The Pick Pocket Tuner

Lucas Grayford, Paul Grayford, Luis Vargas, Jamie Henry

University of Central Florida, Department of Computer and Electrical Engineering, Orlando, Florida, 32816, U.S.A.

Abstract **– The objective of this project is to design an automatic tuner for stringed instruments, mainly guitars. Our project is designed to be utilized for all skill levels of musicians, from beginners to skilled players, to quickly tune their instrument regardless of whether they are on stage in a loud environment, or in a closed practice room. The user should be able to tune their guitars in an accurate and fast manner to the correct frequency.**

I. INTRODUCTION

To tune a stringed instrument, a peg is used for tightening or loosening the string threaded into a cylinder. This peg usually consists of a worm drive and the peg head or key. The worm drive consists of a screw-like gear and a spur gear to create a smaller volume object with similar gear ratios. Some players also use peg locks to help ensure that the peg will stay in place to help reduce loss in tuning on the string.

The purpose of tuning a stringed instrument is to create sounds that illicit emotions in those who hear and play it. Tuning the strings to the correct pitches ensures that all notes that can be played on the instrument will make the correct sounds and will create captivating music. To ascertain that the tuning of a stringed instrument is correct, the user would match the frequency of the plucked string to a known frequency, such as a tuning fork or another audible device, by ear. However, tuning stringed instruments by ear could lead to incorrect tuning and incorrect pitches across the strings. This has led to the creation of automatic guitar tuners to eliminate human error and make the tuning process faster.

Sensors are used by these automatic guitar tuners to help eliminate the human error. These include vibrational and microphone sensors that pick up the frequency of the plucked string. This gives an accurate reading of the string's frequency that can then be compared to the tuning specifications. Our group's Pick Pocket guitar tuner uses a Piezo vibrational sensor that will sit in the housing of the product to pick up that string's frequency. This frequency, read in as analog voltage, will then have to be transformed into the frequency domain to read its value. To achieve this, the Fast Fourier Transform is used. The Fast Fourier Transform was chosen as opposed to the Discrete Fourier Transform solely on computation time. Since time is valuable to our automatic tuner and is one of the deciding factors among competitors.

II. TUNING BACKGROUND

For a guitar, the tuning standards have been established for some time, and many people will argue about which frequency to use for a specific intonation. When referring to standard tuning on a guitar it is usually EADGBe tuning, which is based on the pitch standard of A440. Breaking this down, the thickest and lowest tone string is the E, followed by A, all the way until you get to the thinnest which is the e string, on a traditional, right-handed guitar it would be that the strings go from left to right, from thickest to thinnest.

The A440 standard is referred to as the Stuttgart Pitch, and it is corresponding to the 440Hz frequency for the A note above the middle C in a Piano. This A440 pitch is standardized by the International Organization for Standardization as ISO 16, and as such it is used as a reference frequency to calibrate the equipment and instruments to be used in tuning. EADGBe was chosen as the 'standard' tuning for a guitar, mostly due to playability across keys and scales, making the arrangement of music to be the most pragmatic in this tuning. With that said, since we are operating under the generally accepted standardization of EADGBe tuning, with accompanying Scientific Musical Notation.

Tuning	Frequency	Units	Scientific Musical Notation
E	82	Hz	E2
A	110	Hz	A2
D	147	Hz	D3
G	196	Hz	G3
B	247	Hz	B3
e	330	Hz	E4

Table 1,Frequencies for Guitar Standard Tuning

Tuning any stringed instrument, is essentially following math formulas. For the purpose of our project, as can be surmised from the statements thus far, a guitar was chosen as the basis for the determination, testing, and conclusion of our project. This ties in easily, specifically, to the tuning methodology since a guitar is divided easily into the different components that allow the tuning to be changed on the system significantly quicker than with other instruments due to our group's familiarity with the instrument. [1]

The process for tuning a guitar is broken down into using the tuning pegs that correspond to the order of the strings previously mentioned, that are typically located at the headstock of the guitar. Each tuning peg is turned to either tighten the string that is attached increasing the tension and thus pitch, or loosening the string, decreasing tension, and thus decreasing the pitch of the guitar. This changing of the pitch, through the vibrations through the strings and into the body of our device is what will aid in determining the frequency, octave, and cents necessary for final tuning. [2]

For facilitating the discussion, the previous three terms are defined as such:

a) Frequency – this is the vibrational frequency carried from the moment the string is plucked, through the body, and detected by the sensor on the device, or it is the signal picked up by the electric pickups on a guitar and then the signal is carried onto the device to determine what frequency is picked up.

b) Octave – An octave is an interval in which the frequency of vibration of a higher note has twice the frequency than a note who's lower. For example, the standard pitch A above middle C has a frequency of 440 Hz, and so an octave above this vibrates at 880 Hz, and the octave below this vibrates at 220 Hz. This doubling/halving is consistent across all notes.

c) Cents – A cent is a unit of pitch that is based on an octave. Between each octave there are 1200 cents, and so the division of each interval between octaves is used in determining the semitones with different subdivisions to determine the notes that lie within that octave. [3]

An octave has a span of twelve equally tempered semitones, which is why we can determine that in an octave there are 1200 cents. Our approach to determine the accuracy of tuning based on our inputs utilizes both concepts, hand in hand. Our system utilizes a table of known values that would serve as a basis to compare to, to then calculate the number cents the device needs to adjust the string tension by.

To determine the cents (¢) we use Formula (1).

$$¢ = 1200 \times \log_2\left(\frac{Ref.Freq.}{Input\ Freq.}\right) \ (1)$$

Where $Ref.Freq$ is our reference frequency from the table of known values used, and $Input\ Freq.$ is the input frequency originating from the guitar after being simply processed by our software. [4]

The calculation of an octave comes into play when there are multiple strings, at different octaves, being tuned. Since they are still the same note we don't need to change the overall workings of the system, just adjust the frequencies in order to read that they're the same, but at a different pitch.

$$Octave = \log_2\left(\frac{Ref.Freq.}{Input\ Freq.}\right) \ (2)$$

Similarly, we use the known $Ref.\ Freq.$ and calculate the ratio by dividing with the $Input\ Freq.$ and taking the logarithm with a base of 2 to determine if the $Input\ Freq.$ is an octave of the $Ref.\ Freq.$, using these two equation in conjunction allows the system to determine how much to spin the motor based on the accuracy parameters defined.

III. SYSTEM COMPONENTS

The system is best described as having multiple key components that are shown in Figure 1 below.

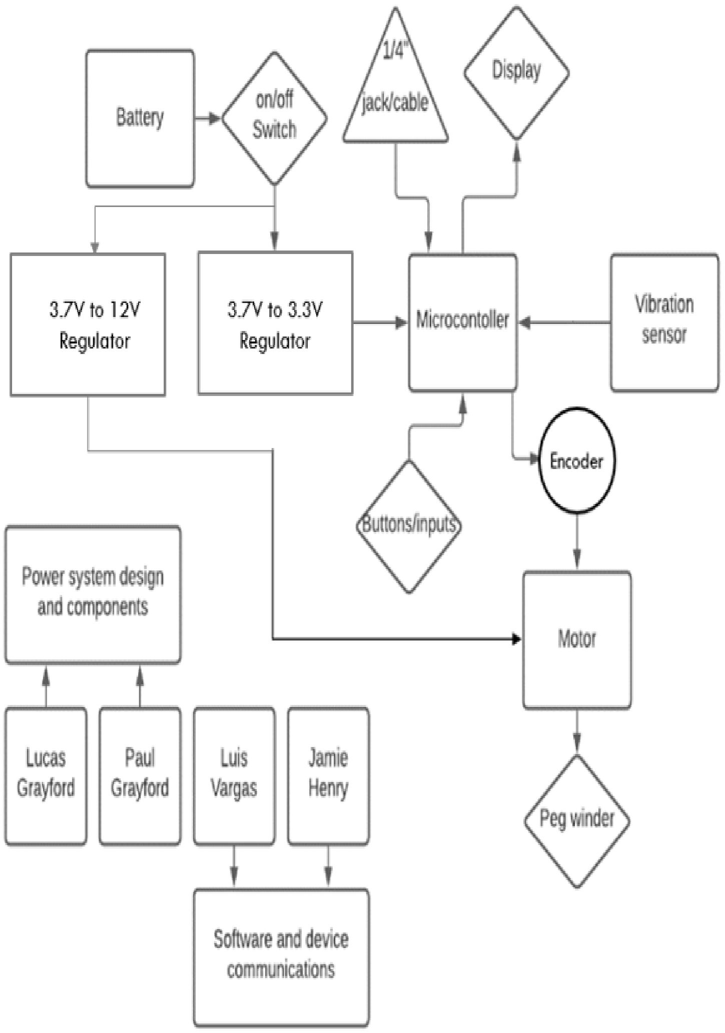


Figure 1, Hardware Block Diagram

A. ESP – 12E Module

The main functioning component of the project is the ESP8266, with the ESP12-E as the main module that is utilized to integrate all the components with. This module was chosen for its memory capacity, low power consumption, and WIFI connection capabilities. The Arduino IDE and the built-in compiler are used in conjunction with a burner fixture test board to program, develop, and test the individual ESP12-E module.

B. Flash Memory

All the necessary code for operation of the final prototype, the miscellaneous GUI elements, and stored frequency libraries are stored on the module's internal ROM/RAM. As such, the libraries that were ultimately used, the software that was developed and integrated into our final project was all chosen after determining that the memory consumption when loading/executing the instructions for the entire program could be run from the available on-board memory contained within the ESP12-E module.

C. DC Brushless Motor

We used a brushless DC motor that operates at 12V that can function up to 159 rpm. The motor comes with an encoder so there is no extra need for a on board driver. The holding torque is .231 Nm which falls into the range that is needed to be able to turn. The motor is compatible with the Arduino software that is being used.

D. 3.7V LiPo Battery

For powering the whole system, a 3.7 Volt Lithium-ion polymer 1200 mAh battery is used. The overall system is going to need power an OLED screen, processor, driver and motor. Since the battery is going to be a DC power supply there is going to be a way to recharge the battery after it is depleted. The charging is USB-C charging since it is commonly found along most phone charger types so if the charger for the device is ever lost a phone charging cable will make a good substitute for it.

E. ¼" OP jack

The Pure Tone output jack has two live connection pins and two ground connection pins that connect to the cable coming from the guitar, mainly for electric. It also has an extra ground pin to complete the connection in the circuit it plugs into. The advantage of having these two extra pins is added reliability, accuracy and longevity of the product.

F. TFT Display

The TFT display screen is a 1.3" LCD and is 240x240 pixels. The screen features a clock and data pin to set up an SPI connection with our ESP8266. It also comes equipped with an Arduino integrated circuit driver with a documented library for implementing graphics in the Arduino ide to help create our graphical user interface. The screen is used for an interactive display for the user to swap between settings for the tuner.

G. Piezoelectric sensor

This Piezoelectric sensor will be directly connected to the MCU to be able to utilize it as the main point of contact at which we will begin to test the capabilities of vibrational sensing and thus begin our signal processing. For the sensor it communicates to the MCU through the analog input pin that the board has, as these are devices that generate an analog 'signal', vibration, to be processed into what we ultimately need to be able to accomplish what Pick Pocket Tuner set out to do.

IV. SYSTEM CONCEPT

The device could be understood as a whole system with the following flowchart showing in detail the operation procedure. Figure 2 shows how the device carries out all the functions and computations that are necessary to achieve the desired result of this system.

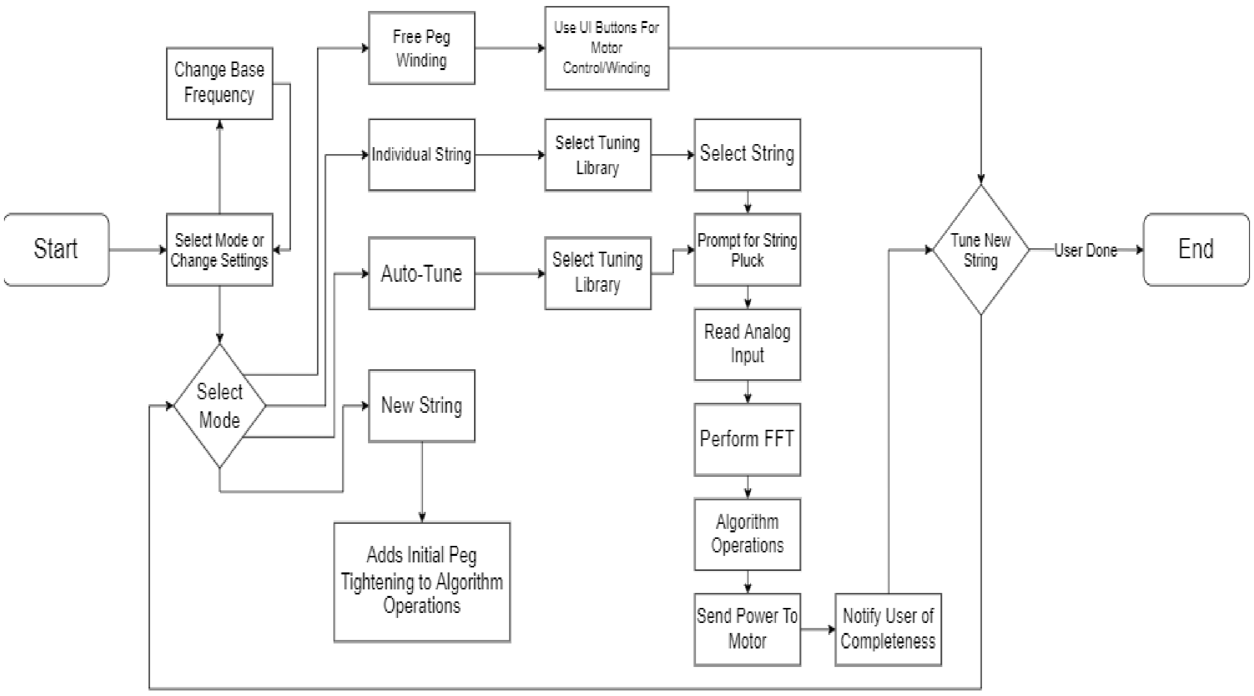


Figure 2. Software flow of system

.

V. HARDWARE DETAIL

Various components were utilized for this project. Between all of the components, the electrical engineering students design two printed circuit boards and a housing that would enclose the entire device in a small hand-held product that could fit in a pocket.

A. *Microcontroller*

The microcontroller board contains the esp12-E module and all the connections for the peripherals. The main goal for this design was to design a board that would be smaller in size to be able to fit in the palm of your hand. This was accomplished after creating smaller the board needed to be redesigned 3 different times, and on the third model, everything but one component was working how we wanted it to work. Some components that are included on the board are: a low dropout regulated that supplies 3.3V from the battery with an efficiency of 78% to 91%, connections for the direction wires on the motor, connections for the screen, connections for the sensors, and buttons to control the interface. There was no need for pins that create a serial connection to the programming setup since we can program the device using over the air transmission. This we very useful when the device was completely assembled, and we were able to debug the device without disassembling the entire project every time we wanted to flash code. The only issue that we ran into without circuit board was the incorporation of a buzzer. The buzzer was connected to GPIO0 and was not allowing the board to boot properly. This as the only peripheral that we were not able to fix in time for our final presentation. [5]

B. *12V Regulator*

The regulator that we designed contained twenty-nine components and utilized an LM25118 controller to regulate to the correct voltage. The input range that the regulator was designed for 3V to 4.2V and the output was designed for an output of 12V and

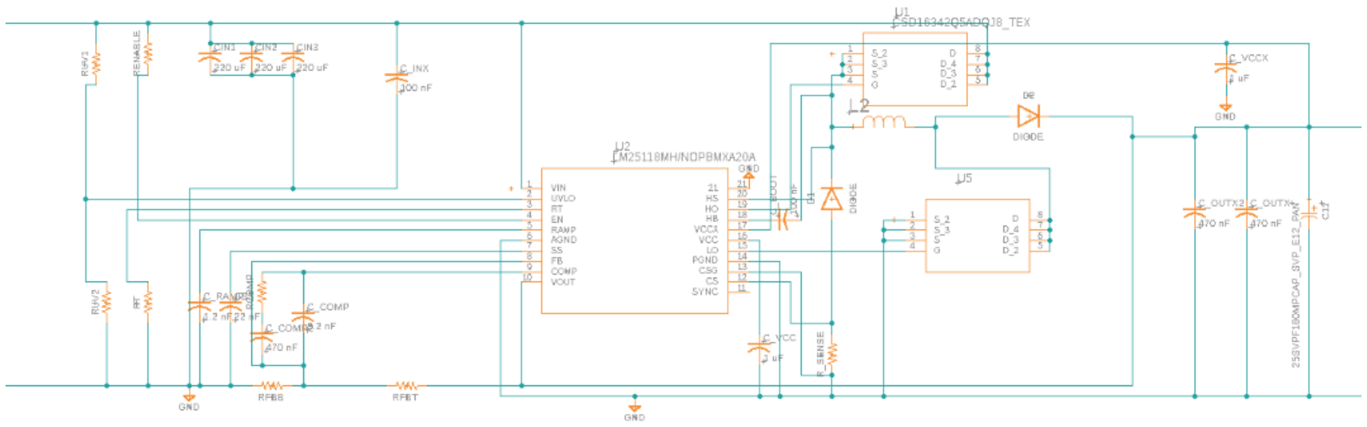


Figure 3, 3.7V to 12V Schematic

700mA. 700mA was the max amperage that the motor would draw which is the reason the regulator was designed this way. There were four iterations of the board, and the final design was very close to working. There were some issues with the footprints that were gathered from UltraLibrarian that were causing the board to have an internal short that was very difficult to find. It was finally found on the final iteration, however there was not enough time to order a final PCB that was expected to work. We substituted the regulator for a premade Buck boost converter that was found online and used for most of our prototyping. [6]

This provided enough current ad voltage to power the motor to full capacity.

C. *Product Housing*

The product housing was designed using Fusion 360. The reason nwe chose to use fusion 360, was that it already contained all our PCB files and were able to model the housing around the 3D models of the PCBs. Paul Grayford modeled most of the housing and Lucas was able to 3D print three iterations of the housing. The final iteration can fit in one hand, and fits all of the components for our project very securely.

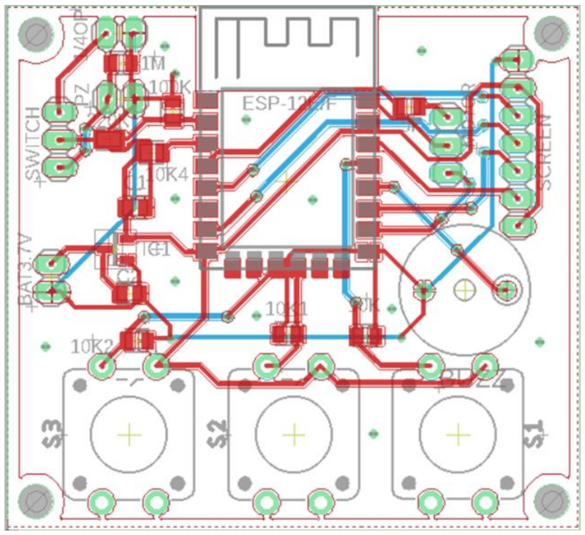


Figure 4, PCB Schematic

VI. SOFTWARE DETAIL

Before the software portion of the entire system can be thoroughly explained, we must establish that the purpose for the whole project is to control the motor, screen, and take the input from a piezo sensor to tune a guitar accurately. The whole crux of this project depends on the proper execution and determination of all the pieces compose a “musical note”. To define this, we reference the formulas defined in the Tuning Background section, and then based on the expected response our designed GUI will relay the necessary information.

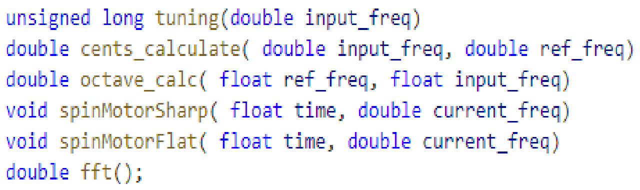
A. *System Firmware*

The software is developed entirely in Arduino which utilizes C/C++ dialect. Our microcontroller communicates with the TFT display via SPI, with the brushless DC motor via PWM, and the buttons via interrupts.

The Wi-Fi module on the microcontroller was used for over-the-air programming. This was required for any changes to the code that were made after the ESP12-E device was soldered onto our PCB. The libraries necessary for making this WIFI connection were EEPROM, ESP8266WiFi, ESP8266WebServer, ArduinoOTA. The ESP12-E is setup to connect to a certain Wi-Fi for programming purposes and if the connection is failed it will create its own web server. This web server can then be connected to for reconfiguring the connections. This is vital, as a connection to the microcontroller can be always made even if initial Wi-Fi connections fail.

The graphics drivers necessary for communicating with the display were the Adafruit_GFX and Adafruit_ST7789 libraries. These libraries aided into the creation of the GUI for the software with their built-in functions for drawing shapes and displaying text.

The utilization of the formulas defined by section II above are paramount in determining that the software we have implements is working appropriately. Through the previously mentioned hardware components we take the raw input from the source and passes the input through an open-source Fast Fourier Transform implementation for us to obtain a frequency value to use. Once this is done, we then take that input value, calculate the cents and octave to ensure where the accuracy of the sensor reading lies, all the while relaying the necessary information to turn the motor in the correct direction and for a pre-determined amount of time. This turning of the motor is what adjusts the tuning peg tension, and this whole process repeats while still accepting the incoming input from the person strumming the guitar. Throughout this entire process, the GUI updates consistently to relay the information necessary to the display to show the user whether the string is at the desired frequency/note. This above is the majority core of functionality of the software system and the functions involved in this process are as follows:



```
unsigned long tuning(double input_freq)
double cents_calculate( double input_freq, double ref_freq)
double octave_calc( float ref_freq, float input_freq)
void spinMotorSharp( float time, double current_freq)
void spinMotorFlat( float time, double current_freq)
double fft();
```

(1) The tuning() function takes the original transformed input frequency values and calculates the cents between the current frequency and the target frequency for the current string being tuned. Based on the cents value the spinMotor() functions are called. Then the new current frequency is calculated, and the new current cents value is calculated to start the process over until the string is tuned.

(2) The cents_calculate() function is used to calculate the cents of the two frequencies passed in, this function is a direct representation of the formula that was discussed in section II. A negative cents value is returned if the input frequency is higher than the target frequency and a positive cents value is returned if the opposite is calculated. The polarity of this value is essential for tuning as the motor will turn the peg in the right direction tightening/loosening the string.

(3) The octave_calc() function is used to calculate the octave in which the input is in. This allows the firmware to be able to determine that at different octaves the note is still the same, for example, an E note with frequency 82.41 Hz is also the same E note if it has double the frequency at 164.82 Hz. This allows us to filter out any errors and improve the efficiency in which we tune.

(4) The spinMotorSharp() function tightens the peg resulting in a higher frequency. This function is called when the cents value returned is positive, therefore being les than the target frequency. Another input for this function is the time duration as it varies due to the magnitude of the cents value as the higher the value, the larger difference between the frequencies.

(5) The spinMotorFlat() function works in the same way as the spinMotorSharp() function. However, it spins the motor in the opposite direction as the cents value returned is negative and therefore loosens the string to a lower frequency.

(6) The fft() function reads in the analog voltage input from either the ¼" jack or the piezoelectric sensor and stores the values into a sample array. This sample array is transformed via the arduinoFFT library and returns the peak frequency value in hertz.

Using these above functions, in one manner or another, through various comparisons is what achieves the automatic tuning based on the parameters we've defined. Throughout the entire firmware, the function fft() is used to convert the raw input data from the guitar, via either the ¼" input jack or the piezoelectric sensor, to a usable frequency in hertz. The cents_calculate and octave_calc functions are implemented using the same methodology described in the Tuning Background section above using the corresponding formulas.

VII. RESULTS

After much debugging and testing, we were able to deliver a product that could tune a guitar in sixty seconds. There were a few design features that did not play out as we expected, like the accuracy of picking up the vibrations. When the device worked well, the accuracy was near perfect, but because of some issues with the housing and other unforeseen issues, the value that was being read was off at some points. We believe that this was due to the housing design. The piezoelectric sensors were placed farther away from the motor than we previously thought, and because of this, we were not able to pick up the signal as clearly as we would have liked to.

Another design change that could have been implemented was to fine tune the Fast Fourier Transform function. Overall, function performed with accuracy, but there was still some margin of error, especially on the G string. The difficulty of this string was that it always gave a lower reading than what it was. Besides this issue, the function performed well enough to get us into the range we needed.

Finally, the last issue we would have liked to solve was the bouncing of the center button on our device. This button would bounce the worst, and we tried increasing the resistance on the connection, and trying to add delays in the software, however, nothing we tried fixed it completely.

Other than these design issues, we were proud with how the final product came out. If the product was working smoothly, we were able to tune a guitar quickly and accurately as well as meet or exceed all of the specification for our product. The senior design team learned a substantial amount from this project and learned many lessons that will translate well, once our team enters the workforce.

VIII. THE ENGINEERS

Luke Grayford – Electrical Engineer

Graduating under electrical engineering comprehensive track and pursuing the next steps toward a future career with any willing engineering company or firm.

Paul Grayford – Electrical Engineer

Graduating as an electrical engineering student, Paul will take his talents to Lockheed Martin and work at the Kennedy Space Center for the commercial project Orion as an electrical test engineer.

Jamie Henry – Computer Engineer

Graduating as a computer engineering student, Jamie has accepted an offer to work for SkillStorm as a Software Engineer in May.

Luis Vargas – Computer Engineer

Graduating as a Computer Engineering student, Luis will be joining Boeing in St. Louis, Mo as a Software Engineer working on the F-15 Mission Systems.

IX. References

[1] Dan. (2019, October 2). *What are the guitar string frequencies?* Fret Success - Guitar Tuition. Retrieved April 19, 2022, from https://fretsuccess.com/what-are-the-guitar-string-frequencies/

[2] Sengpiel, Eberhard. “Cents to Frequency Ratios Conversion and Convert Frequency Ratio to Cents - Sengpielaudio Sengpiel Berlin.” *Conversion of Intervals - Cents to Frequency*, www.sengpielaudio.com/calculator-centsratio.htm. Accessed 19 Apr. 2022.

[3] Rod Nave, Carl. “The Use of Cents for Expressing Musical Intervals.” *The Use of Cents for Expressing Musical Intervals*, 1999, hyperphysics.phy-astr.gsu.edu/hbase/Music/cents.html

[4] Suits, B. H. “Making Sense of Cents.” *Making Sense of Cents*, pages.mtu.edu/%7Esuits/cents.html. Accessed 19 Apr. 2022.

[5] Microchip. (2020, February 28). MCP1700 low quiescent current LDO data sheet. Retrieved April 19, 2022, from https://my.mouser.com/datasheet/2/268/MCP1700-Low-Quiescent-Current-LDO-20001826E-737536.pdf

[6] Power designer. (n.d.). Retrieved April 19, 2022, from https://webench.ti.com/power-designer/switching-regulator/customize/12