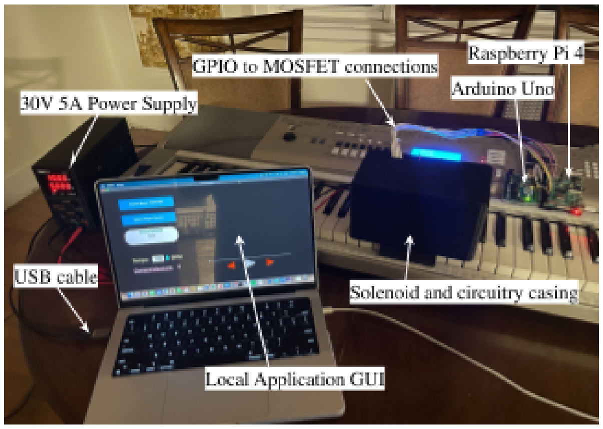


Fig. 2. Overall System

## IV. Design Requirements

### A. *OMR Parser Accuracy*

In order to take advantage of pdf/png inputs, the accompanyBot must be able to parse music scores with high accuracy. We are using OMR software to handle the parsing. Though perfect note parsing would be ideal, there is some room for error as the primary function of accompanyBot is to provide accompaniment. For this reason, our users would find a minimum of 95% accuracy in note parsing to meet their standards for practice or performance. An accurately parsed note is defined as a note whose parsed pitch and duration match the original note.

### B. *Note Scheduling Accuracy*

Once the OMR parser accurately translates the sheet music to an XML file, the note scheduling algorithm should convert that into correct and properly timed signals that are sent to the physical circuitry. This is imperative for the validity of our system since music is all about timing and pitch correctness. Without a high accuracy of note scheduling, the users of our product would never be able to practice and perform with our accompanyBot. Since note scheduling accuracy is of significant importance we are setting this requirement to be 100% accurate in tempo and rhythm. Furthermore, this requirement assumes that the tempo and shortest length note fit within the constraints of our system. This requirement is relative to the output of the OMR parser since the note scheduling algorithm has no control over the accuracy of the upstream parsing.

### C. *Power Consumption*

Larger chords and simultaneous note presses will demand more electrical work to induce the solenoid motions. Since each of the key pressing solenoids will draw a non-negligible amount of power, we must maintain our system to be safe for the average individual to attach to their piano or keyboard. We found that the average electronic keyboard can draw up to 60 watts of power [5]. For this reason, the hardware component of the robot has a quantitative requirement to never consume more than 60 watts of power. This will ensure users have a safe playing experience while also not overcharging their power bill.

From the data sheets of our solenoids, an activated solenoid requires a max of 12V and 1A. Based on measurements we gathered from initial testing, we were able to operate the solenoid with 10V and 0.65A. However, to provide a buffer, we will use the maximum values for power calculations. Thus the power required to activate one solenoid is $P = IV = (12V)(1A) = 12W$. Therefore our system only supports five or fewer solenoids pressed at once.

In cases where the number of notes played at one time exceeds five, which will cause the power to exceed the safety threshold, the additional notes simultaneously scheduled to play should be omitted from playing. This restriction will be enforced by the scheduling algorithm that decides which solenoids to activate. Additionally, the power supply we are using only outputs a maximum current of 5A, so by ensuring that the voltage input is only 10V, we can keep our maximum power below the threshold.

## V. Design Trade Studies

### *A. OMR Framework Selection*

We opted for Audiveris as opposed to other OMR frameworks primarily due to its high accuracy of identifiable notes, ability to segment multiple parts as opposed to other frameworks available, and capacity to recognize and parse polyphonic music. Initially, we had considered using an openCV and tensorflow based ML model built for Python. This was so that our UI hub application, which has also been built with Python, could directly process the music scores without needing to contact an external application. Early on we tested a pre-trained model developed by researchers in Montreal, Alicante, and Valencia [6]. Overall parsing time for a single page was no more than 10 seconds. However, the model was built to process strictly monophonic scores, which did not completely satisfy our use case. Another Python based solution we tested was Oemer [7], which could be easily installed and used via pip and import statements. Oemer met our requirement of processing polyphonic scores but failed to meet our note accuracy and processing time requirements. Initial testing yielded close to 50% of correct notes placed in the XML file structure, and also the time to parse a single page of notes from two staves exceeded a minute.

Eventually we settled for the Audiveris OMR toolchain. Despite having a complicated build process, Audiveris proved to be the best open source solution for our project. Testing its OMR engine on multiple page scores yielded an average time of 20 seconds per page, not unreasonable for a user to wait the one time cost. Additionally, the outputs it produced had seamless integration as input to the music21 Python library, a framework that was essential for us to schedule notes on the hardware. The traditional usage of Audiveris is through its own GUI. However, since we will build out a separate GUI for the accompanyBot user interface, we are making use of the developer command line usage of Audiveris as referenced in its documentation [8]. To further confirm the effectiveness of Audiveris, we played back the MIDI equivalent of XML generated from Chopin and Scott Joplin scores. Aside from the digitized sound of the player and a few odd notes over the course of a ~3 minute piece, the audio matched the written notes.

The last option for us was to build an OMR engine from scratch. Manually crafting an ML model or framework would have gone out of the scope of our goals. From our understanding, many researchers have spent years developing and improving the models for the machine learning and optical classification of notes. They have not yet reached a complete solution, though a good approximation is achievable. Overall, integrating Audiveris allowed us to give better attention to the other subsystems of the accompanyBot.

### *B. XML File Transfer Method*

We initially planned on using the serial communication channel between the computer and RPi to also send the parsed XML files. However, after initial testing of sending an XML file over UART, we measured a 13 second latency for sending a one-page 36 KB file. This would scale linearly with the number of pages in the sheet music. We decided that this delay was not sufficient for our low latency use case requirement. Thus we switched to sending the XML files via secure copy over the network, which resulted in file transfer taking less than 2 seconds on average. The downside of this approach is that the RPi must be connected to the same network as the computer running the local application. Ultimately we decided it was worth switching to scp since the desire for a fast response every time a new file is uploaded is worth the one time cost of setting up the wireless connection.

### *C. Scheduling Microcontroller Selection*

When choosing a specific microcontroller to act as the note scheduler and mediator between the software and physical circuit of our project, we prioritized two characteristics: compute power, and ease of integration with other parts of the project. We originally considered a high number of GPIO pins to be a criteria as well, although our choice to limit the project's scope to just one octave made it so that the number of pins available was no longer a limiting factor. The three options we considered using were the Arduino Uno, STM32F4, and RPi 4. The former two candidates were considered because members of the group had used them for previous projects and classes, while the latter was considered due to its solid reputation for being a reliable general-use microprocessor.

Due to the need to parse the XML file into usable data and schedule signals at specific times, we required a component that had significant computing power and memory available to store the data. This led us to choose a microprocessor over a microcontroller like the Arduino Uno that is more specialized for specific tasks, so we ruled out the Arduino Uno as the agent to run our scheduling algorithm, although later on we still made use of its UART capability.

The second criterion was crucial in deciding which microprocessor to use for the accompanyBot. Since we were developing the user interface in Python, we felt that having the coding environment on the microprocessor also be in Python would make it easier to handle the communication between them through libraries such as pySerial, which we also have experience with. In addition to this, we found through our initial search that there were many Python packages relating to analyzing music, so we could fall back on them when performing the XML parsing and note scheduling (This is precisely what we did in the end with our choice to utilize the music21 library). Thus, for the reasons specified above, we chose to use an RPi 4 from the class inventory.

### *D. Solenoid Selection*

There is a large variety of solenoids on the market. They vary in terms of force, size, type, and electrical requirements. We decided to purchase three different solenoids that varied across each category: a 5N Adafruit Small Push-Pull solenoid, a 25N Adafruit Large Push-Pull, and a 25N Pull Type Open Frame Solenoid Electromagnet Linear Motion JF-1039.

During our testing of the three solenoids, we noticed major downfalls for two of the solenoids which resulted in us choosing the 25N Adafruit Large Push-Pull solenoid. The 5N Adafruit Small Push-Pull solenoid struggled in three categories: stroke length, force output, and power consumption per unit of force. Our project needed solenoids that would be able to reach the keys from a reasonable distance away and with enough force to depress the keys while also not consuming too much power. Constructing a mechanism that would hold the weight of all the solenoids and circuitry close enough to the keyboard without falling over would be difficult. Thus, a stroke length that is too small would be difficult to design around. Additionally, from our initial testing, 5N was not enough force to depress an average piano key. Furthermore, it consumed the same amount of power as the large push-pull solenoid, but the power consumption per unit of force was much higher, so for those reasons, the 5N Adafruit solenoid would not work within the constraints of our system. The 25N Pull Type Open Frame Solenoid was lacking in one major area: the fact that it was a pull type rather than a push-pull solenoid. When researching solenoids we were not aware that the pull type solenoid would not be fixed in place. When we received the part, we discovered that the ferromagnetic rod for depressing a piano key was free to fall out of its exterior metal casing. This was not what we desired. Instead, we preferred the 25N Adafruit solenoid since its ferromagnetic rod was fixed and did not fall out of the external metal casing.

Overall, the design shortcomings of two out of the three solenoids led us to conclude that the 25N Adafruit Push-Pull solenoid was the best option for our accompanyBot. We are able to get a lot of force per watt and a high stroke length without the internal rod falling out of the external metal casing, making it the logical option to use. The final factor that solidified our decision was the price point of the 25N Adafruit Large Push-Pull solenoid. Since Adafruit offered a discounted per unit price when buying in bulk, the price of each solenoid was cheaper than other similar solenoids on the market.

## VI. System Implementation

Our system can be broken down into three domains: a translation path, notes scheduling path, and execution path depicted in Fig. 2. Subsections *A* and *B* make up the translation path, *C* describes the intermediate path between translation and notes scheduling, *D* goes into depth regarding the RPi operation for scheduling, and *E* lays out our setup and connection of the moving parts and electronics built in-house.

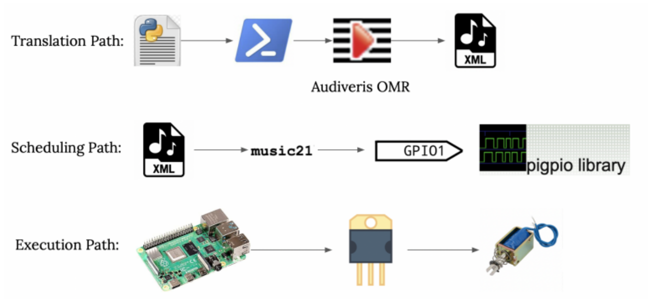


Fig. 3. Visualization of subsystem flows

### A. *Computer GUI Interface*

The user will have the ability to control the accompanyBot over the GUI software on their computer. The necessity for ease of use and visual appeal is important here, so we used the pyGame library in Python to build out this application interface. Fig. 3a illustrates the early model of our application's GUI, while Fig. 3b shows one view of the completed implementation that showcases the different features of the app.

The user is able to choose their input sheet music file and output source through buttons labeled "Import Music PDF/PNG" and "Connect to Player Device" respectively. The user can manually specify the beats per minute in the tempo window of the application. Additionally, a play button and slider allow the user to direct the RPi to start/stop playing or change measures respectively. For convenience, a current measure playing slot shows what measure the notes scheduler is currently on.

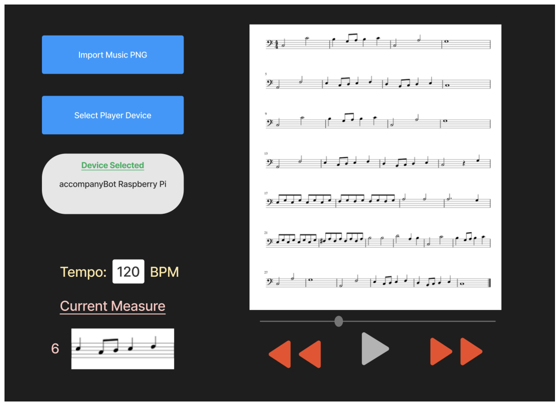
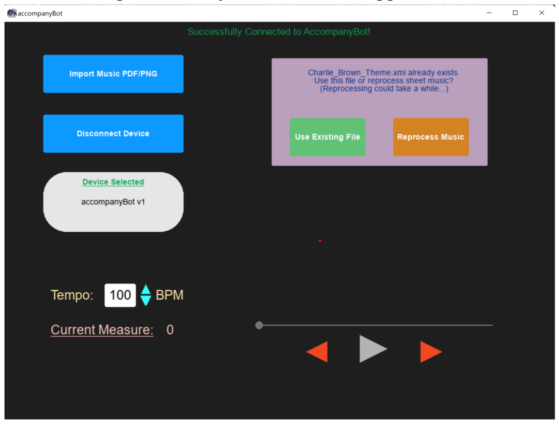


Fig. 4a. Early Model of GUI Application


accompanyBot

Successfully Connected to AccompanyBot!

Charlie_Brown_Theme.xml already exists.
Use this file or reprocess sheet music?
(Reprocessing could take a while...)

Fig. 4b. AccompanyBot Python Application

After building out the sketched design, some noticeable changes are apparent. Firstly, the sheet music of the score is still displayable within the app. The difference now is that before displaying the score, the application checks its local cache to see if it has already processed a score. If so the user is prompted with the purple message block in Fig. 3b. Additionally, the initial design did not incorporate buttons to increase the tempo, relying on just "up" and "down" keypresses to modulate the tempo. The finished app makes accessibility a priority and provides two sky blue tempo buttons in supplement of keypresses. Lastly, some display changes were made to the change measure buttons, current measure indicator, connection button, and the incorporation of a fading out alert message.

### *B. Music Score Reader*

When a music score is imported into the hub application, if a corresponding music XML transcription is not found within the application cache then the software begins a subprocess to execute the Audiveris OMR engine. Due to the limitations of Audiveris dependencies, it must be run only on Windows. Additionally, an install of Java 17 is required to enable Audiveris. For this reason, the hub application executes a PowerShell script to launch Audiveris and process the music score image into an XML file. This XML file then gets saved to a desired output location and also gets added to the application cache for repeat usage efficiency.

### *C. Computer to Microprocessor Communication*

We will be handling the communication between our laptop and the RPi 4 through a serial USB connection with an Arduino piping the communication stream between devices. We believed this was the simplest method of implementation for communication since the user can directly connect the finished product to their computer with a single cable. The connection between the Arduino and RPi will already come pre wired so that no additional setup is necessary there. We have utilized the pySerial library for transmitting the files and command bytes that will let the RPi 4 know to prepare for a file, start playing, stop playing, or change the tempo. In the event there is a disconnection between the computer and Arduino, the simplicity of the pySerial library enables a simple reconnection of the wire for the system to continue operating as intended.

Before building out this serial solution, we considered the possibility of the latency for the XML file transfer being too high. This became a reality after some initial tests, so we had to switch to a wireless SSH protocol. We opted to scp the file from the computer over to the RPi, bypassing the Arduino Uno in the middle. This was effective as both Windows and Linux support this methodology. Traditional scp requires the user to enter the login password of the receiving device. This would become cumbersome and unintuitive for our ideal user. We therefore had to ensure that the public and private RSA keys were appropriately distributed between subsystems to eliminate the password stage. Since scp simply copies the XML file into a directory on the Raspberry Pi, we still use the serial line to communicate the name of the file to be played, since the program needs to know when to switch pieces or when a new file has been sent by the user.

### *D. Note Scheduling*

The first step of the note scheduling process involves parsing the XML file to extract the key signature, time signature, tempo, and individual notes. To aid in the scheduling task, we have employed the Python library music21 to convert the XML file to a data structure that is more readable and allows for easy access to the musical data. Through the `parse` method of the library, we can extract the essential information needed for scheduling: the clef, tempo, key signature, and an iterable array of notes. Additionally, the note object includes information about their pitches, durations, and octave. Fig. 5 illustrates an example of the type of output achievable by using music21. When the snippet of sheet music shown in Fig. 4 is converted to its XML representation, the entire file consists of 184 lines of code. However, after a simple parsing with music21, all the necessary information is condensed into a 20 line Stream object. An important detail to note is that the offset, which is the number in curly braces, indicates the distance in quarter notes that a note or musical feature is located from the beginning of the Stream. The offsets are used by our scheduling algorithm to determine where in the piece a note is played.

The general approach to scheduling is to convert the array of notes in a piece into a data structure that holds information about which notes to press down or lift up at each specific measure and where in the measure that note action occurs. Specifically, this data structure is a dictionary with measure number as the key and dictionaries as the value. The keys of the nested dictionary are the offset in the measure while the values are a set of notes to play at that specific offset in the measure. When the scheduler iterates through the array of notes mentioned above, it extracts the note's measure and computes its fractional offset in the measure using the offset. It then creates a note object containing the pitch, octave, and state (for if the note's corresponding GPIO pin should be on or off at that time instance) and places it into the appropriate set.

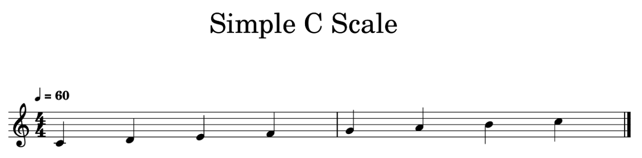
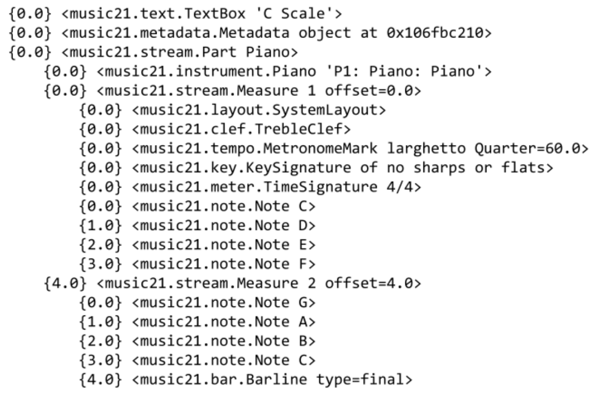


Fig. 4. Quarter note C scale in 4/4 time signature at 60 BPM



```
{0.0} <music21.text.TextBox 'C Scale'>
{0.0} <music21.metadata.Metadata object at 0x106fbc210>
{0.0} <music21.stream.Part Piano>
    {0.0} <music21.instrument.Piano 'P1: Piano: Piano'>
    {0.0} <music21.stream.Measure 1 offset=0.0>
        {0.0} <music21.layout.SystemLayout>
        {0.0} <music21.clef.TrebleClef>
        {0.0} <music21.tempo.MetronomeMark larghetto Quarter=60.0>
        {0.0} <music21.key.KeySignature of no sharps or flats>
        {0.0} <music21.meter.TimeSignature 4/4>
        {0.0} <music21.note.Note C>
        {1.0} <music21.note.Note D>
        {2.0} <music21.note.Note E>
        {3.0} <music21.note.Note F>
    {4.0} <music21.stream.Measure 2 offset=4.0>
        {0.0} <music21.note.Note G>
        {1.0} <music21.note.Note A>
        {2.0} <music21.note.Note B>
        {3.0} <music21.note.Note C>
        {4.0} <music21.bar.Barline type=final>
```

Fig. 5. Example of the compact data structure that is produced from parsing the C scale below using music21

In a loop in the main control function, when a piece is playing, the RPi computes the current measure and the offset within the measure based on the startTime and startMeasure variables. To change the location in the piece that the piano player starts playing at, the local application sends a byte command that indicates the new measure number. The RPi reads this command, updates the starting measure and start time, and then continues executing the loop. Based on which measure is computed, the RPi reads the data structure to see if there are any notes at the specific measure and offset. If it finds a set of playable notes, the control function will output the correct digital output of the GPIO pins based on the state of the notes in the set. During this process, if more than five notes are scheduled, the function will only turn on five GPIO pins and prevent additional ones from turning on.

The main timing parameter that is relevant to the scheduler is the duration of one measure. Equation (1) shows how to calculate the duration in nanoseconds of one measure. This is computed in nanoseconds because the RPi control function uses the system time in nanoseconds for the highest degree of precision. The beats per measure value is found from the beatCount attribute of the time signature parsed by music21.

$$duration = \frac{60{,}000{,}000\ ns/minute}{Tempo\ (beats/minute)} \times beats/measure \quad (1)$$

With this process, the scheduling algorithm only needs to run once to create the data structure. To change the speed of the playback, the RPi simply recomputes the duration of a measure with a new tempo value using the formula mentioned earlier.

We found the physical switching limitation of the solenoid to be about four times per second, which corresponds to 125ms to depress the solenoid and 125ms to retract it. As a result, notes cannot have a duration smaller than 125 ms. This places a cap on the max tempo our piano player can play at since two successive presses of the same key cannot be scheduled in a smaller interval than this time interval. Thus during the note scheduling process, we find the value of the smallest note and calculate the max tempo using Equation (2). From this, we can determine whether or not the piece is playable at the specified tempo. If it is not, we communicate the calculated max tempo back to the local application so that users cannot input anything higher than the max. In the case where the sheet music's tempo is higher than the max, the piano player defaults to the calculated maximum.

$$Tempo = \frac{60{,}000{,}000\ ns/minute}{smallest\ note\ \ duration} \times beats/smallest\ note \quad (2)$$

## E. Physical Components

The physical circuitry consists of multiple components, such as GPIO inputs, an array of MOSFETs, solenoids,

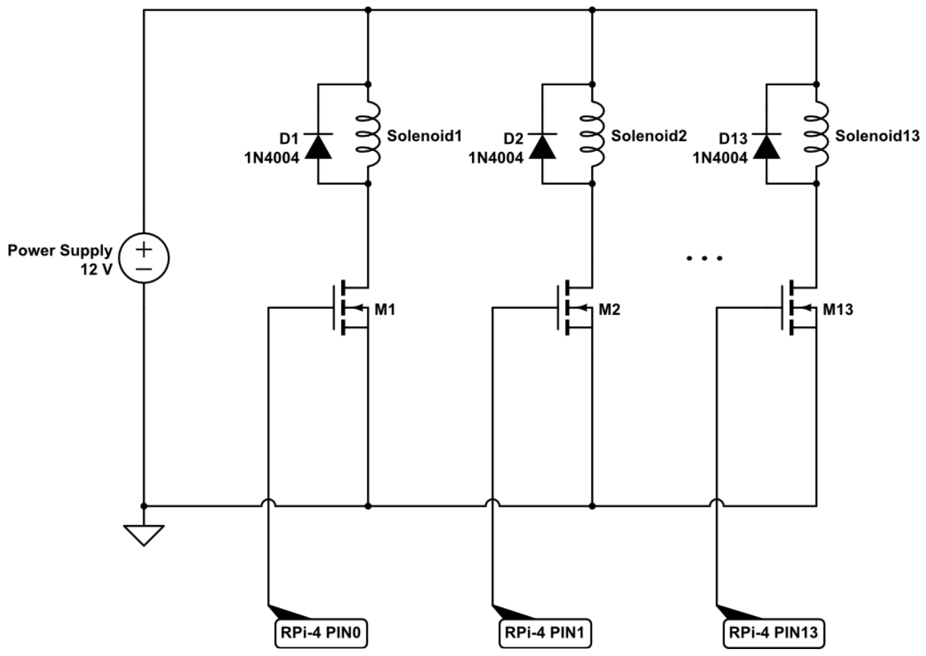


Fig. 6. Schematic of Physical Circuitry

diodes, and a power supply. Each component provides an essential role in making the system able to move and play the piano correctly.

The power source provides the voltage and current needed to actuate the solenoids. The power source has a voltage rating of 30V and a current rating of 5A. With each solenoid drawing at most 1A at 12V, this should be all that we need to power the system since no more than 5 keys will be pressed at a single time.

The output GPIO pins from the RPi 4 are connected to the gates of our MOSFETs. The gates will receive a high or low signal depending on the output of the scheduling algorithm. When high, this signal will force the MOSFET out of cut-off and allow current to flow from the source to the drain and through the solenoid since they are in series. This will power the solenoid and actuate the piano keys when properly scheduled.

The power source will be supplying the current for the 25N Adafruit Large Push-Pull solenoids when the MOSFETs are not in cut-off. The solenoids are large inductors that have a metal rod fixed inside of the coil. Once current is flowing through the inductor, it induces a magnetic field that causes the metal rod to move. The solenoid retracts once the current is cut from the inductor, hence the push-pull name. These solenoids will be pushing on the piano keys depending on the signals it receives from the microprocessor after the notes have been scheduled.

The last major component of the circuit are the 1N4004 diodes. The diodes are placed in parallel with the solenoids to prevent flyback voltage spike when the MOSFETs are turned off and the current supplied to the inductor is cut off. These diodes are needed so that we do not damage our power supply or microprocessor.

Lastly, we have constructed a chassis to house the solenoid controlling circuit as well as the solenoids. The chassis suspends the solenoids above the correct piano keys and once a solenoids' respective MOSFET turns on, the piano key is played. The chassis was designed using Solidworks and constructed out of four parts: the base where the solenoids sit, the walls that encase the solenoid and protoboard, a support beam to hold the chassis over the piano keys, and a top cover that closes the chassis. Additionally, the keyboard has been modified with some adhesive velcro which sticks to velcro strips placed on the outside of the chassis in order to stabilize it while the solenoids are moving. All of these components and modifications help successfully play the piano.

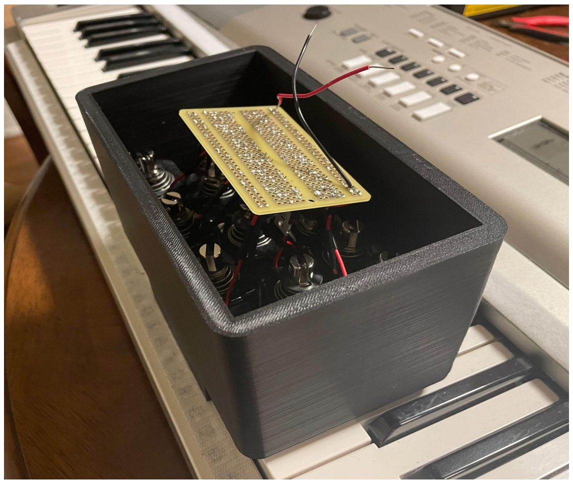


Fig. 7. Chassis Implementation

## VII. Test, Verification and Validation

Table I below shows a compilation of the test results for our design requirements and quantitative use case requirements.

TABLE I. QUANTITATIVE TEST RESULTS

| **Quantitative Metric** | **System Performance** |
|---|---|
| > 95% OMR accuracy of note pitches and note values for 400+ DPI music scores | ~99% accuracy for easy to moderate playing difficulty scores of 240 DPI |
| 100% tempo accuracy | 100% accurate between 30 BPM-150 BPM |
| < 150 ms latency between pressing the start/stop button and hearing a response | Round trip communication latency of ~70ms |
| < 60W average power consumption | Max power of 46.8W |

Table II below provides a compilation of the results for the more qualitative use case requirements we had.

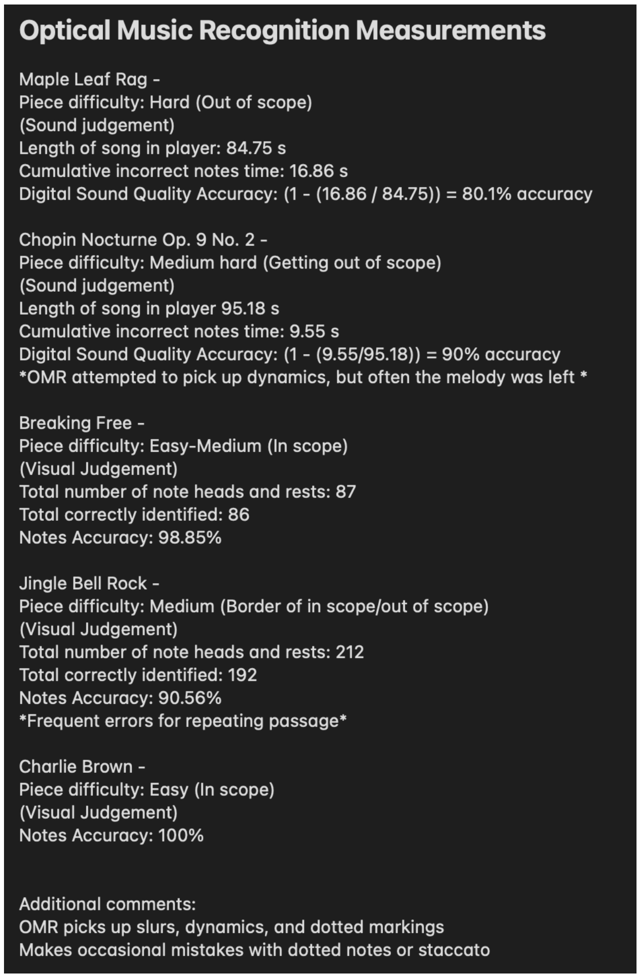
TABLE II. QUALITATIVE TEST RESULTS

| **Qualitative Metric** | **System Performance** |
|---|---|
| Note Playing Accuracy | Pieces played on the accompanyBot sound the same as the MIDI file to our ears |
| Tempo Variability | User is able to change the piano player tempo from the app |
| Latency | No delay noticeable to our ears when starting and stopping a piece |

### *A. Results for OMR Parser Accuracy*

Our use case was to process relatively simple to moderately complex scores of piano music. For those, we desired at least 95% of accurate note values and lengths. Still we ran tests on all types of sheet music to determine results.

For each score tested, after the OMR Parser completed and produced an XML file, we used the music21 library to convert the XML file back to a pdf of the sheet music or MIDI file and compare the original sheet music to the newly generated sheet music/audio. Counting the number of correctly parsed notes and features and dividing by the total number of notes and features will show how much of the music matches the original, determining the OMR parser accuracy. For exceedingly large and complex scores like the aforementioned Chopin piece, accuracy was determined by taking the ratio of time where the song sounded correct after playback through MIDI. Scores with a countable amount of notes were manually verified for accuracy. The following Fig. 9 details the OMR testing results.



**Optical Music Recognition Measurements**

Maple Leaf Rag -
Piece difficulty: Hard (Out of scope)
(Sound judgement)
Length of song in player: 84.75 s
Cumulative incorrect notes time: 16.86 s
Digital Sound Quality Accuracy: (1 - (16.86 / 84.75)) = 80.1% accuracy

Chopin Nocturne Op. 9 No. 2 -
Piece difficulty: Medium hard (Getting out of scope)
(Sound judgement)
Length of song in player 95.18 s
Cumulative incorrect notes time: 9.55 s
Digital Sound Quality Accuracy: (1 - (9.55/95.18)) = 90% accuracy
*OMR attempted to pick up dynamics, but often the melody was left *

Breaking Free -
Piece difficulty: Easy-Medium (In scope)
(Visual Judgement)
Total number of note heads and rests: 87
Total correctly identified: 86
Notes Accuracy: 98.85%

Jingle Bell Rock -
Piece difficulty: Medium (Border of in scope/out of scope)
(Visual Judgement)
Total number of note heads and rests: 212
Total correctly identified: 192
Notes Accuracy: 90.56%
*Frequent errors for repeating passage*

Charlie Brown -
Piece difficulty: Easy (In scope)
(Visual Judgement)
Notes Accuracy: 100%

Additional comments:
OMR picks up slurs, dynamics, and dotted markings
Makes occasional mistakes with dotted notes or staccato

Fig. 8. OMR Testing Results

### *B. Results for Note Scheduling Accuracy*

In order to test and measure the note scheduling accuracy, we tested the tempo played and rhythm of the notes played by our solenoids. Since the solenoids are aligned directly above the correct keys and are turned on based on the notes parsed, we believed we did not need to test that the correct pitches of notes were pressed since the correctness is based on how accurate the XML file is parsed by the music21 library which was out of our control.

#### *1) Results for Tempo Accuracy*

In terms of testing the tempo accuracy, we input sheet music with four quarter notes per measure and varied the tempo with serial commands. We then measured the tempo using a metronome app and compared the measured tempo with our input. From this testing method, we found that the tempo matched the input from 30 BPM to 150 BPM.

#### *2) Results for Note Playing Accuracy*

For evaluating the rhythm accuracy, we qualitatively compared the sound of the notes played on the piano with the output of a MIDI file generated by the parsed XML file for a sample piece (the Charlie Brown Theme). We chose this method rather than comparing the piano player to a human-played piece because this mechanism of testing isolates small errors that may occur earlier on through the OMR parsing, as both the digital audio and notes played by the accompanyBot must rely on the transcribed XML. From this test, we found that the two audio streams sounded similar in rhythm, so we determined that this was sufficient for our use case requirement of not having any mistakes noticeable to the average listener.

### *C. Results for Power Consumption*

After constructing the accompanyBot circuitry, we tested the maximum power consumption in the worst case by turning on five of the MOSFETs such that current is being drawn through all five solenoids. We then measured the voltage output by the power supply and the total current through the MOSFET channels and solenoids. The result was 12V supplied with about 0.81A going to each solenoid. This netted a maximum power of 48.6 watts. Since our maximum power is 48.6W, our average power is guaranteed to be below 60W. Thus we met our design requirement of maintaining average power less than 60W.

### *D. Results for Latency*

The latency was measured as the time taken from when the longest user action signal is sent from the hub application to the microcontroller plus the time until the end of the longest response is received back by the hub application from the RPi microcontroller. After we built out the communication segment of the accompanyBot, we determined which frequent signal(s) had the longest byte sequences. These typically did not exceed more than 4 bytes in length. Then, we wrote test code using timing functions for python. We observed the longest round trip communication time to not exceed 72 milliseconds.

## VIII. Project Management

### *A. Schedule*

The Gantt chart in Fig. 10 below lays out different deliverables from each of us that we accomplished each week of the semester. We also have assignment due dates on the last few rows of the Gantt chart.

We removed the 3 weeks of slack and extended some of our deliverable timelines throughout those slack weeks. Prototyping and getting the design right for the chassis was a major cause of this change. Additionally, integrating everything into a cohesive product needed more time than initially planned. Otherwise, our schedule stayed on course for most of the semester.

### *B. Team Member Responsibilities*

Our team responsibilities were divided into the three major components of our project: the OMR integration and application development, the microprocessor interface and note scheduling algorithm, and the physical circuit implementation. Rahul was responsible for OMR integration and application development. Nora was responsible for the microprocessor and note scheduling algorithm. Aden was responsible for the physical circuit implementation. This division of labor remained consistent with what we laid out in our design review.

Naturally, these three sections of our system relied upon each other for different input and output information, so we worked together throughout the project on problems where our sections intersected. For example, we all relied upon the note scheduling and microprocessor section of the project, so we spent significant time working together on the integration of the RPi to the computer and the RPi to the physical circuit. Additionally, Aden's and Nora's areas of expertise intersect greatly, so they have both made contributions to one another's sections of the project in order to ease the workload and guarantee a working implementation. Rahul has also helped Nora find Python libraries that aid in parsing XML files and storing information in different OOP structures. Rahul and Nora were also responsible for agreeing upon a common protocol for the serial communication.

### *C. Bill of Materials and Budget*

All in all, we have spent $455.59 of our total budget. Of the money we spent, $226.51 is needed for our final implementation. A detailed breakdown of the bill of materials for the final product can be found in Table 1 at the end of this document. The grand total in the table does not include money spent on the test solenoids and spare parts we purchased in addition to what is strictly necessary.

While we initially purchased 15 solenoids so that we could play any octave with any starting key, our finished project only makes use of 12 to play an octave range starting on a C. The remaining three solenoids along with two additional solenoids we bought are now kept as spare parts in case issues arise during our demo. We also did not realize an Arduino or some form of UART would be needed in the design report. Fortunately, the Arduino Uno was of no additional charge to incorporate.

Some of the new costs incurred since the design report include a USB-C to USB-A 2.0 cable to power the RPi and filament for 3D printing the physical solenoid enclosure. We also bought velcro strips to secure the case to the piano. The overall construction costs were estimated to not exceed $200 in total at the design stage. We stayed within our estimate by purchasing only $18.99 worth of filament and avoiding having to use additional materials.

### *D. Risk Management*

One main risk involved in our project is the high degree of accuracy needed in terms of meeting timing requirements. Since some music may be rhythmically complex and very fast-paced, it was difficult to capture those melodies with our system. To mitigate this risk, we limited the scope of possible user inputs to a library of sheet music that does not put strain on the accompanyBot. During our demo, we will still allow users to select different songs to upload, but they will all be chosen ahead of time by us.

While planning out the build process, we acknowledged that physical parts are fragile and often break when constructing something new. Therefore, we purchased additional solenoids and MOSFETs to replace anything that could have broken when building and integrating the whole system. Even with these precautions, after essentially completing our product, the transmission pin of the RPi had stopped functioning, which required us to acquire a new device from the receiving desk. Our design was very modular however, so after the delay of having to switch out the RPi, integrating it only took a few minutes. We also kept additional room in our budget in case more expensive regions of the product were to fail.

Once we determined our actuating solution with the solenoids purchased, we realized that extending the design to cover the range of the entire piano would go over our budget. The only way to mitigate this risk was to limit our design to play one octave of keys. In terms of moving this project forward, we could extend the design at an increased financial cost.

For each of the subsystems that we worked on, there were many risks taken by having to learn new technologies. This included learning Powershell scripting, Solidworks, various python libraries, and the Raspberry Pi setup. We made sure we allocated extra buffer time into our schedules to permit these stages of initially slower development as well as to overcome the challenges that would come with integration. As referenced earlier, only through integration did we discover the necessity of an Arduino Uno to foster the communication between the application and RPi. This brought about another potential risk of the latency time increasing beyond our desired threshold. With sufficient testing, we determined that the increased latency was still within bounds of our requirements.

In general, all of us have stayed on schedule and have been able to make consistent weekly progress. Since the design report we overcame the major risks of not having any working subsystems by completing the local application with a built-in music reader, the notes scheduler and signal transmission modes, the overall circuitry, and casing for the accompanyBot.

## IX. Ethical Issues

One ethical issue related to our product is regarding public health. The accompanyBot is using a GUI to handle user input and directions to the robot. Depending on a user's eyesight, a user could be photosensitive to their computer screen when launching the application. Usually music scores are mainly white, so the mostly bright white pixels that make up the music preview could potentially disorient the individual viewing the app with their system at the maximum brightness setting. We attempted to mitigate this adverse impact by making the GUI application's appearance a dark theme.

We also recognize that this project may require a degree of hand control from the user. Individuals who may be capable of singing but lack motor control in their hands will have difficulty operating the accompanyBot. From a social perspective this would ostracize these individuals or any musicians without hands. Fortunately, all commands that will be executable from the application can be producible via solely mouse drags and mouse clicks. So, if individuals were to use an eye tracking module for mouse control they would be able to regain good control of the accompanyBot.

Considering public welfare, one potential impact for broader social welfare is the concern that the product would replace the need for professional piano accompanists, which might contribute to their unemployment if taken to an extreme. To prevent this possible outcome, we have focused on providing a light-weight solution mainly aimed towards musicians who are on the more amateur side of the spectrum and would not play solo recitals with hired professionals immediately after starting out. Thus once they have practiced with the accompanyBot and honed their skills enough, they would still require professional accompaniment in their musical careers. Since the accompanyBot currently only plays one octave and does not support variable dynamics, there is still a need for professional accompanists who are more capable of highlighting the musicians since they themselves are musicians and can have a better understanding of the music compared to the accompanyBot.

## X. Related Work

We were curious about related projects that make use of FPGAs. For MIT's digital design course, undergraduate students developed their own accompanyBot-like product that processes audio from .wav files and plays matching notes on a keyboard from the output of an FPGA [9]. Similar to us, they made use of Python to gather notes into organized data structures but instead designed an FSM on an FPGA to control motors connected to beams that would rotate and strike the keys, whereas we opted for a Raspberry Pi that would induce solenoids to strike the keys via a linear motion.

Our inspiration to use solenoids controlled by a Raspberry Pi originates from a working keyboard player published in The MagPi, the official Raspberry Pi magazine [10]. Their project successfully played monotonic melody lines on a keyboard. We of course sought to improve upon this project with the power of music21 and our in-built notes scheduler, which made the accompanyBot capable of playing polyphonic melodies and harmonies.

## XI. Summary

In summation, our design consists of three main subsystems: the local application, the signal coordinators, and the physical interface. The local application consists of an OMR parser that takes the input of a pdf/png from the GUI and outputs an XML file containing information about the sheet music. This XML file then gets forwarded to the RPi 4 via scp for further parsing. The RPi 4 then uses the music21 library to extract all the vital information about note pitch, duration, and time stamp of occurrence. Then the parsed notes get scheduled and await for the user to begin playing the piece from the GUI. The communication between the GUI and RPi gets handled by the intermediary Arduino. Once the user starts the piece from the app, high and low signals get sent to the MOSFETs that allow the solenoids to draw current from the power source and press down or pull up according to the output of the note scheduling from the RPi.

Individually, each subsystem when tested was able to meet the design specifications. In general, all of our use case requirements were fulfilled as well after considering the constraints we placed on its scope and use. Limitations that the system has in its current completed state include processing higher complexity or lower quality scores, playing faster pieces and notes outside a one octave range, and having robust functionality without internet. Improving the OMR capabilities would likely not happen with additional time, unless the discovery of a better toolchain than Audiveris was made. Our choices for actuation could be changed and improved with an increased budget, but with more time we may wish to explore a more complex algorithm that can reuse hardware to robotically move solenoids up and down the piano. Removing the necessity of the internet could potentially be overcome by looking into shared memory options or a faster serial mechanism than one that is buffered by an Arduino.

After working individually on our subsystems, the major difficulty was integrating all of these subsystems into one cohesive product. This challenge is inherent with just about any project that has so many moving parts like ours. The major lesson we learned was the importance of dedicating sufficient time to integration. Early on in the process, we were aware of how time intensive this phase of the design process would be. However, despite the fact that we allocated three weeks of slack time in our initial schedule plan, we still ended up using all of it and more to fully complete the integration process. Thus our advice to any teams that wish to pursue a similar project is to keep pace at the start of your project and potentially try to be ahead of schedule early so that you have more time to deal with additional issues and challenges that you inevitably encounter.

## Footnotes

1. High Resolution music scores are limited to those generated via computer software or transcribed to modern print quality. The accompanyBot will accept and attempt to parse low quality or handwritten scores, however, at the cost of lowered accuracy and overall performance than the metrics outlined in the use-case requirements.

## Glossary of Acronyms

BPM – Beats per minute
FTP – File Transfer Protocol
GPIO – General Purpose Input/Output
GUI – Graphical User Interface
MIDI – Musical Instrument Digital Interface
MOSFET – Metal Oxide Semiconductor Field Effect Transistor
OMR – Optical Music Recognition
OOP – Object Oriented Programming
RPi 4 – Raspberry Pi 4
SCP – Secure copy
SSH – Secure Shell
USB – Universal Serial Bus
XML – Extensible Markup Language

## References

[1] "Average piano accompanist hourly pay," PayScale. [Online]. Available: https://www.payscale.com/research/US/Job=Piano_Accompanist/Hourly_Rate [Accessed: 02-May-2023].

[2] "Edelweisspianos.com." [Online]. Available: https://edelweisspianos.com/wp-content/uploads/2022/07/LookBook_V2-Digital-Final.pdf. [Accessed: 05-May-2023].

[3] A. Jain, R. Bansal, A. Kumar, and K. D. Singh, "A comparative study of visual and auditory reaction times on the basis of gender and physical activity levels of medical first year students," International journal of applied &amp; basic medical research, 2015. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4456887/. [Accessed: 02-May-2023].

[4] F. Nah, "A study on tolerable waiting time: How long are web users willing to ..." [Online]. Available: https://www.researchgate.net/publication/240624333_A_study_on_tolerable_waiting_time_how_long_are_Web_users_willing_to_wait_Citation_Nah_F_2004_A_study_on_tolerable_waiting_time_how_long_are_Web_users_willing_to_wait_Behaviour_Information_Technology_f. [Accessed: 05-May-2023].

[5] "Do electric pianos use a lot of electricity? [explained]," Piano Sounds, 21-Jan-2023. [Online]. Available: https://pianosounds.com/do-electric-pianos-use-a-lot-of-electricity. [Accessed: 05-May-2023].

[6] I. Ebonko, "Play sheet music with python, opencv, and an optical music recognition model," Medium, 05-Oct-2021. [Online]. Available: https://heartbeat.comet.ml/play-sheet-music-with-python-opencv-and-an-optical-music-recognition-model-a55a3bea8fe. [Accessed: 05-May-2023].

[7] BreezeWhite, "Breezewhite/oemer: End-to-end optical music recognition (OMR) system. transcribe phone-taken music sheet image into musicxml, which can be edited and converted to MIDI.," GitHub. [Online]. Available: https://github.com/BreezeWhite/oemer. [Accessed: 05-May-2023].

[8] H. Bitteur, "Command line interface," Audiveris Pages. [Online]. Available: https://audiveris.github.io/audiveris/_pages/advanced/cli/. [Accessed: 05-May-2023].

[9] "Piano Man FPGA piano-playing robot - Massachusetts Institute of Technology." [Online]. Available: https://web.mit.edu/6.111/volume2/www/f2019/projects/brendana_Project_Final_Report.pdf. [Accessed: 06-May-2023].

[10] P. King, "Piano-playing robot," The MagPi magazine. [Online]. Available: https://magpi.raspberrypi.com/articles/piano-playing-robot. [Accessed: 05-May-2023].

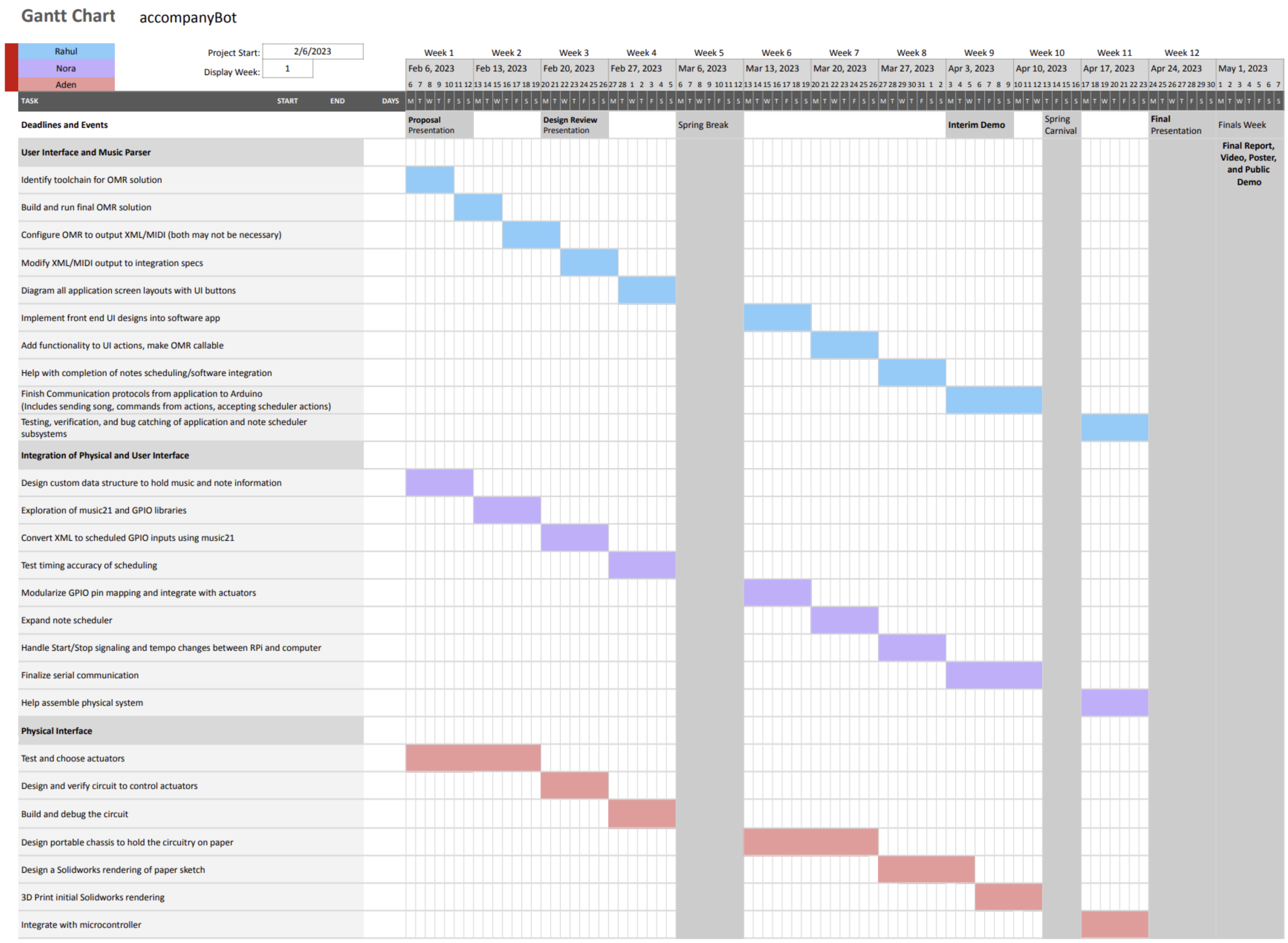


Fig. 9 Gantt Chart

TABLE III. BILL OF MATERIALS

| Description | Manufacturer | Quantity | Cost Per Unit | Total Cost |
|---|---|---|---|---|
| N-Channel Power MOSFET - 30V/60A | Adafruit | 12 | $2.03 | $24.36 |
| Large Push-Pull Solenoid | Adafruit | 12 | $13.46 | $161.52 |
| 1N4004 Diodes | - | 12 | $0 | $0 |
| Protoboards | Adafruit | 1 | $0 | $0 |
| Raspberry Pi 4 - 4GB | - | 1 | $0 | $0 |
| Electronic Keyboard | - | 1 | $0 | $0 |
| Power Supply 30V/5A | - | 1 | $0 | $0 |
| USB to USB-C cable | Amazon | 1 | $5.99 |  |
| PLA 3D Printing Filament (1kg, 1.75mm) |  | 3 |  | $18.99 |
| VELCRO Sticky Back Hook and Loop Fasteners | VELCRO | 1 | $15.65 | $15.65 |
| Arduino Uno |  | 1 | $0 | $0 |
|  |  |  | **Grand Total** | **$226.51** |