Robot Hendrix

In Hwan "Chris" Baek, Joshua Rooks, Hanwen "Kevin" Wang, Tianrui "Ray" Zhang
Department of Electrical Engineering, University of California Los Angeles
Henry Samueli School of Engineering and Applied Science
Email: chris.inhwan.baek@gmail.com, jrooks@ucla.edu, hanwenwang@ucla.edu, jlhsrayzhang@gmail.com

***Abstract*—In this paper, we will describe the overall design of our Robot Hendrix guitar playing robot and the design decisions and implementation of the systems that comprise it. Our system is evaluated with qualitative analysis and quantitative analysis. The evaluation results conclude that our robot is able to successfully play songs on the guitar in rhythm.**

I. Introduction

The goal of our project was to create an electromechanical robot that is able to take in music files and then play the contents of those files on a guitar as instructed by a user. Our system is made up of three parts, a left hand that presses down on the guitar strings at the proper fret to select designated notes and chords, a right hand that picks or strums the strings to play the selected chords and notes in time with the designated song, and a control system that interacts with the user and sends data about the selected song to the left and right hands to play.

II. Background

A. Guitar

A guitar is a string instrument with a fretted neck. There are two different types: acoustic and electric. It is a versatile instrument that can be played with a number of different techniques. Robot Hendrix is designed to play an electric guitar with different techniques.

Fig 1 is a diagram showing the parts of an electric guitar. A

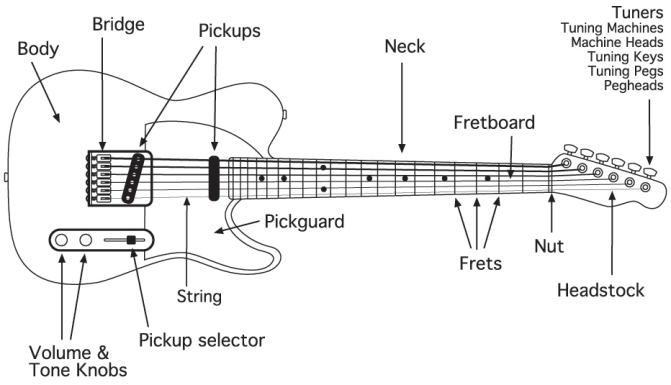

Fig. 1. Electric guitar parts [1]

typical electric guitar has six strings that are tuned to different notes. A guitarist presses strings against frets, metal bars on the guitar neck, to make different notes. The closer the fret is to the body, the higher the note. As shown in Fig 1, frets are not evenly spaced. We had to take into account this uneven fret spacing when we designed Robot Hendrix's left hand.

Another important characteristic to be considered for the left hand design is the string tension. Because the left hand must be able to exert more force than the string tension in order to press the strings, smaller string tension is preferred. The string tension is mainly affected by the string gauge. Typically guitar strings are sold as sets of six strings. A light gauge set includes thinner strings than those in a heavier gauge set. Each string within a set also has different gauge. The first string is the thinnest while the sixth string is the thickest. It is important to test the left hand's strength against the sixth string.

In addition to the string tension, there are other factors that affect the amount of force required to press the strings: string action and fret location. String action is the distance between a string and a fret. The higher the string action, the more the string needs to bend to touch the fret. On most of guitars, the string action is adjustable. We made the string action as low as possible for our system to make our robot's left hand require less force to press the strings. Which fret a string is pressed onto greatly affects how much force is required. It is much easier to bend a string near the mid point than near the ends of the string. The mid point of the strings on a guitar is the twelfth fret. Closer the fret to the twelfth fret, less force is needed to press a string onto. Thus, we need to test the left hand's strength against the fret that is farthest from the twelfth fret to ensure it is strong enough.

The acoustic sound generated by the guitar strings are converted into an electric signal through pickups. Pickups have magnets that magnetize the strings. In addition, as shown in Fig 2, the pickups also have inductive coils that generate an electric signal as the magnetized strings vibrate. As such, the pickups are sensitive to magnetic interference. The robot's

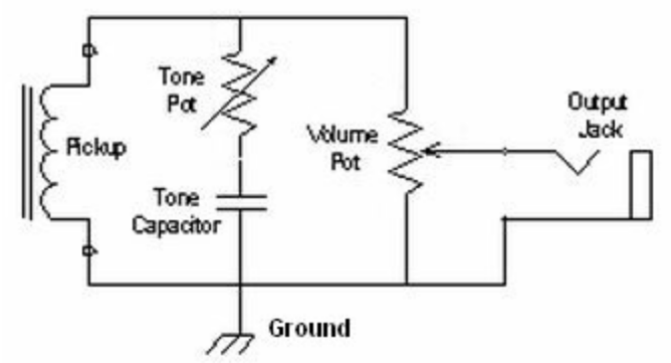

Fig. 2. Guitar Circuit [2]

right hand is what strums or picks the strings. Where strings are strummed or picks affects the tone. Strumming near the midpoint of the strings gives smooth but dull tone while strumming near the ends of the strings gives clear but sharp tone. The most balanced tone is achieved where the pickups are located.

B. Related work

We searched online and found two guitar playing robotic systems similar to what we wanted to accomplish.

1) FolkBox: FolkBox is a robot designed to help people with limited left-hand dexterity to play the guitar. The left hand motions to select different chords play a important role in guitar performance and can be fairly complex. Folkbox allows users to select chords using buttons and forms the chord for them. The main part of Folkbox is a stand embedded with three rows of solenoids. The solenoids are placed right above different chord positions, and are controlled directly through buttons located close to the body of the guitar.

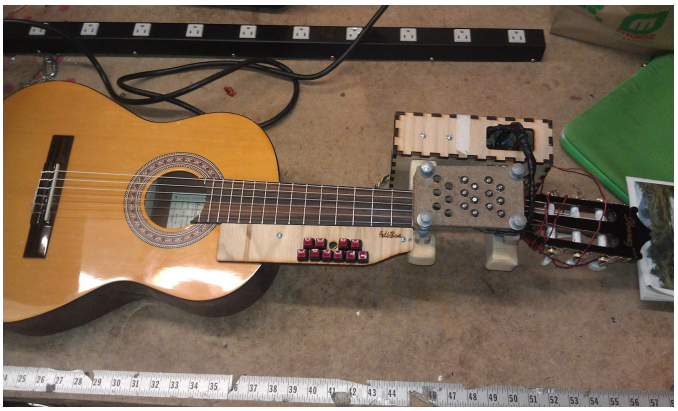

Fig. 3. Folkbox Robot [3]

2) Robot Guitar: Robot Guitar is also a project developed with emphasis on human robot interactions. The main part is a metal stand with six servos attached on both sides. The servos are positioned above six strings with just enough height to perform picking motions. All six servos are directly controlled by an Arduino Uno, which in turn is connected to a Raspberry Pi. The Raspberry Pi is used as a development platform to write Arduino sketches, which are then uploaded to Arduino directly from the Raspberry Pi. Buttons connected to the Arduino allow users to interact with servos.

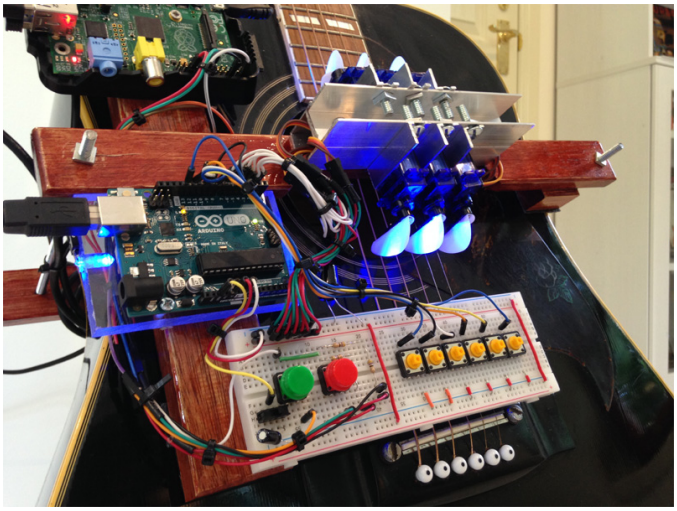

Fig. 4. Robot Guitar [4]

III. System Design

A. Overview

We are building a robotic system which can interact with human and play guitar. This system is composed of three parts just like how human brain control the left and right hand to play a song. The central part is the Edison board which listen to user command and sends control signals to both right and left hand. In order to send the right control signal, the Edison board has a list of songs hard coded into the system as a reference when a certain song needs to be played. The song is stored in the format of MIDI file which then is translated into data structures containing the variables to tell right and left hand what to do.

The user interaction interface is implemented based on speech recognition. The microphone will transfer and collect the analog signal into digital speech signal for the Edison board to process. It will then be recognized through Google speech recognition service. Based on the text file returned by Google service, Edison board can take corresponding reactions.

Since the overall system is composed of several separate pieces, it needs ways to effectively communicate with each subparts. The data transfer between audio interface and Edison is through USB since large about of data need to be transferred in a short amount of time. The Edison sends commands to the left hand circuits through GPIO since our left hand only respond to a press or release signal, which means one bit for each ”finger” is enough. The Arduino and Edition communicate though UART. Since we have six servos to control and sometimes we need some of them to work simultaneously, it requires a signal sent in the scale of byte to identify the servos.

B. Control

On the top layer, Robot Hendrix has its control system. The control system includes three main parts: human interaction,

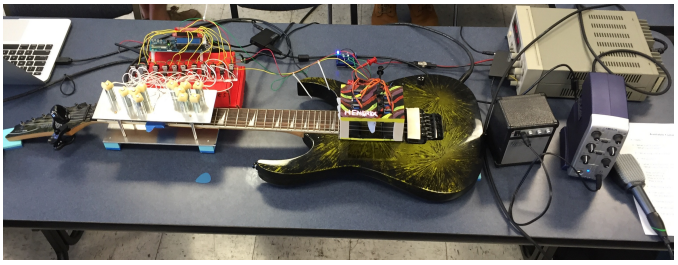

Fig. 5. Robot Hendrix

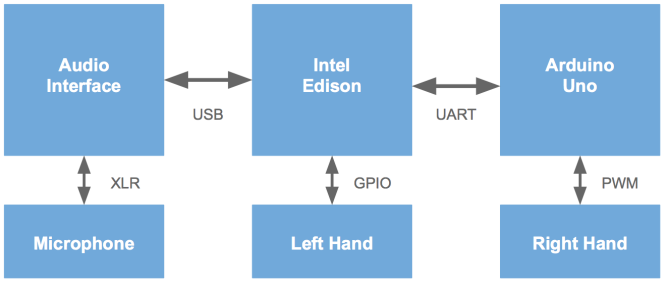

Fig. 6. Overall System

music interpretation, and actuation command control.

Fig 7 illustrates how the control system works. Our robot

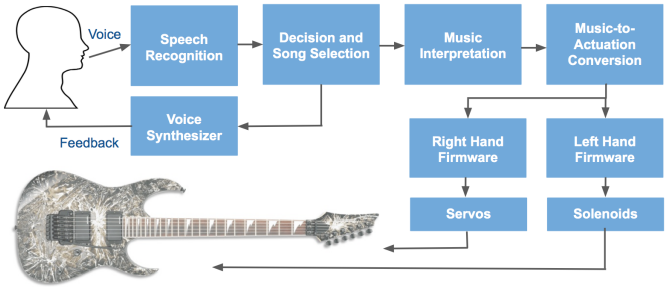

Fig. 7. Control System

listens to humans and performs speech recognition. The text translated from the speech is then fed into the decision and song selection module, which gives a response in synthesized voice and decides which song to be played. The selected song is interpreted and converted into a form robot can understand. The music is then translated into a series of actuation commands, which are sent to the right hand firmware and the left hand firmware. The firmware controls hardware actuators such as servos and solenoids to play the guitar.

1) Voice control: The motivation for voice control is to give an intuitive and natural way for humans to interact with our system. We designed Robot Hendrix in such way that it can adapt a humanoid form. As such, we decided to use speech, the most common method of human-to-human communication, to interact with our robot.

The human voice is captured by a microphone. The microphone generates an analog signal, which is converted to a digital signal by an audio interface. The audio interface we

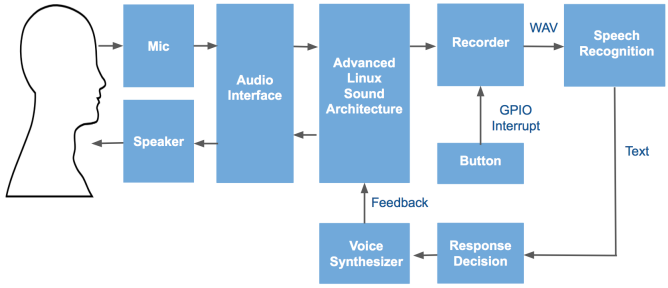

Fig. 8. Voice Control

used includes a mic preamp and ADC. It is ALSA-compatible so that we could use it on an Edison. ALSA, also known as Advanced Linux Sound Architecture, is part of Linux kernel that provides an API for audio interface drivers [5]. It also provides front-end software such as aplay and arecord.

We added a button to initiate the voice control. When the user presses the button, the system starts recording at 44.1 KHz sample rate. Once the user is done, he or she can release the button to stop, this is implemented with GPIO interrupt. When the system detects a falling edge, it kills the recording process and the captured audio is saved as a WAV file.

A robust speech recognition system is not easy to build from scratch. Instead, our system uses Google Speech Recognition API. First, the system converts the WAV file to a FLAC file since Google only accepts audio files in FLAC format. Then, it sends the audio file as an HTTP packet to Google servers. This leads to a system limitation in that our robot requires an Internet connection to have voice control.

Google Speech Recognition returns the generated text, which is passed to a module that determines the response and makes decisions. Our robot does not have a true natural language processing, which is based on statistical machine learning. Natural language processing is also a difficult task and out of this project's scope. Instead, our robot is given a simple logic to process the commands. Some simple commands such as "What is your name?" are directly compared to predefined strings. In other cases, Robot Hendrix parses the command by words and looks for keywords. For instance, if the given command is "Let's jam.", it detects the keyword "jam" and decides to play a blues jam track.

The returned text from the response/decision module is fed into a voice synthesizer. We decided to use a compact open source software called eSpeak. eSpeak uses a formant synthesis method, which is based on additive synthesis and an acoustic model rather than human speech samples [6]. A formant synthesis method made eSpeak a very compact voice synthesizer, but it is not as natural or smooth as larger synthesizers which are based on human speech recordings [6]. The synthesize audio is converted to an analog signal in the audio interface, which includes a DAC. The analog signal is amplified by the audio interface's on-board headphone amp and played on a speaker.

2) Music interpretation: As mentioned earlier, we wanted to make our robot to behave as human as possible. Thus, we designed a music interpretation system for our robot so that it can interpret music and actuate based on the interpreted musical information rather than make a series of hard-coded actuation commands. Robot Hendrix can read music in the industry standard digital format for music called MIDI (Musical Instrument Digital Interface). MIDI files are easy to generate. There are software tools available to transcribe music and save it as a MIDI file. You can also use a midi controller to record music and save it as a MIDI file. Our robot is capable of understanding MIDI files generated in any method.

Fig 9 illustrates our music interpretation system. MIDI files

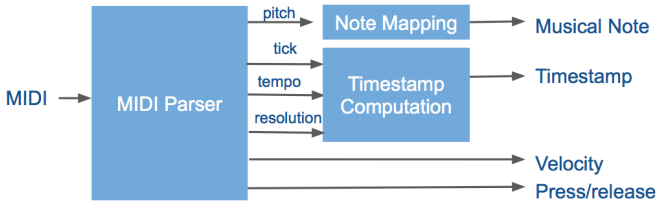

Fig. 9. Music Interpretation

consist of global settings such as tempo as well as series of note events. Each note event contains pitch, tick, velocity, press/release, note bending, fading, and other parameters. The MIDI parser extracts necessary parameter values from the global settings and each note event. As shown in Fig 9, the necessary parameters are pitch, tick, tempo, resolution, velocity, and press/release.

The pitch is given as a positive integer, which is directly mapped to a musical note. For instance, the value of 41 is mapped to F2 note. Incrementing the value by 1 shifts the note up by a half step. So, the value of 42 is mapped to F2 note. Our robot is only capable of playing seventeen notes between E2 and G4. The mapping function only maps pitches that corresponds to the playable notes and mark non-playable notes as ”NA”.

The MIDI provides a relative information on when each note event occurs. This is given as a tick. A tick represents the lowest level resolution of a MIDI track [7]. Ticks alone do not provide enough information on when exactly events occur. In other words, we know how many ticks we have at the instance a note event occurs, but we do not know how much time each tick represents. We need two more types of information, tempo and resolution, to find time. Tempo is given as Beats Per Minute (BPM), which is the number of quarter notes in a minute. Resolution is the number of ticks in a quarter note. From tempo, we can calculate the duration of a quarter note. The quarter note duration and resolution are used to calculate the duration of a tick. With the number of ticks and the duration of each tick, we can generate timestamps for every note event.

As Fig 9 suggests, the velocity and press/release information is used directly. Velocity is how hard a note is played, with a higher value means the note is played harder. Press tells when a note starts playing and release tells when the note stops playing.

3) Actuation command control: The interpreted music is then passed to the actuation command control system, which converts musical information into a series of actuation commands for the left hand and the right hand. Fig 10 illustrates how this is done. We have a timer to handle note event at the

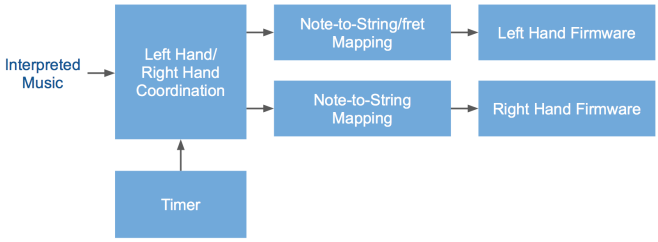

Fig. 10. Actuation Command

correct time. For instance, if we have a press event at 0 second and a release event at 2 second, first, the control system sends actuation command to both left hand and the right hand at 0 second to press and play the note. A timer is used to wait for 2 seconds and then send another command to the left hand to release. The left hand/right hand coordination module splits the interpreted music into the left hand relevant information and the right hand relevant information. The information is then mapped to guitar relevant information, which involves converting musical notes into where to press with the left hand and what strings to pluck with the right hand.

If the robot is playing the same note two times consecutively, it would press and pick the string, then release, then press and pick, and then release again. This is not desirable and affects the sound of the guitar. What we want instead is to press down on the string and pick twice before releasing it. To address this, it must detect consecutive notes and remove release events before the system divides the musical information into the left hand part and the right hand part. If there is two consecutive same note, the release event of the first note is at the same timestamp as the press event of the second note. Our system is capable of detecting this and instead of processing each note event at a time, the system creates a dynamic buffer to store all note event at the same timestamp. Then, the note events in the buffer are processed together to find the optimal motion to be done at every timestamped instance. The example above is optimized by replacing the first release event and the second press event into hold events. This results in the desired actuation sequence.

C. Right hand

For the right hand, our goal is to design a system that can hit the correct strings in time and at the desired speed and angle. Moving the pick in a similar manner to a human right hand is critical to generating an authentic sound. We will discuss more details in the following sections.

1) mechanical: One method of implementation is using a kinematic chain with a guitar pick as an end effector. By adjusting the angle between links on the chain we can move the pick in the right trajectory. After evaluation, this approach has the benefit of generating a more human like strumming trajectory on strings since the movement of the end effector can fully be controlled. However, this approach has has many issues. First, it has the potential for a noticeable delay between two strums which means the song cannot be played in rhythm. Second, to have an end effector hit just on one or two strings when playing a song, we need very precise control, which cannot be implemented easily by servos and arms within our budget.

For our second method of implementation, we decided to use six actuators, one for each string. Although this cannot give us the optimal strumming trajectory, it gives us a very reliable solution. This design gives us guaranteed timing to pick one or multiple strings. Since the actuator is very close to the surface of strings it only needs a very small angle of rotation to pick a note.

For the actuator, we choose to use servos because they compact enough to easily fit in position over the strings. Moreover the feedback control feature in servos allow us to efficiently and precisely control the rotation angle.

Once we decided to use servos for the actuators, we designed a 3D printed frame which can be easily mounted on top of the six strings. The frame holds two rows of servos with one row holding the servos for the even numbered strings and the other row holding the servos for the odd numbered strings. This approach gives us the space needed to pick the strings.

2) electrical: Since we decide to use six servos to hit every one of the strings and each servo has one signal wire, we need to generate six PWM signal simultaneously to effectively control them and play in rhythm. The Edison board that is used for central control cannot be directly used because it only supports 4 PWM pins and 11 of the 13 GPIO pins are used to control the left hand. Because of this, we decided to use an Arduino Uno board to generate the six PWM signals according to the data sent from Edison over UART. UART was used because this it gives us simple and reliable communication from Edison to Arduino.

To supply the six servos with stable power, just using the power supply on the Arduino is not good enough since the I/O pins can only supply at maximum 50 mA of current and the LDO on the arduino can only supply a max of 150 mA which is much lower than what we need to six of them. Because of that, we decide to use an external power supply to generate 5 volts DC and connected it to all six servos.

Servo motors have one ground wire. The ground wire should be connected to a ground pin on the Arduino board. Since we are using an external power supply, we need to make sure its ground wire is also connected to the ground pin on the Arduino board.

The signal pin is typically yellow, orange or white and should be connected to a digital pin on the Arduino board.

One issue we considered when designing the power supply circuit is voltage sag. At the moment when the servos start to move, they draw a lot of current from the voltage source. This can cause the voltage output to sag or even brown out. To make the power supply stable, we connected a decoupling capacitor across each servos' power and ground wires to minimize the voltage sag.

The electric guitar is a good music instrument to be played by a robot since the sound can be amplified by a speaker. However, since the guitar uses inductive pickups to detect string vibration, the EMF generated by the servos can also be captured by the pickups leading to noticeable servo noises at the output. By wrapping each servo with a piece of conductive material to block the magnetic field we can create a pseudo Faraday cage to block EMI from the servos. We used aluminum tape is an easy way to wrap conductive material around the servo.

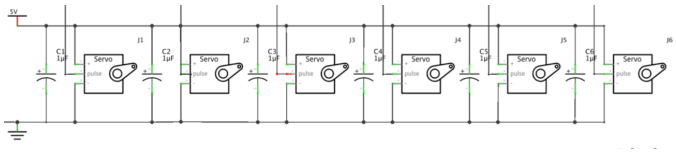

Fig. 11. Right Hand Power Supply Circuit

D. Left hand

The goal of the left hand is to replicate the way a human hand is able to press down on the guitar strings at different frets to form chords or selected notes when the strings are strummed or picked by the right hand. In order to play a wide range of songs, we picked 11 finger positions for the left hand to press, combinations of which allow us to play 7 of the most widely used chords. The chords G, Em, C, Am, D, Dm and F can all be played from a mix of the finger positions along with other, lesser used chords, such as Dsus4 and Em7. The 11 finger positions can be seen in the figure below. In order to reach our overall left hand goal, the following tasks need to be accomplished. First we need actuators to push down on the strings at all 11 positions we designated. The actuators have to be fast enough to quickly change chords, strong enough to firmly push down on the strings so that we get a clear note, small enough that we can fit all 11 on the fretboard and able to retract from the strings so that a higher note on the string can be played. In addition to the actuators themselves, we needed a method to hold the actuators in the correct position in relation to the guitar. Finally we needed a method to control the actuators electronically.

1) mechanical: The first step of our left hand design is part selection. There are two potential design directions we considered for the actuators to obtain controlled linear motion, pneumatic actuators and solenoids. Pneumatic actuators usually consist of a piston inside of a cylinder that is forced to move in a desired direction using a compressed gas, typically air. For our project, we would use a double-acting cylinder which allows us to extend and retract the piston using

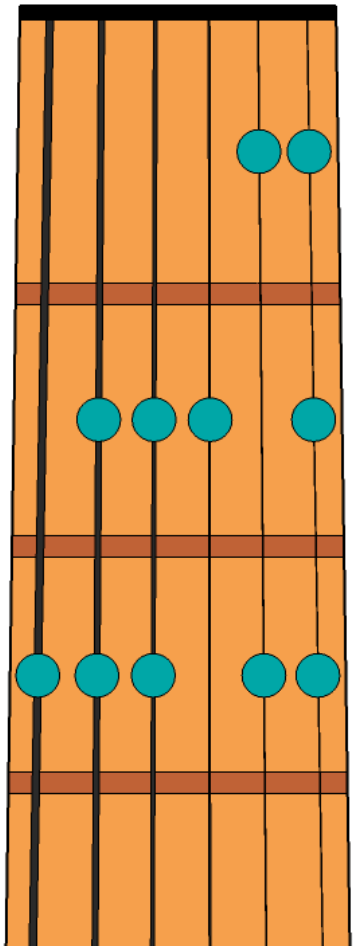

Fig. 12. Finger Positions

a solenoid valve to control the direction of actuation. The solenoid valve allows us to control the actuation electronically by feeding compressed air onto one side of the piston while releasing air from the other side. In addition to the actuator and the valves used to control them, a typical system also needs a compressor to compress air to the proper pressure and a tank to store the compressed air. Plastic tubing is used to allow air to flow between all of the components. The advantages of using pneumatics are that very small cylinders are readily available, the typical actuation distances are larger than for solenoids and you can easily adjust the force that the actuator exerts by changing the pressure of the gas used in the system. In addition the amount of power needed for normal operation is fairly small compared to using solenoids. The most critical disadvantages of using pneumatic actuators are the high cost of components, as well as the amount of space required for the control and supporting infrastructure.

Linear solenoids usually consist of an inductive coil that is wound around a ferromagnetic armature, with the armature free to move forward in back through the coil. When a current is applied to the coil, a force is applied to the armature proportional to the current and position of the armature. Most solenoids have some sort of return mechanism, typically a spring, so that the armature moves in the desired direction when current is applied and then returns to its initial position when the current is cutoff. The solenoid is typically controlled by a relay or transistor which will be discussed further in the electrical section. Solenoids are cheaper than pneumatic actuators and can be controlled electronically, but tend to be bigger and draw a lot of current. They also actuate very quickly. The greater the distance a solenoid actuates, the weaker the amount of force it can can apply becomes. This problem can be addressed to some extent by using Pulse Width Modulation (PWM) to increase the total amount of force the solenoid applies, however this can drastically increase the current consumption of the solenoid. For example, for the solenoid we used, at a 2 mm actuation distance, the force applied using continuous current is 260 grams while the force applied using 10% duty cycle PWM is 1200 grams according to the data sheet. This a substantial improvement with the force applied increasing by a factor of over 4.5 by using PWM. The drawback to this though is that while a solenoid using continuous current would pull 5.5 Watts, a solenoid using 10% PWM would pull 55 Watts of power at the same voltage, 10x what the continuous current solenoid would pull. The increase of power needed by using PWM is over double the increase of force achieved.

The small size of the pneumatic actuators combined with the longer actuation distance and the ability to adjust the force exerted by the actuators would allow our design to be more straightforward and flexible, making them an ideal choice. Unfortunately, one of the main considerations for our design is budget, and to stay within our budget using pneumatic actuators we would have to scale back on the number of finger positions we cover. This would, in turn, reduce the number of chords and notes available, limiting the types and amount of music we can play. Solenoids allow us to hit all 11 finger positions, giving us the musical flexibility we want, while staying within our budget constraints.

One of the most important aspects to look at when choosing a solenoid is the amount of force it can exert. In order to press down on the guitar strings in the region of the guitar we want to play on, we need to exert a force of at least 80 grams. Any solenoid we use needs to be able to apply at least that much force at a distance of 2 to 4 mm in order to not interfere with the string when the solenoid is in its initial position and be able to press down the string to the fret when it is actuated.

Space constraints also play a large role in picking a solenoid. As the strings travel from the top of the neck to the base of the guitar, the space between the strings widens as does the width of the neck. In addition, the spacing between the frets gets narrower as you move from the top of the neck to the base. The spacing between the strings is between 7 and 8 mm for the first 7 frets and the spacing between frets changes from 36 mm to 22.9 mm in the same stretch. Because the smallest solenoids that we could find that can exert enough force are larger than 8 mm, some offset is needed to fit the solenoids

in proper position.

Solenoids typically come as either tubular or open frame. A tubular solenoid allows you to panel mount the solenoids, while an open frame needs a more complicated method of attaching the solenoids. Because of this extra complication and the fact that the open frame solenoids we found were larger, we decided to use tubular solenoids. The smallest tubular solenoids we could find that exerted enough force at distance were Sun Magnet SMT - 1632S Solenoids, with an outer diameter of 16mm and an applied force of 100 grams at 4 mm.

The other part of the mechanical design is the frame holding the solenoids in position above the strings. The most straightforward design we could think of consists of two plates offset by standoffs and held together with screws as seen in the figure below. The top plate has holes drilled into it for all 11 solenoids as well as for the screws to attach the plate to the standoffs. The bottom plate acts as a base.

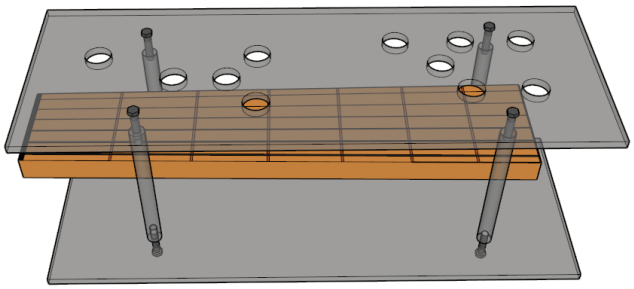

Fig. 13. Model of Frame Design

Because of how much bigger the solenoids are compared to the spacing in between the strings, solenoids within two strings of each other have to be offset vertically along the fretboard. If possible, this should be kept to a minimum since the further away a string is pressed from the fret, the more likely it is to make a buzzing sound when picked. In the initial design, as seen in the figure below, the solenoids were able to fit so that every finger position is covered, but some armatures are very far from the frets and the solenoids are very close together, some with less than 1 mm spacing between them. Since the section of the solenoid that mounts is only 11 mm in diameter, this leaves less than 6 mm between some of the holes drilled in the plate to mount the solenoids.

To improve the spacing between the solenoids as well as their proximity to the frets, we came up with a new design where the solenoids on even numbered strings stay in their normal position while solenoids on odd numbered strings are put 4 frets down from their normal locations. By detuning the odd number strings 4 half steps and putting a capo that only affects the odd numbered strings on the 4th fret we increase the pitch on those strings by 4 half steps, effectively allowing us to achieve normal tuning while increasing the space between the solenoids. This also allows us to keep the armatures close to

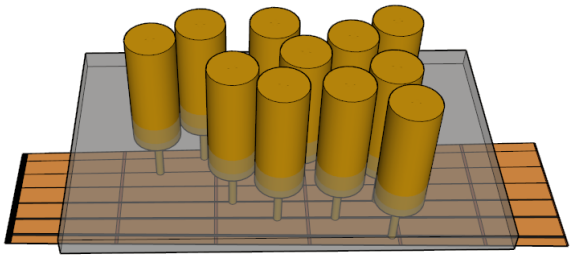

Fig. 14. Initial Solenoid Positioning

the frets to avoid buzzing and increases the distance between holes in the plate to over 8.5 mm. This can be seen in the figure below.

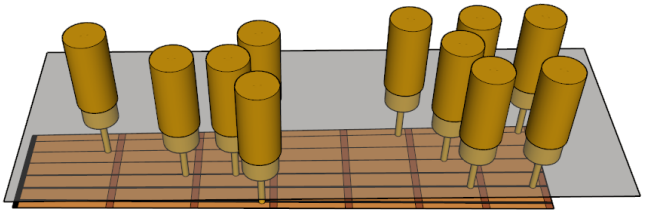

Fig. 15. Final Solenoid Positioning

Both plastic and aluminum were considered as the base material for the frame. By making the frame out of plastic using 3D printing, we could easily go through multiple design iterations giving us more flexibility to address unforeseen problems. It also allows us to be fairly precise since we can configure all the dimensions using 3D modeling software. The downside to plastic however is that it is weaker than aluminum and the 3D printer readily available to us has a hard time with the resolution of small holes. Aluminum is strong, lightweight and cheap, but needs to be cut and drilled to be made into the frame that we want. We chose aluminum as our material because the distances between holes is very small and we were concerned about the plastic cracking under the pressure when the solenoids press against the strings.

2) electrical: The design goal of the electrical system for left hand is to provide sufficient voltage and current to drive all the solenoids needed during a performance. Therefore, our first step is to make sure that the chosen power adapter is capable of supplying such voltages and currents, and the wires picked are thick enough to tolerate the current draw. From the datasheet, each solenoid requires 12V and at least 500mA to operate. However, at least three different solenoids will be needed at

the same time while chords are being played. This imposes the lower limit for our choice of adapter and wires. Since 3 simultaneous solenoids require about 1.5 Amps of current, we chose a 12V, 2A adapter and wires that can tolerate 3.5A.

Next, in terms of controlling the solenoids, we have a choice between PWM signal and continuous signal that can be output by GPIO pins. According to the datasheet, solenoids can output stronger forces when inputs are PWM signals with the proper period. This may be useful depending on the strength of the solenoid return mechanism. However, due to the fact that Edison only has four PWM pins, the overall circuit design will be more complicated if we adopt this method. For example, we will need to use an external PWM board or IC to generate more PWM signals. In order to select the correct PWM pins, we may have to set up communications between this external board and Edison as well. This will introduce higher complexities to the overall circuit designs and make the system more error-prone. As mentioned above, solenoids driven by PWM waves can draw as high as ten times more power than when driven using continuous signals. This not only puts extensively more pressure on the amount of current provided by the power adapter, but can generate a considerable amount of heat at the same time. Since our system ended up working with the amount of force produced when using continuous current, that became our final choice.

For controlling the high voltage circuit with the low voltage outputs from Edison board, we have also tried two different methods. First, we were considering using a relay board. The relay board has 16 relay module on it which are electromechanical switches that can be controlled by a 5V header pin. The relay board gives us a easy solution to control all eleven solenoids at the same time. Also, because relay is a mechanical switch in essence, when it is at off state, it has a very high breakdown voltage and, therefore, can protect the rest of the circuit. However, there are also drawbacks associated with it. It cannot react swift enough to the control commands. The fastest reaction time is around 2ms, but sometimes we need the solenoids to react faster. At the same time, it is very noisy during operation. The noise may affect our guitar performance and we may have to find other ways insulate the noise. The other option for controlling the high voltage circuit is to use a transistor. Both MOSFETs and BJTs act as switches, allowing us to control whether current is passed through or not depending on the voltage applied to the gate or base terminal. Both transistors allow for much faster switching than the relay, although the BJT disipates more current across it and switches slower than a MOSFET. In this design, we used 12 N-channel MOSFETs to instead of the relays. The details will be discussed in the implementation section.

3) control: The left hand control module is relatively simple. It listens for a command from the control layer's, which contains the information of (string number, fret number), and then triggers the operation of corresponding solenoids.

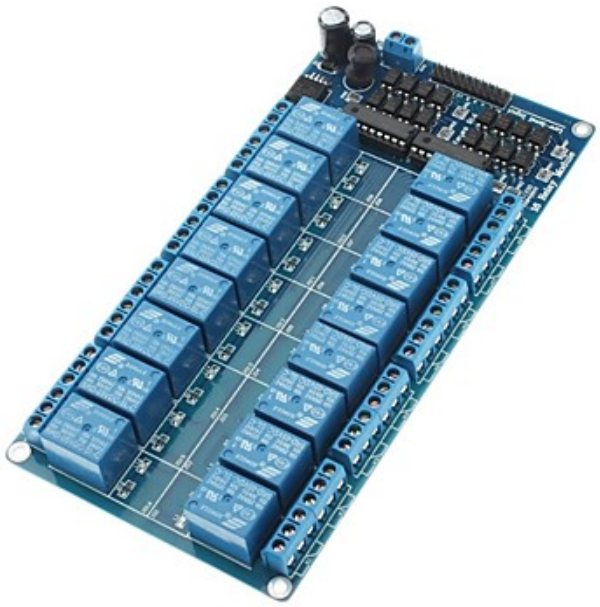

Fig. 16. 16 Channel Relay Board

IV. System Implementation

A. Control

The control system is hosted on an Intel Edison, which run embedded Yocto Linux. Having Linux enables many different options for software implementation. In the interest of saving development time, the software side of the control system is implemented in Python. Instead of implementing from scratch, we also utilized Python libraries and software.

1) Voice control: In order to have voice control, our robot must be able to listen and speak. This requires hardware to capture and play audio. We implemented this with a microphone, an audio interface, a speaker, and other miscellaneous components. The complete setup of the hardware is shown in Fig 17. The box-shaped component on the left is the speaker. The component with six knobs in the center is the audio interface. Lastly, we have a microphone on the right.

The audio interface we chose is Lexicon Lambda, which can be connected to the Intel Edison via USB. As mentioned, the Lexicon Lambda is ALSA-compatible. We tested its compatibility with a software tool provided by ALSA called aplay. "# aplay -lL" command displays all hardware audio interface devices ALSA detects. Fig 18 shows the list of audio interfaces available with our setup. We wrote a simple configuration file, as shown in Fig 19 to set the Lexicon Lambda as the default device.

Another ALSA tool, arecord, is used as the audio recorder. When a user presses the button, the Python program makes a system call to start an arecord process to record audio and save it as a WAV file. When the button is release, the program kills the process.

Fig. 17. Audio Hardware Setup

```
**** List of PLAYBACK Hardware Devices ****
card 0: Loopback [Loopback], device 0: Loopback PCM [Loopback PCM]
  Subdevices: 8/8
  Subdevice #0: subdevice #0
  Subdevice #1: subdevice #1
  Subdevice #2: subdevice #2
  Subdevice #3: subdevice #3
  Subdevice #4: subdevice #4
  Subdevice #5: subdevice #5
  Subdevice #6: subdevice #6
  Subdevice #7: subdevice #7
card 0: Loopback [Loopback], device 1: Loopback PCM [Loopback PCM]
  Subdevices: 8/8
  Subdevice #0: subdevice #0
  Subdevice #1: subdevice #1
  Subdevice #2: subdevice #2
  Subdevice #3: subdevice #3
  Subdevice #4: subdevice #4
  Subdevice #5: subdevice #5
  Subdevice #6: subdevice #6
  Subdevice #7: subdevice #7
card 1: dummyaudio [dummy-audio], device 0: 14 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: dummyaudio [dummy-audio], device 1: ((null)) []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: dummyaudio [dummy-audio], device 2: ((null)) []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 2: Lambda [Lexicon Lambda], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

Fig. 18. List of Detected Audio Interfaces

The WAV file is converted to a FLAC file, which is then sent to Google Speech Recognition web service as an HTTP packet. This process is done with the Python speech recognition package, which provides functions that further abstract the Google Speech Recognition API. The response from the web service is stored as a string for the response/decision module to process.

The response to the command is returned as a string and rearranged into a Linux command for the control program to make a system call with. With the Linux command, we run eSpeak and pass the response as its argument.

```
pcm.!default {
        type plug
         slave {
                pcm "hw:2,0"
        }
}
ctl.!default {
        type hw
        card 2
        device 0
}
```

Fig. 19. Audio Hardware Configuration

2) Music interpretation: All MIDI files are stored in a directory on the Intel Edison for the control system to read from. It is easy to add more MIDI files to expand our robot's song repertoire. Although any method to generate MIDI files is supported, we used a software application called Guitar Pro to transcribe songs and save them as MIDI files.

We used a Python MIDI library function to read the MIDI files. As it reads a MIDI file, it will reorganize it into a list of objects. We created a program that extracts useful parameter values from the objects. It translates the parameter values into timestamps, notes, velocity, and press/release. These translated values are rearranged into a data structure that is understood by our robot. An example of rearranged music data is shown in Fig 20. Each row corresponds to note event. The first column

```
[0.0, 'E2', 72, 'ON']
[0.49968, 'E2', 'NA', 'OFF']
[0.49968, 'G2', 72, 'ON']
[0.99936, 'G2', 'NA', 'OFF']
[0.99936, 'C3', 80, 'ON']
[1.2492, 'C3', 'NA', 'OFF']
[1.2492, 'B2', 80, 'ON']
[1.4990400000000002, 'B2', 'NA', 'OFF']
[1.4990400000000002, 'A2', 80, 'ON']
[1.99872, 'A2', 'NA', 'OFF']
[1.99872, 'A3', 64, 'ON']
[2.4984, 'A3', 'NA', 'OFF']
[2.4984, 'G3', 64, 'ON']
[2.9980800000000003, 'G3', 'NA', 'OFF']
[2.9980800000000003, 'C4', 88, 'ON']
[3.123, 'C4', 'NA', 'OFF']
[3.123, 'D4', 88, 'ON']
[3.24792, 'D4', 'NA', 'OFF']
[3.24792, 'E4', 88, 'ON']
[3.49776, 'E4', 'NA', 'OFF']
[3.49776, 'F3', 88, 'ON']
[3.7476, 'F3', 'NA', 'OFF']
[3.7476, 'E3', 88, 'ON']
[3.9974399999999997, 'E3', 'NA', 'OFF']
```

Fig. 20. Rearranged Music Data Example

is the timestamp. The second column is the note. The third column is the velocity. The fourth column is press/release. For instance, [0.49968, G2, 72, ON] means Hit G2 note with velocity value 72 at 0.49968 seconds.

3) Actuation command control: The actuation command control system is implemented in pure Python. It reads the music data structure and computes the time duration between two consecutive note events. If the time duration is 0 second, it increments a counter. The counter value is used to find the events that occur at the same instance and store them into a buffer. A nested loop checks all events in the buffer and finds all notes that are played multiple times consecutively. This process is not as simple as comparing one event to the next event. We can have multiple "press" events at the same instance when the robot is pressing multiple string at once. The algorithm must search through the whole buffer. The redundant consecutive notes are marked as "hold" events.

The music data is divided into the left hand data and the right hand data. The timestamps are necessary for both hands as the left hand needs to know when to press/release and the right hand needs to know when to pick. The notes are necessary for both hands as the left hand needs to know where to press/release and the right hand needs to know which string to pick. The velocity is only relevant to the right hand control because the note velocity determines how hard the string is picked. Press/release information is necessary for both hands as it tells the left hand whether to press or release and the right hand whether to pick or not.

The note information is mapped to the string and fret numbers for the left hand control. The left hand control system is modular and only knows what string and fret to press on. The mapped string number and the fret number is arranged along with press/release/hold into a list, which is passed to the left hand control firmware at the correct time based on the given timestamp. The note information is also mapped the string numbers for the right hand control. The string number is passed to the right hand control firmware at the correct time based on the given timestamp.

B. Right hand

1) mechanical: Since each servo is only 9 grams, the total weight can easily be supported by normal plastic material. Moreover, to build a reliable frame with precise servo positions, it requires an efficient way to build new frames based on the adjustment made on the last version. Based on these two reason, we decided to use a 3D printer to implement our design.

To generate the 3D model, we used SketchUp which is an software that is free and easy to use. Since the space between two adjacent strings is too small to fit adjacent two servos, we decided to use two rows of servos to eliminate the picking interference. As you can see in the figure below.

The arms on the servos doe no have a good shape to effectively pick strings and since it's too short, the linear velocity at the end of the arm is not big enough to generate the sound. To solve this, we mounted a guitar pick on each of the servo arms. To make the two separate pieces connected tightly together, we need to use screws. Since we want to make them all have the same length so that they can generate a uniform sound, the drill position on all the picks needs to be the same.

Fig. 21. Servo Arm Shape

When human being plays guitar, they usually use hands to mute the sound when it is needed. To let our robot have the similar ability, we paste a thick piece of tape at the end of the string to dampen them and reduce the time the strings vibrate.

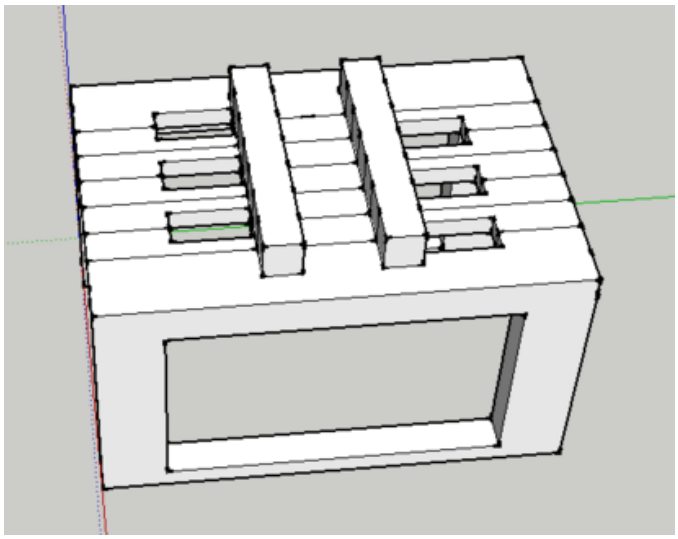

Fig. 22. Right Hand Frame 3D Model

2) electrical: All servos are connected in parallel with a 5V DC voltage source and a ground pin on Arduino board. The signal wire between power and ground wire is connected to five PWM pins on the Arduino board. To prevent the voltage sag , every servos' power and ground wires are also connected with a decoupling capacitor. The signal wire of every servo is connected to one of the PWM pins on the Arduino board.

3) control firmware: The control of the six servos are done by the Arduino board. We wrote firmware on the Arduino

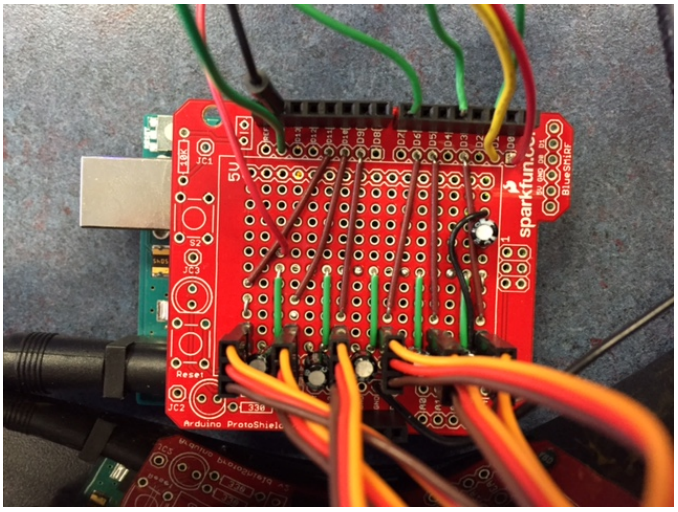

Fig. 23. Right Hand Power Supply

board which allows it to interact with the higher level control. In the firmware, we provide two kind of functions. First one is for picking a desired string. We first map every string to its respective servo. To let each servos' arm start to pick from the appropriate position, we set 6 offset variable to adjust the initial configuration of the servo. During the setup, we also need to connect the correct PWM pin number with the correct servo. This is done by the function attach(int PinNumber) in the servo library.

After the setup, to we address the complication of controlling the servo according to their previous position. This is necessary because after every strum, different servos arms will be at a non-uniform position. So, we set a flag for every servo as a reference to their current state. When the flag is one, it means it's in its most right position. Negative one means the opposite. After a pick function call to a servo, their state is multiply by a negative one. In the pick function, it will check servo's current state and do the right rotation.

Another function provided by this framework is for strumming. To get the closest performance effect, we separate the strumming functions into up strumming and down strumming. That is because when a human strums the strings the effectively pick each string in order with a small delay in between. To fully mimic this behavior, we add a small delay in every pick function with a down strum starting from the top string and an up strum start from the bottom string. For even more accuracy, we multiple that delay with a scale number passing from higher layer of control. For example, if the robot is playing a fast song, the delay in the strum needs to shrink.

C. Left hand

1) mechanical: To find the exact locations needed for the solenoids and standoffs, we first modeled the guitar neck in Sketchup. We then modeled the actuators and moved them into the correct position on the neck so that the armatures are as close to the frets as possible while maintaining at least 3 mm between the solenoids. The standoffs were positioned so that they hugged the neck of the guitar. A model of the standoffs and and armatures was created based on those positions and 3D printed. It was then placed on the guitar neck to confirm the locations. Once the position of all the solenoids and the

Fig. 24. Model and Print of Solenoid and Standoff Placement

standoffs were finalized, we found the center point of each hole and then created a mechanical drawing in AutoCAD to use when machining the aluminum for the frame. The mechanical drawing can be seen in the figure below.

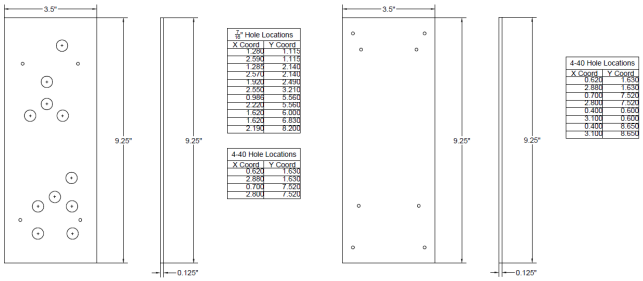

Fig. 25. Mechanical Drawing of Left Hand Frame

To make the frame, first the two 6 x 12 aluminum plates were cut down to 3.5 x 9.25 using a shear. Next the two plates were stacked together and clamped, and the four screw holes for the standoffs were drilled. Using a mill with an x-y coordinate display, the holes were then drilled in the top plate. Each hole was drilled 3 times using progressively larger drills bit to achieve the 7/16 opening and then cleaned up using a dremel. The standoffs were then attached to the base plate using 4 4-40 screws and the solenoids mounted to the top plate.

One of the biggest issues we faced with the solenoids is that they didnt come with any return mechanism. To address this, we first ordered springs that fit the armature, however they had a spring constant of .9 lb/in or 157.6 grams/mm. With these springs all of the force the solenoids output is used up compressing the spring even at the minimum 2 mm distance. We searched for springs of the appropriate size and spring constant, but they were unavailable in such small quantities. Instead we discovered some foam around the lab and found it to be an ideal replacement for the spring, easily compressible but able to return to its initial height after being compressed.

We also had an issue lining up the armatures to press down directly on the strings without slipping. Since the armatures are only 2.5 mm in diameter and both the armatures and the strings are metal, the positioning precision required was hard to achieve even with modeling the guitar neck and using the

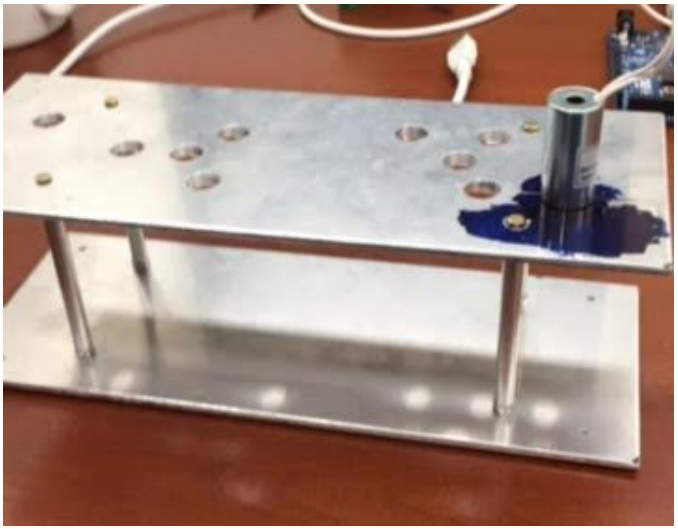

Fig. 26. Preliminary Frame Assembly

mill. We cut rubber feet to size and epoxied them to the end of the armatures. This gave us a larger area to hit the strings as well as more grip.

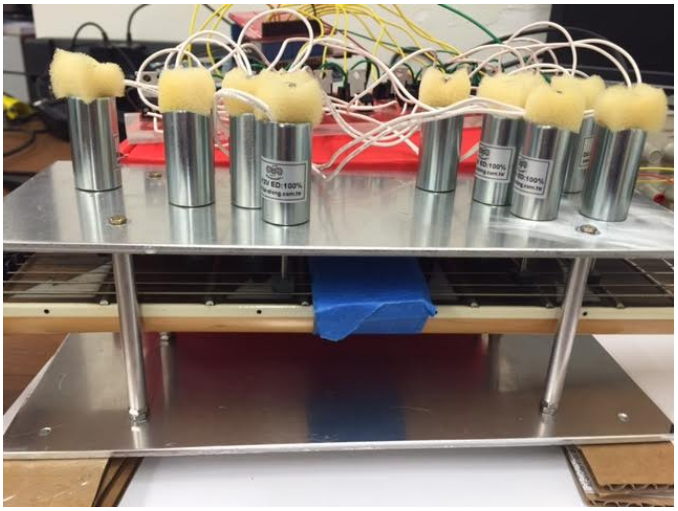

Fig. 27. Left Hand Assembly

The final piece of mechanical design was to 3D print blocks to hold up the neck and base of the guitar as well as the frame. We used cardboard and sheets of paper to find the correct height of the all 4 blocks and then measured them and created models in Sketchup before sending them to the 3D printer.

2) electrical: The implementation strictly follows the design. For the MOSFET, we used a Sparkfun MOSFET breakout board. It has screw terminals at both ends, which simplifies our wire connections. Specifically, the positive end of the power adapter is connected to the positive end of the solenoid. The negative end coming out of solenoid is in turn connected to the drain of N-channel MOSFET, and the source of the MOSFET feed backs to the negative pin of the adapter. Lastly, the gate of MOSFET is controlled by the GPIO pin of the Edison board. The MOSFET are connected to the negative pin of solenoid because N-channel MOSFET is more efficient to pull down the voltage rather than pull up. All positive ends of solenoids are daisy chained together so that they all share a common VDD. All ground wires and the GND pin of Edison board are also daisy chained to have a common ground. It is also worth mentioning that we put a diode between the two terminals of each solenoid to protect the rest of the circuit. The helps protect the Edison from transient spikes that might build up in the solenoids.

The schematic of the circuit is shown below:

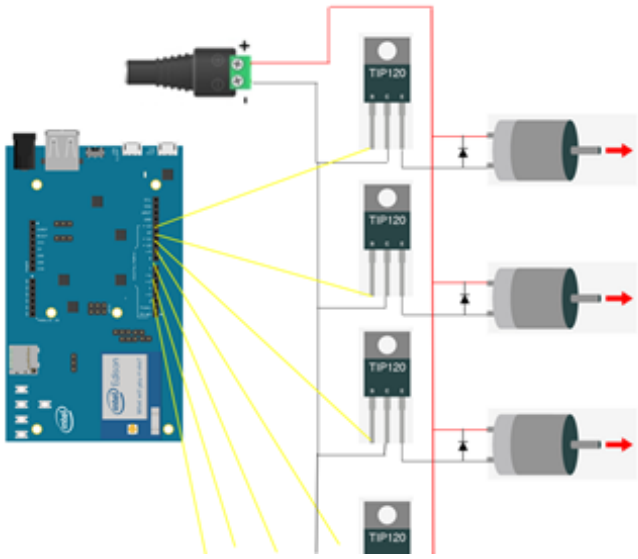

Fig. 28. Left Hand Circuit Schematic

The actual circuit is shown in Fig 29 (The red breakout board is one soldered by ourselves to make connections between GPIO pins and MOSFET gate more portable):

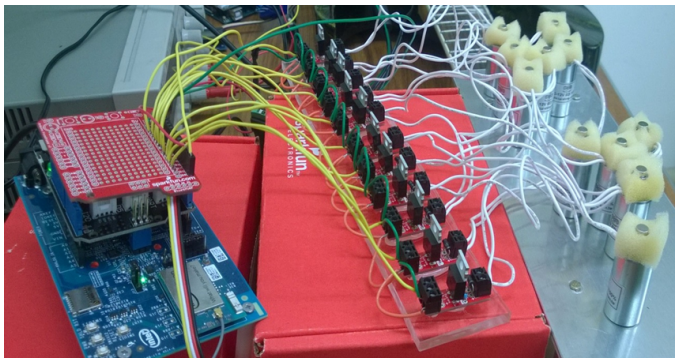

Fig. 29. Left Hand Circuit layout

3) control: Given the tasks for the left hand control module, the left hand low level control is implemented as a Python function call. The module internally maintains a mapping between the (String number, fret number) parameters and the GPIO pin number for each solenoid. Whenever called, it will turn on or off the required solenoid according to control layer command using the mapping.

D. System integration

We decided to design the system to be highly modular with interfaces between modules clearly defined beforehand. We purposely did this so that each team member could work on their own modules. More importantly, this makes the system integration more easy as long as each module strictly follows their interfaces. In fact, it turned out that this decision helped us a lot during the final system integration.

Specifically, in terms of hardware, we wanted to keep the hardware parts as few and simple as possible. Because the Arduino Uno is necessary for right hand control and recording devices are definitely needed to acquire the sound information, we needed at least need these two components with addition of Edison board. Next, in terms of the hardware interfaces between them, the recording devices connect to Edison board through USB serial port. We used a Lexicon Lambda Desktop Recording device, and fortunately, the Edison Arduino break-out board also supports USB interface.

On the other hand, the communication between the Edison and the Arduino is up to us to choose. As mentioned earlier, UART interface became our final choice. One competitor in this case is I2C connection, which is also a common used interface for short distance communication. I2C requires two pull up resistors to operate adding additional parts to our design and since it is half-duplex issues of bus contention can arise unless properly managed. At the same time, UART can perfectly do the job with two wires and is full-duplex. Pin0 and Pin1 on both the Edison and Arduino are reserved to support UART communications so we simply crossed and connected them together.

For the design of the software, in general, the whole robotic system can be divided into three main parts, that is, the control layer, left hand module, and right hand module parts, in which the control layer can be further divided into two subparts - voice control module and command generation module. The whole control layer and the left hand control module can coexist as a single large Python program and sit in the same Edison device. Therefore, the interfaces between subparts of control layers as well as interfaces between the control layer and left hand module are all simply our own defined python API calls. We just need to make sure that the parameters of APIs defined include necessary information that the upper layer needs to pass down. For example, the API for left hand module is defined as (String Number, fret Number, Press/Release). The API details can be found in our source codes. As mentioned, the right hand control involves UART communications. Therefore, the interface is defined as the format of data sent through the wires. Since we only need to know which string to pick for the right hand, one byte of integer value will meet the requirements. Data is sent over the UART at a baudrate of 115200.

During integration, we encountered several challenges such that the whole system did not work as we expected. Because the issue can come from mechanical implementations, circuit problems, or simply software bugs, every time we encounter problems, we need to troubleshoot methodically. Luckily, the problems we faced were not too serious and were solved relatively easily. One typical problem we encountered was wiring issues due to a cold solder joint on the protoboard or loose connection in screw terminals. Another problem was that we had software bugs when parameters passed are in boundary cases. We believe that our choices during the design phase were very helpful in this case.

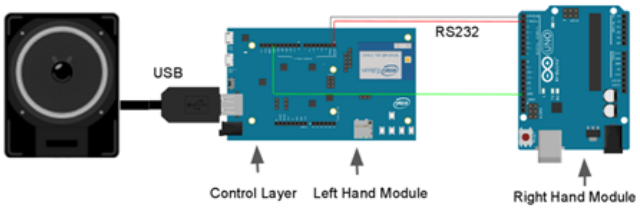

Fig. 30. System Overview

V. System Evaluation and Results

The system is evaluated with qualitative analysis and quantitative analysis. The qualitative analysis is human evaluation, in which our robot plays various songs and test whether humans can recognize. The quantitative analysis is waveform analysis on audio recordings of the robot's playing. The waveform analysis is performed with Audacity and Apple Logic Pro's Flex time tool.

A. Human Evaluation

Songs we tested are "Sweet Home Alabama", "Let It Be", "You Shook Me All Night Long", "Wonderwall", "La Bamba", "Fortunate Son", and "Blues Jam". The quality of the robot's performance was different for different songs. The table I shows the reviewers' average score in a scale of one to ten.

Song	Average Score
"Sweet Home Alabama"	8
"Let It Be"	0.19
"You Shook Me All Night Long"	8.25
"Wonderwall"	7.5
"La Bamba"	6.5
"Fortunate Son"	7
"Blues Jam"	9

TABLE I
Performance Evaluation

Songs with high scores are "Sweet Home Alabama", "You Shook Me All Night Long", and "Blues Jam". These songs generally have low notes and chord strumming with some note picking. Another common characteristic is that these songs are very rhythmic. "Wonderwall" and "Fortunate Son" scored slightly lower because they were chord-only songs and it was harder to recognize these songs. "La Bamba" scored lower because this song lacks stacked notes. The right hand picks better on some strings than it does on other strings. When it plays multiple notes at the same time, the performance gets evened out. Because every note in "La Bamba" is played by

itself, the some notes sound louder than others. As such, the performance is uneven. ”Let It Be” received a very low score. The MIDI file for ”Let It Be” covers both the chord and the melody. Our robot is not very good at play melodies. If a human player is given the music sheet for ”Let It Be”, he or she would pick the notes in the melody harder than the notes in the chords so that the melody is well heard. Our robot is not capable of distinguishing the melody from the chord and pick strings with different velocities. The melody in ”Let It Be” is in high notes, which aren't played well. ”Let It Be” does not a distinctive rhythm pattern like other songs do.

B. Waveform Analysis

First, we made the right hand to pick the same string as fast as possible to measure the minimum note duration our robot can support. The left hand speed analysis is omitted because the servos on the right hand are the bottleneck of our robot's speed. We chose to test on the sixth string because it is the thickest string and it is harder to pick than the others. The open sixth string gives an E2 note, whose frequency is 82.4Hz. In order to make the recorded waveform clearer, we applied a low pass filter with the cutoff frequency at 90Hz. Fig 31 is the waveform of the recorded audio after applying the low pass filter. Each burst represents a pick. We measured the time

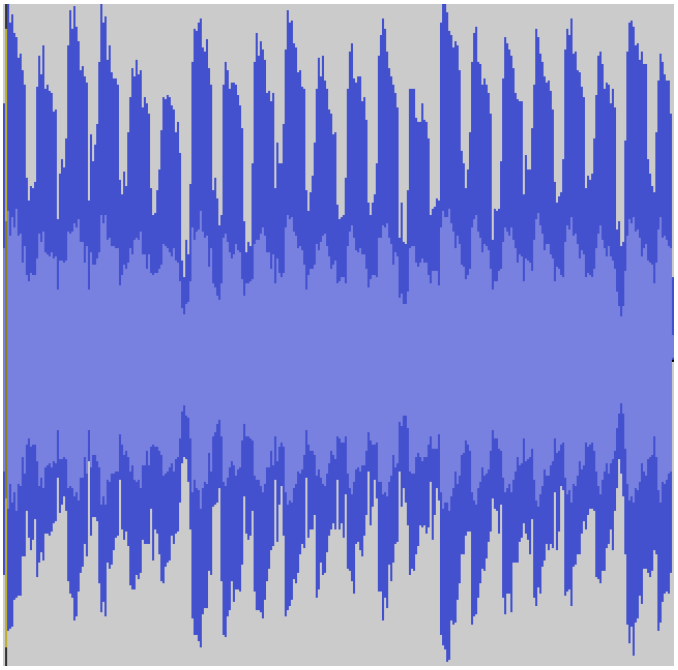

Fig. 31. Picking at Maximum Speed Waveform

duration between two consecutive bursts. This time duration represents the minimum note duration our robot can support. The table II shows the time duration of the first six bursts in the waveform. According the our measurements, it takes 0.2 second on average to pick once. In a music expression, the fastest our robot can play is a sixteenth note at 75 BPM or an eigth note at 150 BPM.

Burst	Time duration (second)
#1	0.21
#2	0.19
#3	0.20
#4	0.21
#5	0.19
#6	0.19
Average	0.20

TABLE II
TIME DURATION BETWEEN EACH PICK

The timing analysis on our robot's actual music performance is as simple as that on repetitive picking. Fig 32 is the waveform of an audio recoding of our robot performance on the song ”La Bamba”. Unlike the picking waveform, it is hard

Fig. 32. Waveform of Robot's Performance on ”La Bamba”

to find the notes on the ”La Bamba” waveform. Apple Logic Pro's Flex time tool comes in handy when finding beats in a waveform. It adds markers to what it thinks is a beat. It often misses some beats or adds a marker when there is no beat. However, the tool finds the precise location of obvious notes. Fig 33 is the waveform with the Flex time markers. As you

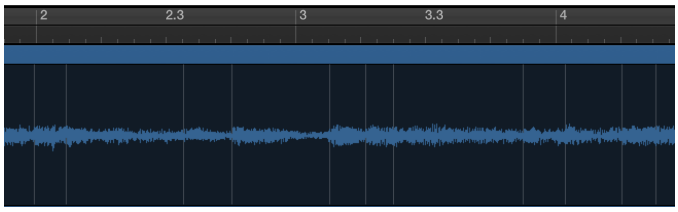

Fig. 33. Waveform Robot's Performance on ”La Bamba” with Flex Time Markers

can see, most of the Flex time markers fit fairly well to the musical grid on the top of the figure. Each tick on the ruler represents a sixteenth note duration.

”La Bamba” only includes notes that are longer than the minimum note duration our robot can support. The shortest note in the song is 0.28 second long. We did the same timing analysis on a different song that includes notes that are shorter than the minimum note duration. ”Sweet Home Alabama” includes notes that are 0.15 second long. Fig 34 is the waveform of our robot's performance on ”Sweet Home Alabama”. The first note of the second bar fits to the grid fairly well because the first bar does not include any short notes. However, the second bar includes notes that are too short for

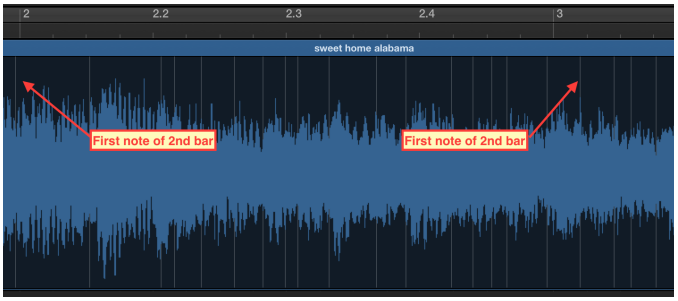

Fig. 34. Waveform of Robot's Performance on "Sweet Home Alabama" with Flex Time Markers

our robot to play in time. The duration of these notes are 0.15 second and played with 0.2 second duration instead because 0.2 second is the minimum duration. There are five of these short notes. Thus, $(0.2s - 0.15s) * 5 = 0.25s$ delay added to the second bar. In Fig 34, the marker position of the third bar's first note reflects this 0.25 seconds delay since each tick is roughly 0.3 seconds in this case.

VI. Limitations and Future Work

As section V suggests, Robot Hendrix has good performance in general. However, there is room for improvement. The voice control system requires an Internet access to send the speech to Google Speech Recognition web service. There is an offline speech recognition system called CMU Sphinx. We have not been able to install it on the Yocto Linux image. We can try installing Debian on the Intel Edison and then try installing CMU Sphinx.

Since we are just using a simple one bit signal to control the press and release of solenoid, every time the solenoid presses on the string, it causes the string to vibrate and create noises since the speed of it is not in control. In the future, we can use PWM signal to slow the pressing with a low duty cycle.

The right hand picks strings by letting the servo rotate its arm for a small angle. Such picking motion is different from how human guitar players pick the strings. Humans not only rotate their wrist but also move their whole arms. The servos on the right hand picks the strings at a right angle while humans vary the angle to have different expressions. For instance, if the pick is tilted during the contact with a string, the sound has less attack leading to softer tone. Moreover, humans can change how hard they pick and emphasise different notes while strumming. In the future, we can focus on how to mimic the human picking.

The right hand is the bottleneck of the maximum speed our robot can achieve. The timing analysis in section V shows that the shortest note duration our robot supports is 0.2 second. Most guitar songs do not have notes shorter than 0.1 second. If we add two servos per string, we can double the speed and our robot will be capable of playing many more songs.

VII. Conclusion

This project gave our team great experience in system design, electrical design and mechanical design and control systems. By implementing our left hand, right hand and control systems, we were able to realize a reliable robotic system, that is able to successfully play songs on the guitar in rhythm and interact with human beings through voice commands.

Acknowledgment

The authors would like to thank Professor Ankur Mehta.

References

[1] M. Starlin *Guitar Parts*, http://www.markstarlin.com/guitar/2014/8/28/guitar-parts, Aug 2014.
[2] R. Robinette *How Guitar Tube Amplifiers Work*, https://robrobinette.com/HowAmpsWork.htm.
[3] J. Lange *FolkBox: more twang, more solder*, http://itp.nyu.edu/ jl4554/blog/?p=324, Apr 2012.
[4] J. Hopson *ROBOTIC PLAYER GUITAR ROCKS OUT ON ITS OWN*, http://hackaday.com/2015/04/11/robotic-player-guitar-rocks-out-on-its-own/, Apr 2015.
[5] ALSA Project *Advanced Linux Sound Architecture (ALSA) Project Homepage*, http://www.alsa-project.org.
[6] eSpeak *eSpeak text to speech*, http://espeak.sourceforge.net.
[7] G. Hall *Python MIDI*, https://github.com/vishnubob/python-midi.
[8] Intel *Intel Edison Kit for Arduino Hardware Guide*, Feb 2015.
[9] Intel *Internet of Things: Using MRAA to Abstract Platform I/O Capabilities*,
[10] I. Baek and W. Kaiser *Intel Edison Tutorial 5: SPI, PWM, and More GPIO*