Automatic Piano Tuner - Final Report

By

Joseph Babbo

Colin Wallace

Riley Woodson

Final Report for ECE 445, Senior Design, Spring 2022

TA: Zhicong Fan

4 May 2022

Project No. 49

Abstract

The Automatic Piano Tuner is a hand-held which allows users to easily tune a piano by selecting a frequency, given as a list of notes, and playing a note on the piano, from which a motor automatically moves the note's connected piano pin to match up the two frequencies. It includes a fully functioning audio acquisition subsystem, control subsystem, battery subsystem, power regulation subsystem, electric motor drive subsystem, display subsystem, and button subsystem.

Contents

1 Introduction

1.1 Purpose

1.1.1 Problem

In order to maintain accurate pitch, all stringed instruments must undergo periodic tuning. For most instruments this process entails a basic hand tightening of tuning pegs. For a piano however, the tuning process is exceedingly difficult, typically requiring the usage of specialized tools, the hiring of expert tuners, and 1 - 2 hours of time. Most piano tuning services will charge customers $100 - $200 per tuning session[1], which could be required anywhere from once a year to several times a year depending on customer needs. Due to the inherent difficulty in piano tuning, customers are forced to either accept the costs and time required for professional turnings or limit the number times they tune their piano so as to save money and time.

1.1.2 Solution

As a solution to this problem, Team 49 has created the Automatic Piano Tuner. The Automatic Piano Tuner has sought to correct the technical skill, high cost, and time consuming processes that normally come with tuning a piano. Through an intuitive interface and built in frequency detection and correction algorithm, the device makes tuning quick and easy. And with a material cost of $150, the Automatic Piano Tuner breaks even on investment after only one use.

1.2 Functionality

To guide the design of the project, three high level requirements for functionality were created. The first was that the tuner be capable of distinguishing between signals separated by 1.6 Hz.[2] The purpose in this was to ensure that the device was capable of distinguishing between every note on a piano, the lowest of which being separated by 1.6 Hz. The next high level requirement was that the device be capable of producing 9-14 Nm of torque[3]. Research shows that piano pins are designed to resist forces up to that range, meaning the tuner would need at least equal torque to overcome the force of the pin. The final high level objective was that the tuner be capable of tuning all 88 notes on a piano[4]. If the piano tuner was to be considered successful, it would need to have the full functionality of tuning every not on a piano.

1.3 Subsystem Overview

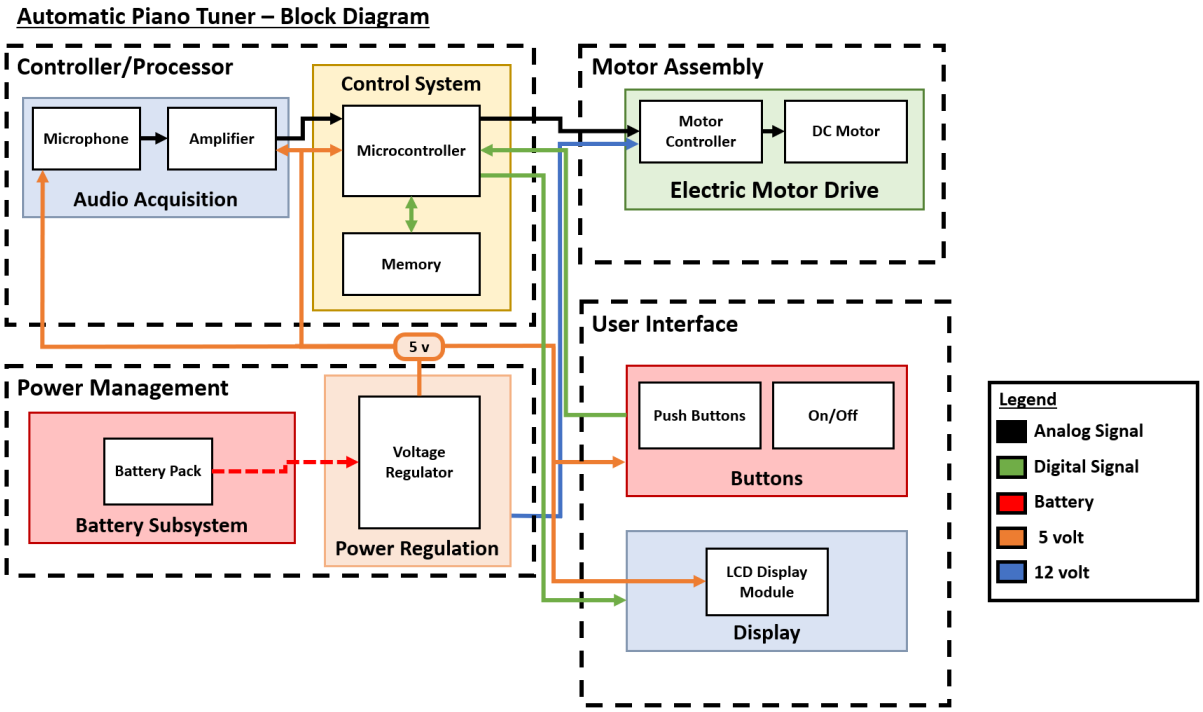

Figure 1: Block diagram for the Automatic Piano Tuner.

The design consists of four main areas of focus which represent the major divisions of functionality in the project as can be seen in figure 1. These abstract systems get further broken down into more specific subsystems. The power management portion is made up of the battery and power regulation subsystem. These are responsible for supplying a smooth 5 Volt line to digital elements on the device, as well as delivering a 12 Volt, 4 Amp power supply to the motor. This system is capable of dampening any disturbances caused by the switching of the motor. The controller/processing system is broken up into audio acquisition and microcontroller subsystems. The audio acquisition system converts audio waves into voltage waves which are then sampled by the microcontroller. The microcontroller can then determine the motor adjustments needed for tuning the note being played. The motor assembly consists of a motor controller and motor capable of delivering enough torque to turn a piano pin. Finally the User interface consists buttons for control and a LCD for displaying the current note and frequency. All digital subsystems are controlled centrally from the microcontroller and the motor controller is managed via PWM digital signal to emulate an analog response.

2 Design

2.1 Physical Design

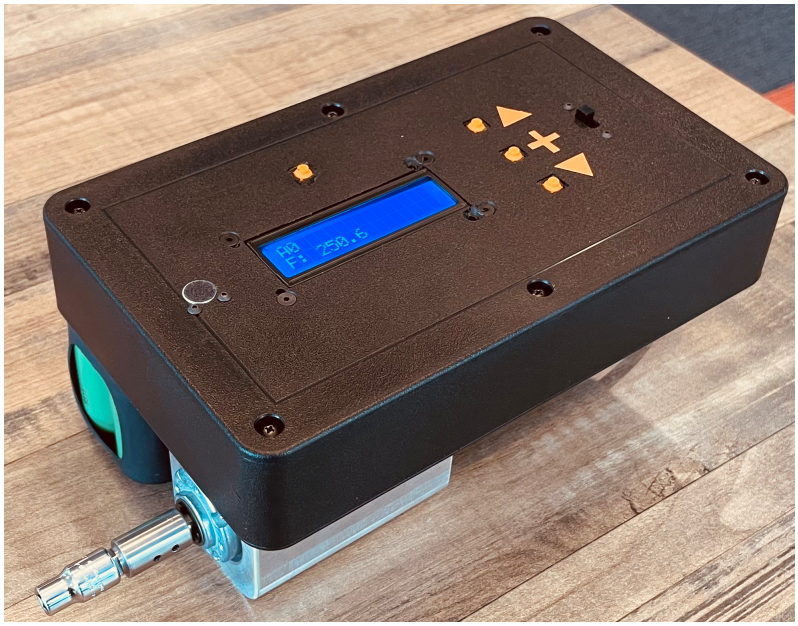

Figure 2: 'Final physical design of the piano tuner'.

The original design was imagined as a cordless hand drill as consumers are already well versed in the ergonomics and use of such a product. Although the final design strayed from this plan, the idea of making a user friendly device was still upheld. The device is now a black box, easily gripped by two hands, with a display for selecting current note to be tuned, and several buttons for navigating the display as seen in figure 2. It also contains an easy to reach trigger button above the display for activating the motor and a power switch for battery conservation when the device is not in use. The motor is mounted in a press fit metal case, and secured with set screws. The piano tuning bit is screwed into an adapter, which is secured to the motor driver with set screws. The battery is held in place with metal braces that screw into the black box. The device has a recharging cord that hangs freely.

2.2 Subsystem Overview

2.2.1 Audio Acquisition Subsystem

The Automatic Piano Tuner uses an Adafruit 1063 microphone with built in amplifier to measure audio. The microphone has a ground, Vcc, and data port which is connected to the ATMega2560 PB0 pin as can be seen in figure 3. The microphone works by centering the zero-crossing of waves at half of the received Vcc (5 volts) signal. This is highly compatible with the microcontroller ADC, which measures values between 0 and Vcc. The microcontroller samples the microphone at 78 kSa/s and 8-bit resolution. These sampled points can then be used to estimate the frequency of the received signal. The sample rate was chosen to be

significantly above the Nyquist Sampling Rate, but not too high to minimize interrupts to the main control loop.

The frequency detection is executed via period estimation scheme. The microcontroller compares two successive ADC samples and looks for a zero crossing and a positive slope. Once this has been found, a counter is start that is incremented with every new ADC sample. Once those conditions are met again, the total count is recorded and the counter is reset. The final counter value corresponds directly to the period of the current note being observe, and is used to calculate the frequency in equation 1. An example of the functioning period estimation is displayed in Figure 4.

$$\text{Frequency} = \frac{\text{SampleRate}}{\text{Period}} = \frac{78,000}{counter} Hz \tag{1}$$

The original plan for the frequency detection scheme was to utilize auto-correlation to detect frequencies. Due to a memory limitation of the ATMega2560 microcontroller used and large frames of data needed for auto-correlation, this plane was scrapped for the alternative described above. This detection method would have offered a highly accurate and noise resilient method for finding frequency.

This subsection passed all requirements as seen in Appendix A Table A.

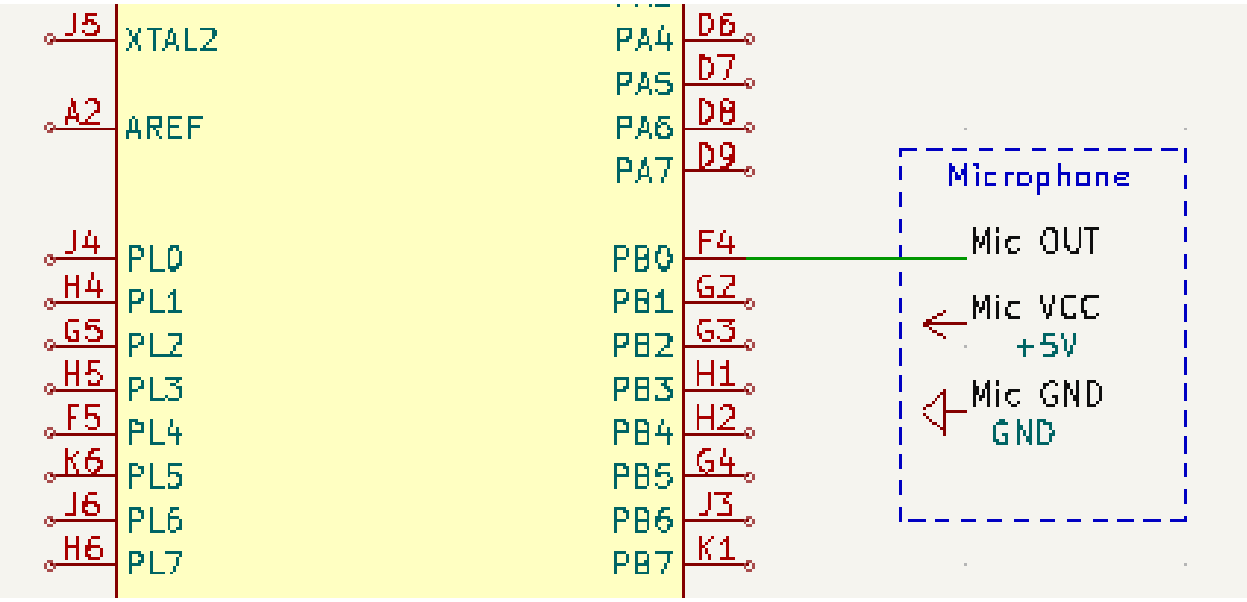

Figure 3: 'Audio Acquisition Subsystem' (connecting to 'Control System') circuit schematic.

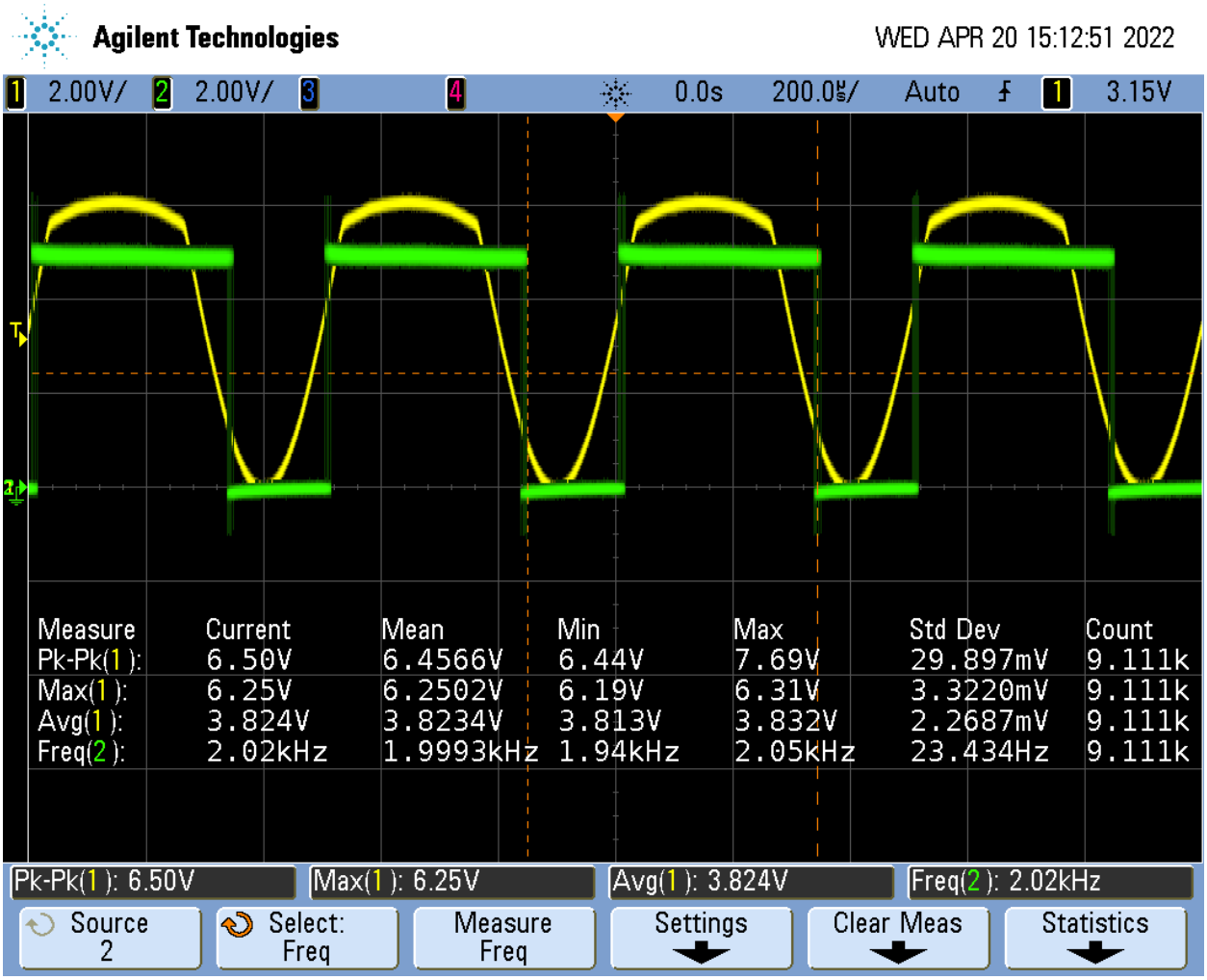

Figure 4: Period estimation test. Yellow test signal(2 kHz), green output detected frequency (1.9993 KHz).

2.2.2 Control System Subsystem

The Control System is composed of an ATMega2560, as seen in Figure 5, which is responsible for sampling and interpreting the microphone and button inputs, displaying relevant information to the LCD, and sending signals to the motor controller to make adjustments to the piano pins. The code works by running a setup function which assigns input and output pins and initializes the ADC. The rest of the code runs in a loop function which is called for the remainder of the execution. This loop function checks button states, updates the LCD, updates the current frequency estimation, and sends signals to the motor controller. The ADC inputs operate on an interrupt approach, being handled as soon as there is a new ADC sample to process.

The ATMega2560 was chosen initially for its high SRAM and program memory, 8 kB and 128 kB respectively. It was also selected for its high internal clock rate, which enabled the board to sample the microphone at a sufficient rate without the need of an additional external oscillator. Although this controller was sufficient

for the success of the project, future iterations would take advantage of a controller with larger memory space to facilitate more complex frequency detection algorithms.

See Appendix E and F for the main loop code and piano tuning library used to control the device and detect pitch.

This subsection passed all requirements as seen in Appendix A Table A.

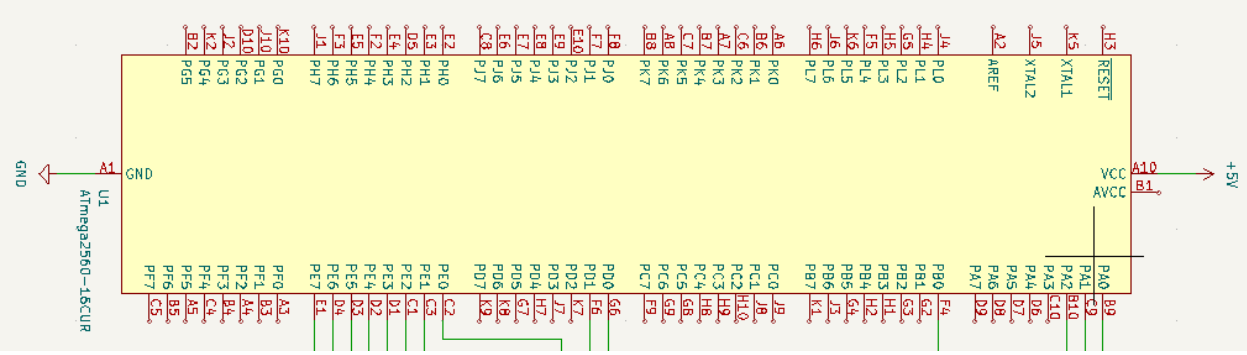

Figure 5: 'Control System Subsystem' circuit schematic.

2.2.3 Battery Subsystem

The Automatic Piano Tuner uses the AA Portable Power Corp MH-C5000B, a 12 volt, 5000 mAh battery that is capable of delivering 5 amps. The battery was selected for its ability to provide the motor with 4 amps, the calculated current needed to deliver desired torque. It is also capable of reaching a 2 hour life span, the average time it takes to tune a piano[5]. The battery has the functionality of being rechargeable via its provided 1.8 Amp "smart charger".

This subsection passed all requirements as seen in Appendix A Table A.

2.2.4 Power Regulation Subsystem

The power regulation system is responsible for smoothing the 12 volt signal from the battery and providing the digital components with a stable 5 volt signal. Ripples in the battery power are smoothed out by a 680 uF capacitor, a sufficient size to dampen the effects of switching the motor on and off. The 5 volt line is down converted from the 12 volt battery with a LM2596S-5 switching power regulator, as seen in Figure 6. This is capable of converting voltages of up 48 volts down to 5 volts, also taking advantage of a 33 uH series inductor, 220 uF capacitor, and diode to further regulate power.

This subsection passed all requirements as seen in Appendix A Table A.

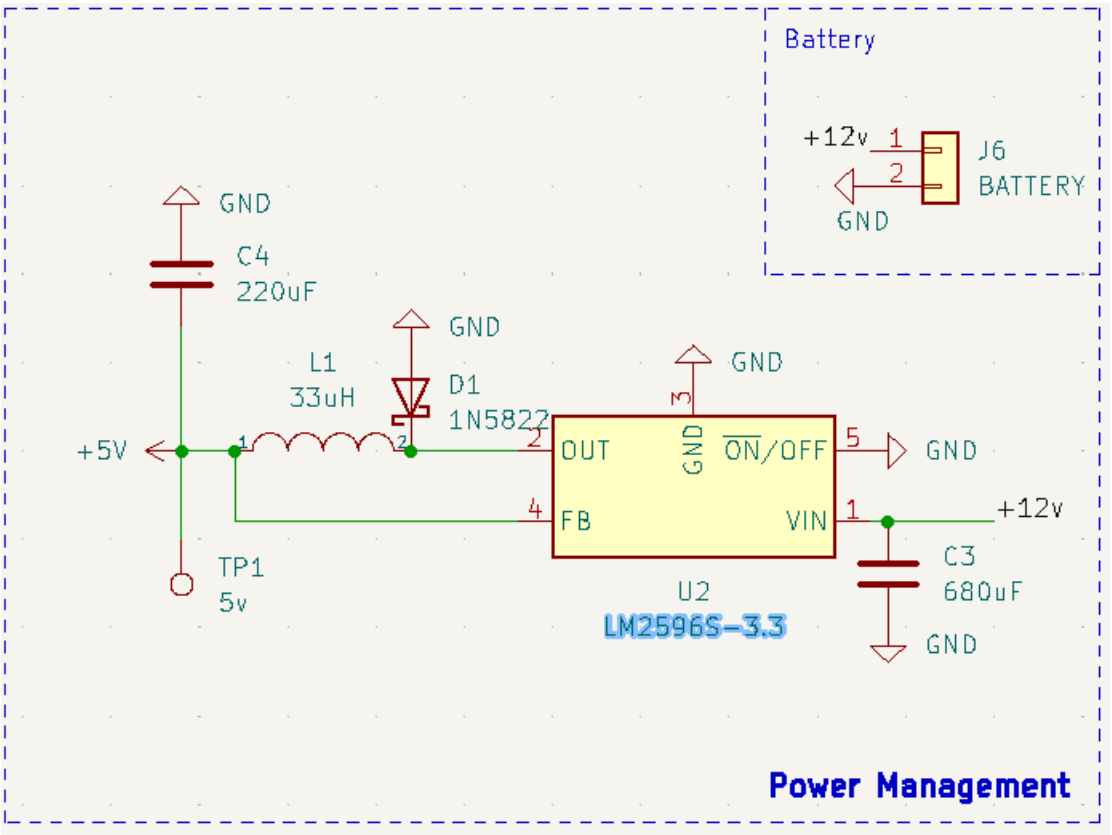

Figure 6: 'Power Regulator Subsystem' circuit schematic.

2.2.5 Electric Motor Drive Subsystem

The motor system consists of a motor, and motor controller. The motor is a planetary gear brushed DC motor made by ISL Products International. The motor performed excellently at 12v, and can handle up to 10A. It is able to produce up to 29.42 Nm of torque if necessary [6]. The motor controller is a HT BTS7960 H-Bridge Motor Driver. This controller is able to handle 12V, and up to 43A [7]. The motor controller requires a 5v signal to the left enable and right enable pins. The setup of these signals can be seen in Figure 7. It then accepts a PWM signal into either the left or right PWM ports, which it uses to determine the desired speed of the motor. The controller then sends a signal to the motor through the M+ and M- ports. It also requires connections to the 12V and 5V power and ground ports.

After some initial testing, similar to the motor RV scale method mentioned in appendix A, the motor produced 0.59Nm at 0.2A with the standard 12V. The relationship between current and torque is linear, such that the current could be calculated at our desired torque. Using equation 2 the desired current is calculated to be 4.06A. The desired current is well within the operating ranges of both the motor and motor controller.

$$I_{\text{desired}} = \frac{\tau_{\text{desired}}}{\tau_{\text{measured}}} * I_{\text{measured}} \tag{2}$$

This desired current was well above the 2A max of the original DRV8871 motor controller we used on our initial PCB. To fix this issue, the motor controller was changed to the HT BTS7960 currently used. This controller is not able to be mounted on the PCB, and was moved off board.

This subsection passed all requirements as seen in Appendix A Table A.

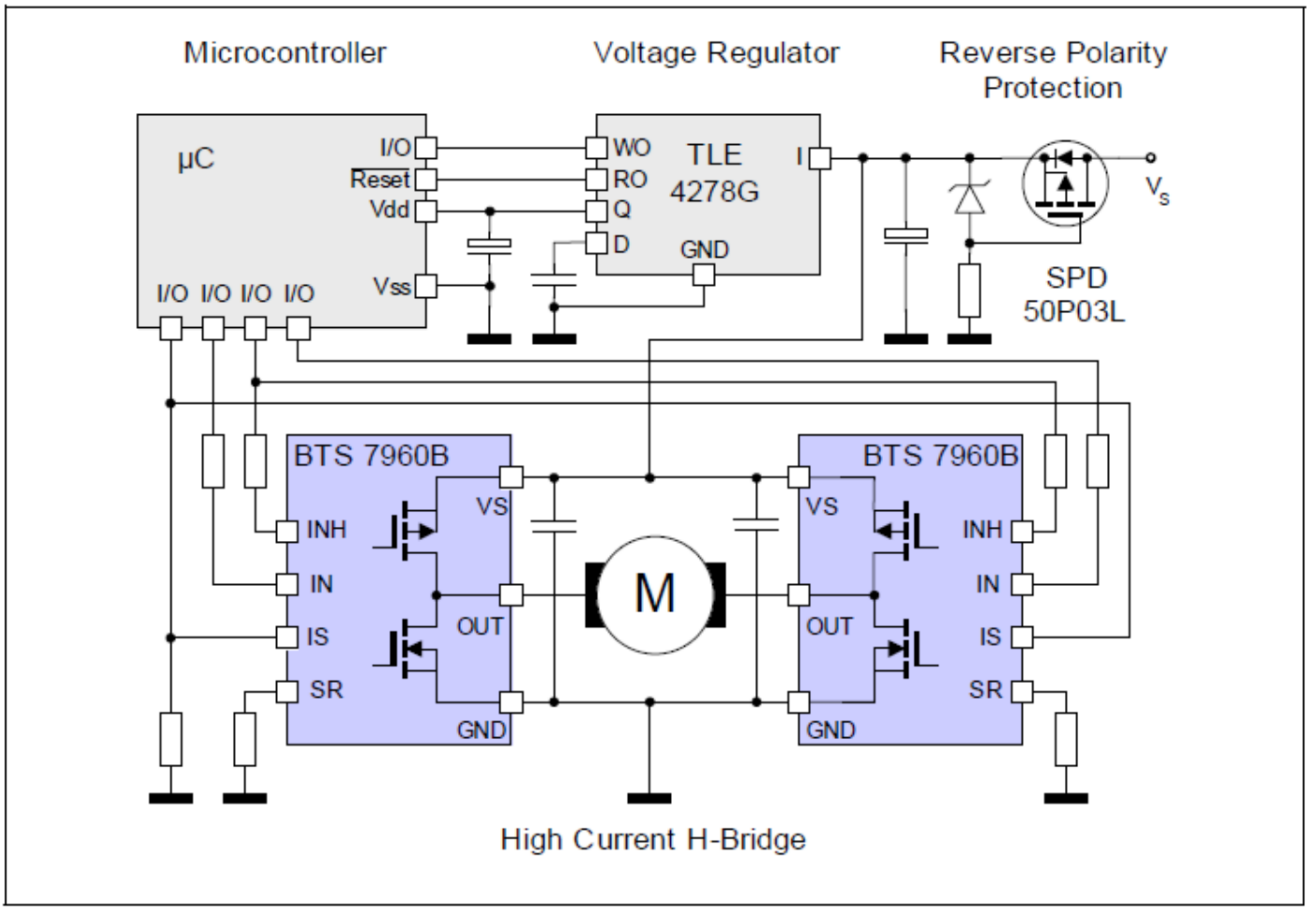

Figure 7: 'Electric Motor Drive Subsystem' circuit schematic.

2.2.6 Display Subsystem

The Automatic Piano Tuner uses a LCD to show the current note, including if the note is sharp, alongside the current octave number. The screen displays the note letter: A, B, C, D, E, F, G, the octave number: 0, 1, 2, 3, 4, 5, 6, 7, 8, and the sharp symbol when needed. The LCD is programmed through the microcontroller using the LiquidCrystal Arduino library. The microcontroller utilizes the LCD's VO, LEDA, LEDB, R/W,

RS, E, DB4-7, VDD, and GND pins, as seen in Figure 8.

The final design of the LCD system corrected a couple of flaws in the initial design. The VO pin, that controls the contrast of the letters on the screen, did not have the correct voltage requirement. Now the VO pin is connected to a voltage divider, that services its voltage requirement. The LEDA pin needed a lower resistor on its connection, as it was too saturated with the initial design.

This subsection passed all requirements as seen in Appendix A Table A.

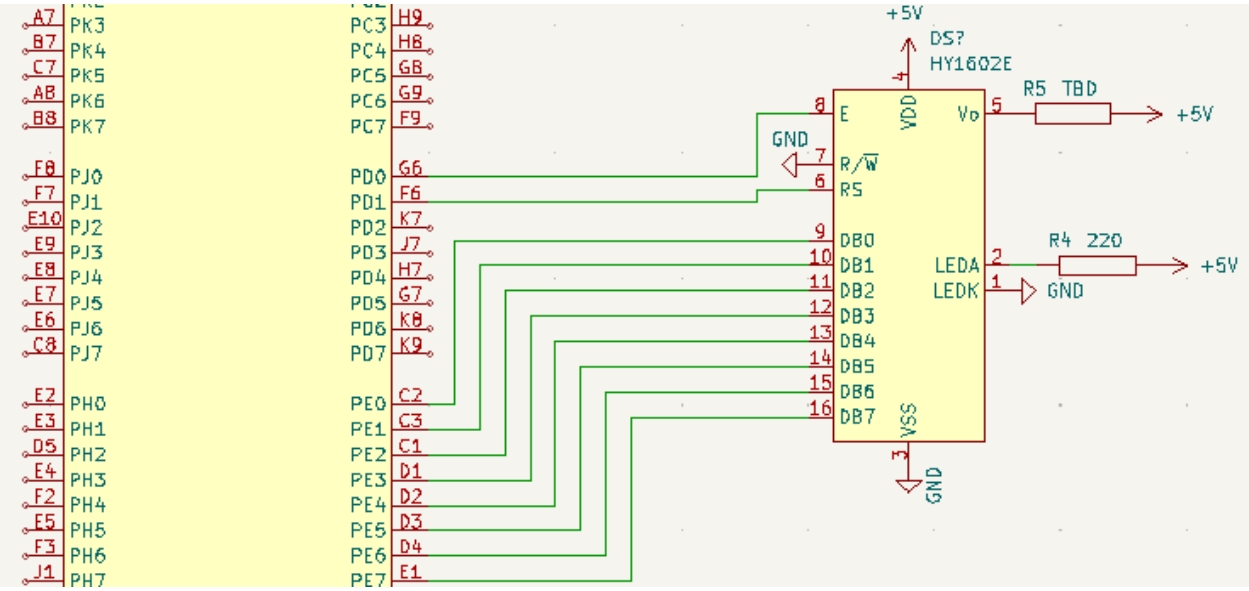

Figure 8: 'Display Subsystem' (connecting to 'Control System') circuit schematic.

2.2.7 Buttons Subsystem

There are three buttons on the Automatic Piano Tuner associated with moving up and down keys as well as changing octaves. These buttons are referred to as the 'note select' buttons, and can be visualized in figure 10. Each button is debounced using a resistor to GND, as seen in Figure 9. These buttons send a digital signal to the microcontroller which then alters the desired frequency. These buttons signals also tell the software to get the LCD display to correctly show the new key/octave. The up button moves the key up one note in order: C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B. The down button moves the key's in the reverse direction: B, A♯, A, G♯, G, F♯, F, E, D♯, D, C♯, C. If the current key is B and the up button is pressed, the key wraps around to C with the octave increasing by 1. Likewise if the current key is C and the down button is pressed, the key wraps around to B with the octave decreasing by 1. The octave button automatically moves the octave up by 1 to quickly change keys.

Additionally there is a trigger button that must be pressed to enable the motor rotation. This button is included in the design for safety, as it is undesired for the motor to move for every detected frequency. In the final design the motor will only rotate, based on frequency discrepancy, when the user wants it to do so.

A power switch is also included so that the user can put the device into power saving mode. This mode is meant to prolong the battery life of the device.

This subsection passed all requirements as seen in Appendix A Table A.

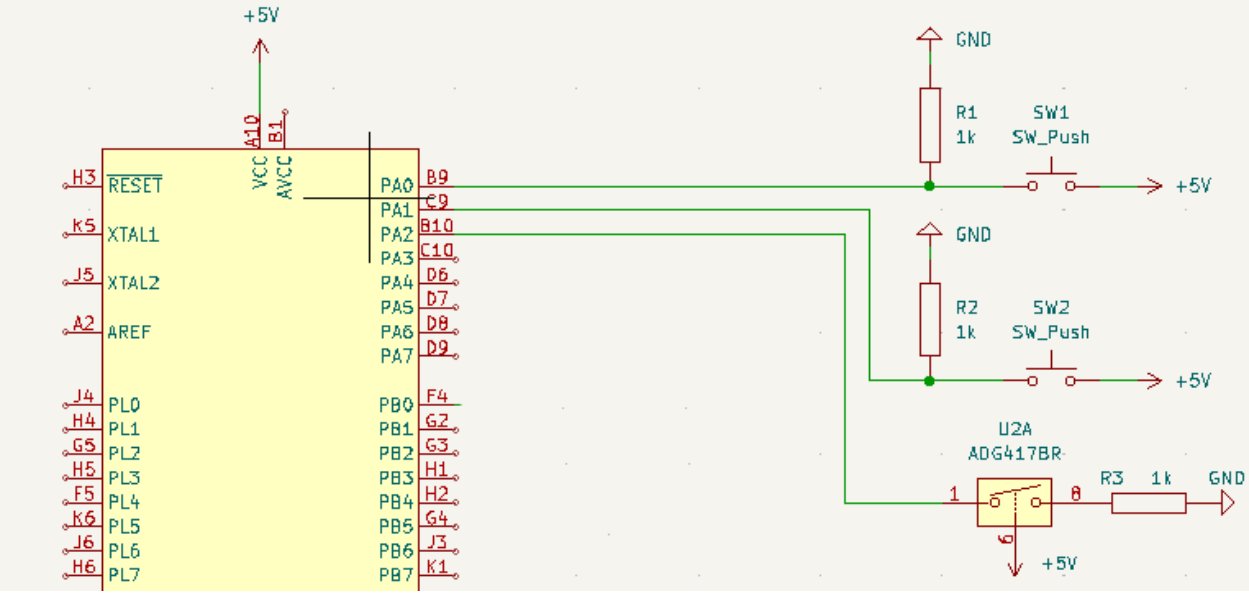

Figure 9: 'Buttons Subsystem' (connecting to 'Control System') circuit schematic.

3 Cost and Schedule

3.1 Cost Analysis

3.1.1 Labor

Taking the average salary of UIUC electrical and computer engineering graduates [8][1] alongside the relative population of the two majors [9][2], the average ECE graduate from UIUC makes an average starting salary of $89,796.

$$Avg.Salary = EEAvgSalary \cdot \frac{\#ofEEMajors}{TotalECEMajors} + CompEAvgSalary \cdot \frac{\#ofCompEMajors}{TotalECEMajors} \tag{3}$$

$$= \$79,714[8] \cdot \frac{870[9]}{2089[9]} + \$96,992[8] \cdot \frac{1219[9]}{2089[9]} \tag{4}$$

$$\approx \$89,796. \tag{5}$$

Assuming an average work week of 40 hours alongside 52 weeks in a year, gives an average hourly rate of $43.17 an hour. Assuming work on the project is done for 15 hours a week over 15 weeks in the semester, the per person labor rate is $\$43.17 \cdot 15 \cdot 15 = \$9713.15$. Multiplying this by a factor of 2.5x overhead multiplier we get $\$9713.15 \cdot 2.5 = \$24,282.88$. With three people working on the Automatic Piano Tuner the labor cost increases to $3 \cdot \$24282.88 = \$72,848.64$.

3.1.2 Parts

Table 1: Parts Costs

Description	Manufacturer	Part #	Quantity	Cost ($)
Schottky Rectifier	STMicroelectronics	1N5822	2	$0.44 \cdot 2 = 0.88$
Arduino (used for microcontroller)	Arduino	A000067	1	40.30
Diode	Micro Commercial Co	MBR0520-TP	2	$0.35 \cdot 2 = 0.70$
Display	Adafruit	1447	1	10.95
Microphone	Adafruit	1063	1	6.95
Tuning Bit	McMaster-Carr	5543A23	1	5.40
10k Ω Resistor	Vishay	CRMA1206AF10K0FKEF	12	$0.35 \cdot 12 = 4.20$
32k Ω Resistor	KOA Speer	RN732ATTD3202D25	2	$0.38 \cdot 2 = 0.76$
220 Ω Resistor	Panasonic	ERA-6VEB2200V	2	$0.70 \cdot 2 = 1.40$
1 uF Capacitor	Taiyo Yuden	TMR107B7105KA-T	6	$0.18 \cdot 6 = 1.08$
680 uF Capacitor	EPCOS / TDK	B41888C4687M000	2	$0.97 \cdot 2 = 1.94$
47 uF Capacitor	Nichicon	UCL1E470MCL1GS	1	0.61
Continued on next page				

[1] Using 2018-2019 numbers.
[2] Using undergraduate fall 2021 numbers.

Table 1 – continued from previous page

Description	Manufacturer	Part #	Quantity	Cost ($)
224 uF Capacitor	Panasonic	EEE-FK0J221V	1	1.02
33 uH Inductor	Taiyo Yuden	CBC3225T330KR	2	$0.30 \cdot 2 = 0.60$
Motor	ISL Products International	MOT-IG32PGM1:264E	1	50.00
Battery Pack	Battery Space	CH-UN180C37	1	92.00
Motor Controller	HiLetgo	BTS7960	1	10.99
Quoted Machine Shop Labor Hours			20	$36.65 \cdot 20 = 733$
Total				962.78

3.1.3 Sum

Combining the total part costs with the total labor costs we get a final cost analysis of $\$951.79 + \$29,139.75 = \$30,102.53$.

3.2 Schedule

Table 2: Schedule

Week	Joseph Babbo	Colin Wallace	Riley Woodson
1/17	Jointly brainstorm project ideas	Jointly brainstorm project ideas	Jointly brainstorm project ideas
1/24	Jointly determine project, laboratory safety training, CAD assignment	Jointly determine project, laboratory safety training, CAD assignment	Jointly determine project, laboratory safety training, CAD assignment
1/31	Jointly write RFA, answer RFA questions, get project approved	Jointly write RFA, answer RFA questions, get project approved	Jointly write RFA, answer RFA questions, get project approved
2/7	Project proposal ethics and safety, project proposal subsystem overview, project proposal LaTeX formatting	Project proposal block diagram, project proposal subsystem overview	Project proposal visual aid, project proposal tolerance analysis, initial conversation with machine shop
2/14	Jointly research parts	Jointly research parts	Jointly research parts
2/21	Block diagram paragraph, requirements and verification, cost analysis, schedule, expand ethics and safety, LaTeX formatting	Revisit block diagram, circuit schematic, requirements and verification	Revisit machine shop, revisit tolerance analysis, physical design
2/28	Ordering/sourcing components	Design layout of PCB and estimate size of the box	Finalize machine shop design
3/7	Order the first PCB	Design box to hold the electrical components	Fit motor to the machine design
3/21	Test electrical components, specifically power regulation for performance	Solder components to the first PCB design	Test the motor for correct torque vs current settings
Continued on next page			

Table 2 – continued from previous page

Week	Joseph Babbo	Colin Wallace	Riley Woodson
3/28	Test the individual components on the PCB to verify functionality	Program and verify that the microcontroller is capable of reading tones and writing to the display	Test functionality of the entire PCB and system
4/4	Create revision of the PCB based on initial testing	Further develop microprocessor code for frequency detection and motor control	Test entire first system on piano pins
4/11	Test audio + control subsystems requirements	Test battery + power subsystems requirements	Test motor + display + buttons subsystem requirements
4/18	Jointly prepare mock demo	Jointly prepare mock demo	Jointly prepare mock demo
4/25	Jointly prepare demo and mock presentation	Jointly prepare demo and mock presentation	Jointly prepare demo and mock presentation
5/24	Jointly prepare presentation and touch-up on final paper	Jointly prepare presentation and touch-up on final paper	Jointly prepare presentation and touch-up on final paper

4 Conclusion

4.1 Accomplishments and Uncertainties

The major accomplishment of this project was the complete functionality of the requirements and the high level goals. The Automatic Piano Tuner was able to detect the full range of frequencies that span a normal piano, resolve between all of them, determine adjustments needed for tuning, and provide enough torque to turn the pin. This was also done with the added benefit of staying within the cost of a single professional tuning session, making the device a viable alternative to professional tuning.

Every system worked as anticipated, leaving no uncertainties. There could be further work to optimize and improve systems, however this was outside of the scope of our requirements.

4.2 Future Work

This device would benefit from a couple of addons before its real market release. A new casing would adapt the device to be more similar to common day power tools, which would allow for instant familiarity for consumers. The ideal design would have a compact casing around the motor, which would also hold the circuit boards, connected to a handle with a rechargeable battery mounted on the bottom. The device also needs an upgraded microphone before market release. The current microphone was able to demonstrate the feasibility of the product, but was chosen for its cheap price. A higher quality microphone would most likely deal with the low and high frequencies much better. The software control would also have the auto correlation function instead of the adapted period estimation that is currently in use. A micro controller with higher RAM capacity would be required for this change.

4.3 Ethical Considerations

In the project, there was firm adhesion to both the IEEE and ACM Code of Ethics. While developing the Automatic Piano Tuner, there was care to cite and credit all work that used. Additionally advice from TAs and professors was sought and used throughout the project per IEEE Code of Ethics I.3. [10]

Throughout the entire development of the Automatic Piano Tuner, anyone who helped contribute to the project was respected. Due to requiring outside people to successfully accomplish the project, it was made sure throughout the process to treat everyone fairly as per IEEE Code of Ethics II [10].

Due to the usage of a battery in the project design, considerations for safety and regulatory standards regarding preventing fires and explosive injuries from lithium powered devices were followed. This included keeping the battery at temperatures below 130° F and above 32° F, and making sure the battery was not not dropped, crushed, or punctured [11].

References

[1] J. Ross, "How Much Does It Cost To Tune A Piano," Joshua Ross Piano. [Online]. Available: https://joshuarosspiano.com/how-much-does-it-cost-to-tune-a-piano/

[2] "Keyboard and Frequencies," Sengpiel Audio. [Online]. Available: http://www.sengpielaudio.com/calculator-notenames.htm

[3] J. Harold A. Conklin, "Tuning pin for pianos," U.S. Patent US4 920 847A, 1989. [Online]. Available: https://patents.google.com/patent/US4920847A/en

[4] "How Many Keys On A Piano?" Broughton Pianos. [Online]. Available: https://www.broughtonpianos.co.uk/blog/how-many-keys-on-a-piano

[5] "Care and Maintenance of a Piano." [Online]. Available: https://www.yamaha.com/en/musical_instrument_guide/piano/maintenance/maintenance002.html

[6] "Mot-ig32pgm1:264e," ISL Products International. [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/ISL%20PDFs/MOT-IG32GM-12V-264E.pdf

[7] "Bts7960 high current 43a h-bridge motor driver," Handson Technology. [Online]. Available: https://www.handsontec.com/dataspecs/module/BTS7960%20Motor%20Driver.pdf

[8] "Salary Averages." [Online]. Available: https://ece.illinois.edu/admissions/why-ece/salary-averages

[9] "Rankings and Statistics." [Online]. Available: https://ece.illinois.edu/admissions/why-ece/rankings-and-statistics

[10] "IEEE Code of Ethics," IEEE. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

[11] "Preventing Fire and/or Explosion Injury from Small and Wearable Lithium Battery Powered Devices," OSHA. [Online]. Available: https://www.osha.gov/sites/default/files/publications/shib011819.pdf

Appendix A Requirement, Verification, and Result Tables

Table 3: Audio Acquisition Subsystem Requirements, Verification, and Results

Requirements	Verification	Results
1. Pick up frequencies accurately down to 27.5Hz.	1 With an external device play a perfect 27.5Hz tone. 2 Using the audio acquisition subsystem alongside the software frequency analyzer on the microcontroller, determine the frequency (as given by the microphone). 3 Make sure the tone is within ±10% of 27.5Hz to fulfill the accuracy requirement.	Tested at 28.5Hz using sub-woofer (sub-woofer could not accurately output frequencies less than 28.5Hz). Value as given by the microphone was 28.4Hz, within ±10% of both 27.5Hz and 28.5Hz, fulfilling the requirement.
2. Pick up frequencies accurately up to 4186Hz.	1 With an external device play a perfect 4186Hz tone. 2 Using the audio acquisition subsystem alongside the software frequency analyzer on the microcontroller, determine the frequency (as given by the microphone). 3 Make sure the tone is within ±10% of 4186Hz to fulfill the accuracy requirement.	Tested at 4186Hz using phone speaker. Values as given by the microphone were 4273.6Hz and 3846.2Hz, both within ±10% of 4186Hz, fulfilling the requirement.

Continued on next page

Table 3 – continued from previous page

Requirements	Verification	Results
3. Differentiate between frequencies 1.6Hz apart.	1 With an external device play a tone starting at 27.5Hz. 2 Sweep up slowly until the software frequency analyzer on the microcontroller notices a frequency change. 3 Make sure the software frequency analyzer on the microcontroller is able to detect a separate frequency at 29.1 Hz or earlier.	Starting at 28.5Hz (due to subwoofer limitations) microphone was able to accurately detect a new frequency at 29Hz, a frequency smaller than 1.6Hz apart fulfilling the requirement.

Table 4: Control System Subsystem Requirements, Verification, and Results

Requirements	Verification	Results
1. Accurately determine heard frequency in less than 5 seconds.	1 With an external device play a tone (use a tone between 27.5Hz and 4186Hz to insure the 'Audio Acquisition Subsystem' can pick up the frequency). 2 Measure the time it takes for the microcontroller to analyze and determine the tone's frequency. 3 Make sure the time from tone played to frequency determined is less than 5 seconds.	Accurately determines frequency 690Hz (displaying 686.8Hz) in 46ms. This is less than 5 seconds, fulfilling the requirement.
2. Store all 88 standard piano frequencies.	1 Attempt to store table of the 88 piano keys alongside their known frequencies in the microcontroller's memory. 2 Make sure the entire table fits inside memory for later retrieval.	All 88 standard piano frequencies are stored in the ATmega2560's memory fulfilling the requirement.

Table 5: Battery Subsystem Requirements, Verification, and Results

Requirements	Verification	Results
1. Remain powered for 2 hours or greater at load.	1 Turn on device and use at high load and measure time until the battery runs out. 2 Make sure the time until shutdown is greater than 2 hours.	Device used at high load for 2 hours. After 2 hours only 1/3 of the battery was drained fulfilling the requirement.

Table 6: Power Regulation Subsystem Requirements, Verification, and Results

Requirements	Verification	Results
1. Provide 5V to the 'Audio Acquisition', 'Control System', 'Button', and 'Display' subsystems.	1 Plug in battery to power the 'Power Regulation' subsystem. 2 Measure output of the 5V lead. 3 Make sure voltage levels are within ±5% of 5V.	5.19V measured from the voltage converter IC on the PCB. This is within ±5% of 5V, fulfilling the requirement.
2. Provide 12V to the 'Motor Assembly' subsystem.	1 Plug in battery to power the 'Power Regulation' subsystem. 2 Measure output of the 12V lead. 3 Make sure voltage levels are within ±5% of 12V.	11.945V measured from the battery to the motor controller. This is within ±5% of 12V, fulfilling the requirement.

Table 7: Electric Motor Drive Subsystem Requirements, Verification, and Results

Requirements	Verification	Results
1. Able to rotate piano tuning pin.	1 Put piano tuner bit on tuning pin. 2 Power motor manually using leads connected to controller. 3 Make sure that the tuning pin rotates.	Rotated a piano pin successfully fulfilling the requirement.

Continued on next page

Table 7 – continued from previous page

Requirements	Verification	Results
2. Able to produce the required 9-14 Nm .	1 Attach one meter lever arm to motor shaft. 2 Place other end of arm onto weight scale. 3 Power motor, with arm rotating into weight scale, and record value. 4 Use equation $1m \cdot 9.81m/s^2 \cdot Akg$ (where A is the weight-scale measurement) to calculate torque in Nm. 5 Make sure torque is within 9-14 Nm range.	Attached 3.93 in lever arm to the motor shaft, and rotating it onto a scale measured 22.1 lbs, finding a calculated torque value of 9.83Nm.

Table 8: Display Subsystem Requirements, Verification, and Results

Requirements	Verification	Results
1. Display current note and octave at all times.	1 While Automatic Piano Tuner is in use, record the display. 2 Make sure the current note and octave are always displayed on screen for the entire time.	The current note and octave are always displayed on the device, fulfilling the requirement.
2. Update display within 1 second of a button push.	1 Upon pressing a button track time until the display updates. 2 Make sure the display update time is less than 1 second.	Display is updated on average within 19.9ms of a button press. This is less than 1 second fulfilling the requirement.

Table 9: Buttons Subsystem Requirements, Verification, and Results

Requirements	Verification	Results
1. Pressing the down button should update the frequency listening against in less than 1 second.	1 Press the down button and measure time. 2 Check when the software on the microcontroller changes the frequency it is checking against. 3 Make sure the time between button press and software change is less than 1 second.	Frequency listening against is updated on average in 31 ms of a down button press. This is less than 1 second fulfilling the requirement.
2. Pressing the up button should update the frequency listening against in less than 1 second.	1 Press the up button and measure time. 2 Check when the software on the microcontroller changes the frequency it is checking against. 3 Make sure the time between button press and software change is less than 1 second.	Frequency listening against is updated on average in 31.1 ms of an up button press. This is less than 1 second fulfilling the requirement.
3. Pressing both buttons at the same time (within 10ms of each other) should do nothing.	1 Press both the up and down button simultaneously (within 10 ms of each other). 2 Check the software on the microcontroller for any update in the frequency it is checking against. 3 Make sure there is no change in the frequency the software on the microcontroller is checking against.	Frequency listening against is not updated when both the up and down buttons are pressed within 10 ms of each other, fulfilling the requirement.

Appendix B Physical Design

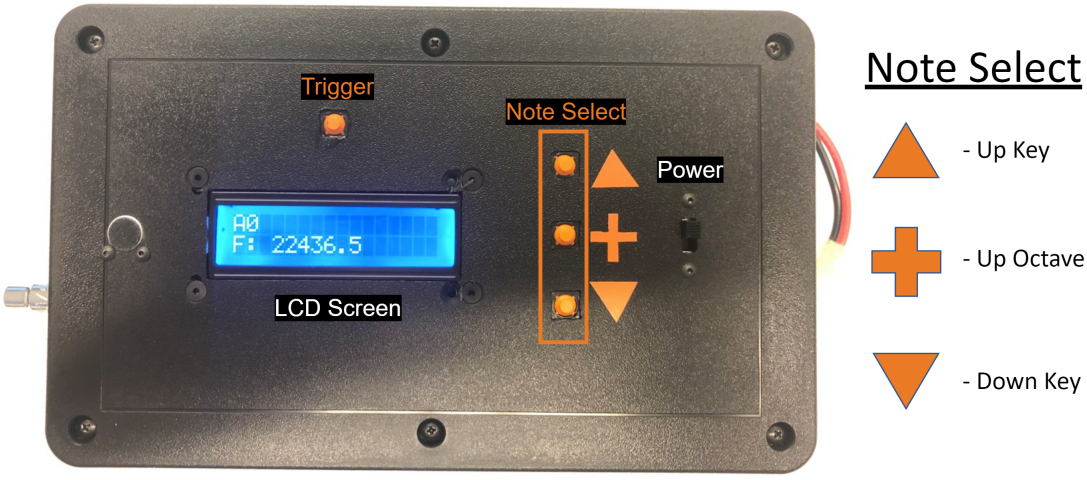

Figure 10: 'Final physical design of the user interface'.

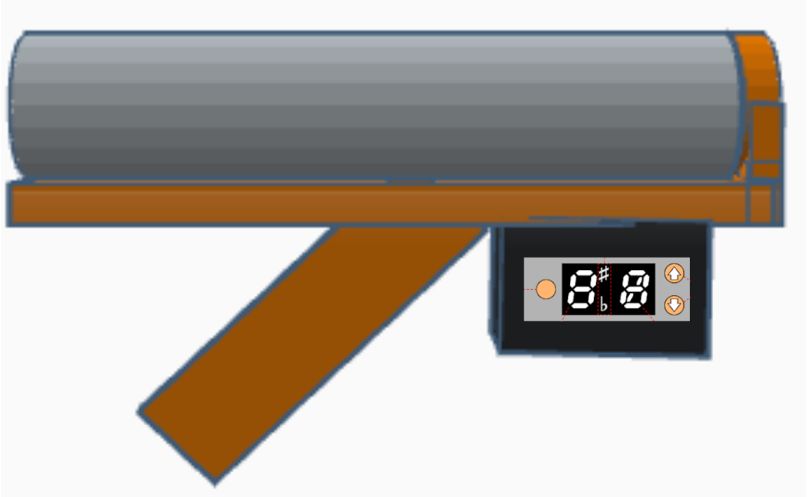

Figure 11: 'Original CAD mock-up of the side view of the Automatic Piano Tuner.

Appendix C Circuit Schematic

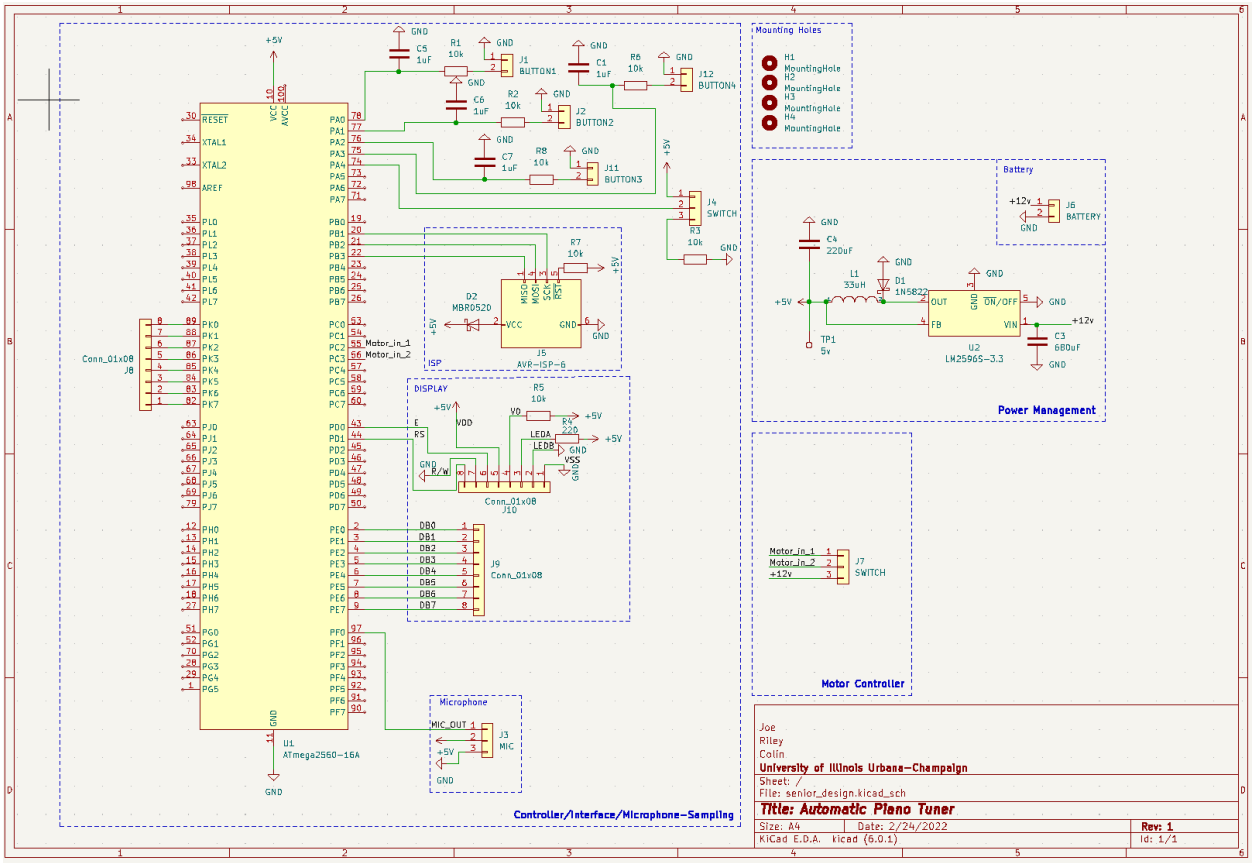

Figure 12: Automatic Piano Tuner circuit schematic.

Appendix D PCB Schematic

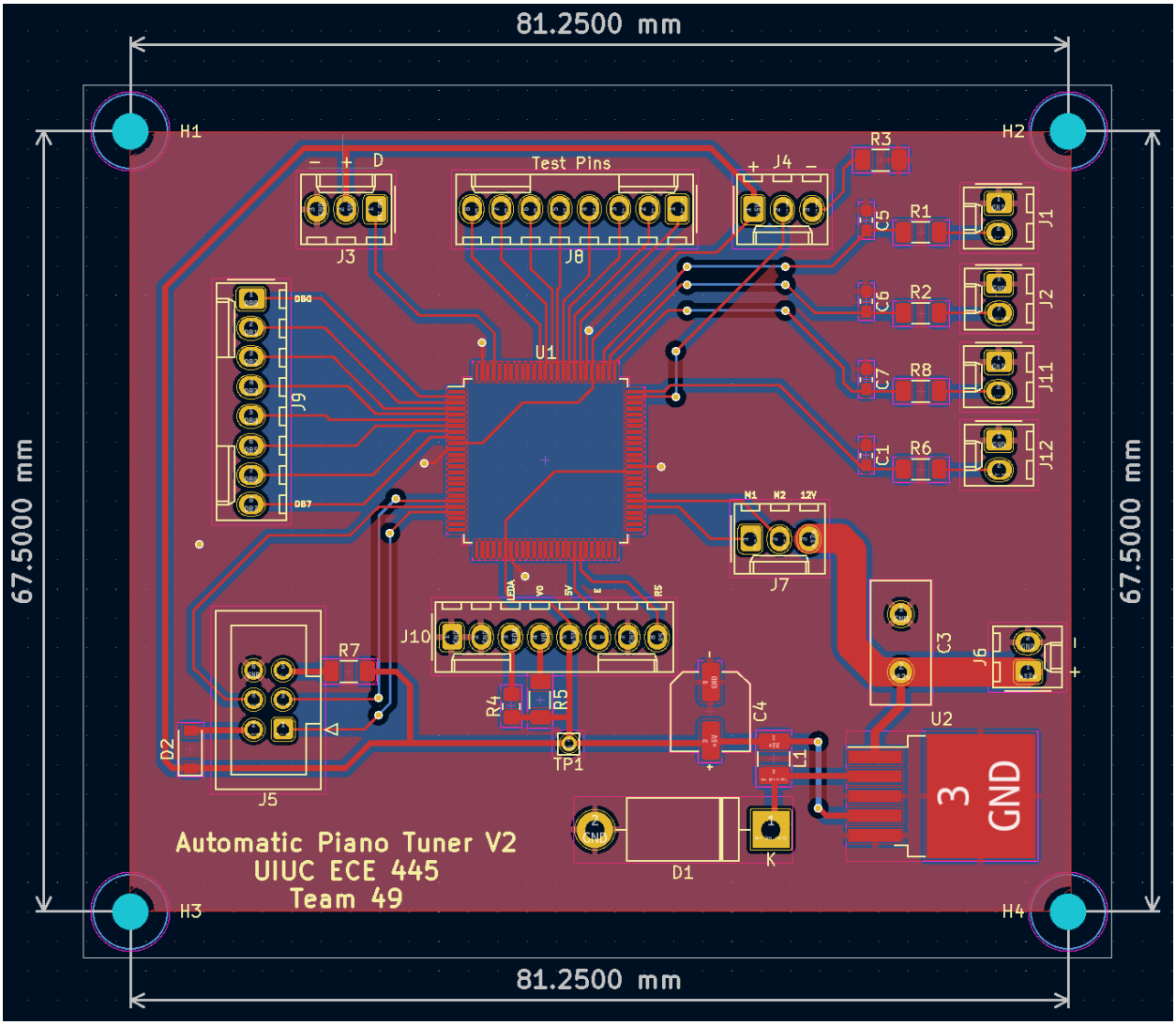

Figure 13: PCB layout for the schematic.

Appendix E Main Control Loop

Main arduino code loop placed on the ATmega2560 to fulfill the Automatic Piano Tuner's requirements.

```
#include "piano_tuning_lib.h"
#include <string.h>
#include <LiquidCrystal.h>

extern "C"{
  #include "ADC.h"
};
#define ADC_LEN 512

using namespace std;

int ADC_input = 5;

bool up_button_pressed;
bool down_button_pressed;
bool octave_button_pressed;

int up_down_timer;
bool up_down_check;
bool up_down_good;

int motor_direction;

int timer_goal;

unsigned long time_print;

//data storage variables
byte newData = 0;
byte prevData = 0;

//freq variables
unsigned int timer = 0;//counts period of wave
unsigned int period = -1;
float prev_freq = 0;
int t = 0;
int t_up_down;
```

```
LiquidCrystal lcd(35, 34, 4, 5, 6, 7); //working

uint8_t PIN_octave_button = 67;
uint8_t PIN_safety_button = 66;
uint8_t PIN_up_button = 65;
uint8_t PIN_down_button = 64;
uint8_t PIN_power_switch = 0;  // ATMEGA2560 only has a few interrupt pins

uint8_t PIN_switch = 63;

float adc[ADC_LEN];
float actual_freq;

int print_num = 0;

int RPWM_Output = 47;
int LPWM_Output = 46;



void setup(){
    cli();//diable interrupts

    pinMode(PIN_safety_button, INPUT_PULLUP);
    pinMode(PIN_up_button, INPUT_PULLUP);
    pinMode(PIN_down_button, INPUT_PULLUP);
    pinMode(PIN_octave_button, INPUT_PULLUP);
    pinMode(PIN_power_switch, INPUT);
    pinMode(82, OUTPUT); //3
    pinMode(83, OUTPUT); //2
    pinMode(84, OUTPUT); //1
    digitalWrite(82, 1);
    digitalWrite(83, 1);
    digitalWrite(84, 1);

    pinMode(RPWM_Output, OUTPUT);
    pinMode(LPWM_Output, OUTPUT);


    up_button_pressed = false;
    down_button_pressed = false;
    octave_button_pressed = false;
```

```
    up_down_timer = 0;
    up_down_check = false;
    up_down_good = false;

    motor_direction = 0;

    timer_goal = 100;
    lcd.begin(16, 2);
    lcd.clear();
    lcd.print(returnKey());
    //adc_init(char(ADC_input));


    //set up continuous sampling of analog pin 0

    //clear ADCSRA and ADCSRB registers
    ADCSRA = 0;
    ADCSRB = 0;

    ADMUX |= (1 << REFS0); //set reference voltage
    ADMUX |= (1 << ADLAR); //left align the ADC value- so we can read highest 8 bits from ADCH register

    ADCSRA |= (1 << ADPS2) | (0 << ADPS1)| (0 << ADPS0); //set ADC clock with 32 prescaler- 8mHz/32=250
    ADCSRA |= (1 << ADATE); //enabble auto trigger
    ADCSRA |= (1 << ADIE); //enable interrupts when measurement complete
    ADCSRA |= (1 << ADEN); //enable ADC
    ADCSRA |= (1 << ADSC); //start ADC measurements

    sei();//enable interrupts
    return;
}

// Interrupt handler for ADC
ISR(ADC_vect) {//when new ADC value ready
  prevData = newData;//store previous value
  newData = ADCH;//get value from A0
  if (prevData < 127 && newData >=127){//if increasing and crossing midpoint
    period = timer;//get period
    timer = 0;//reset timer
  }
  timer++;//increment timer at rate of 38.5kHz
}
```

```
void loop(){
            lcd.setCursor(0,0);

            //check power switch
            if(digitalRead(PIN_power_switch) == LOW){
                digitalWrite(84, 0);
                digitalWrite(83, 0);
                lcd.noDisplay();
                sleepFunc();
                lcd.display();
                digitalWrite(84, 1);
                digitalWrite(83, 1);
            }

            //check buttons
            if(digitalRead(PIN_up_button) == LOW && digitalRead(PIN_down_button) == LOW)
                ;
            else if(digitalRead(PIN_up_button) == LOW && !up_button_pressed){
                if (!up_down_check){
                    //Display time to update listening frequency time
//                    time_print = millis();
                    ;
                }
                else{
                    print_num = nextKey();
                    up_button_pressed = true;
                    up_down_check = false;
                    lcd.setCursor(0,0);
                    lcd.print("                ");
                    lcd.setCursor(0,0);
                    lcd.print(returnKey());

                    //Display time to update listening frequency time
//                    lcd.setCursor(0, 1);
//                    lcd.print("                ");
//                    lcd.setCursor(0, 1);
//                    lcd.print("F: ");
//                    lcd.print(getFreq(print_num));
//                    lcd.print(" ");
//                    lcd.print("T: ");
//                    time_print = millis() - time_print;
//                    lcd.print(time_print);
//                    lcd.print("ms");
```

```
            }
        }
        else if(digitalRead(PIN_down_button) == LOW && !down_button_pressed){
            if (!up_down_check){
                //Display time to update listening frequency time
//                  time_print = millis();
                ;
            }
            else{
                print_num = prevKey();
                down_button_pressed = true;
                up_down_check = false;
                lcd.setCursor(0,0);
                lcd.print("                ");
                lcd.setCursor(0,0);
                lcd.print(returnKey());

                //Display time to update listening frequency time
//                  lcd.setCursor(0, 1);
//                  lcd.print("                ");
//                  lcd.setCursor(0, 1);
//                  lcd.print("F: ");
//                  lcd.print(getFreq(print_num));
//                  lcd.print(" ");
//                  lcd.print("T: ");
//                  time_print = millis() - time_print;
//                  lcd.print(time_print);
//                  lcd.print("ms");
            }
        }

        if(digitalRead(PIN_octave_button) == LOW && !octave_button_pressed){
            print_num = nextOctave();
            octave_button_pressed = true;
            lcd.setCursor(0,0);
            lcd.print("                ");
            lcd.setCursor(0,0);
            lcd.print(returnKey());

            //Display time to update listening frequency time
//              time_print = millis();
//              lcd.setCursor(0, 1);
```

```
//                lcd.print("                ");
//                lcd.setCursor(0, 1);
//                    lcd.print("F: ");
//                    lcd.print(getFreq(print_num));
//                    lcd.print(" ");
//                lcd.print("T: ");
//                time_print = millis() - time_print;
//                lcd.print(time_print);
//                lcd.print("ms");

            }

            //button guard
            if (digitalRead(PIN_up_button) == HIGH && up_button_pressed)
                up_button_pressed = false;
            if (digitalRead(PIN_down_button) == HIGH && down_button_pressed)
                down_button_pressed = false;
            if (digitalRead(PIN_octave_button) == HIGH && octave_button_pressed)
                octave_button_pressed = false;

            //up-down check
            if (!up_down_check){
                if ((digitalRead(PIN_up_button) == LOW && digitalRead(PIN_down_button) == LOW) || (digi
                    up_down_check = false;
                    up_down_timer = 0;
                }
                else if (up_down_timer < timer_goal)
                    up_down_timer++;
                else{
                    up_down_check = true;
                    up_down_timer = 0;
                }
            }


            //turn motor correctly
            //read_adc_seq(adc, ADC_LEN);
            actual_freq = 38462/(float)period;//timer rate/period
            //Display Target frequency for RVs
            t++;
            if(prev_freq != actual_freq && (t>=1000)){
              t = 0;
              prev_freq = actual_freq;
```

```
              lcd.setCursor(0,1);
              lcd.print("                ");
              lcd.setCursor(0,1);
              lcd.print("F: ");
              lcd.print(actual_freq, 1);
            }

            if (digitalRead(PIN_safety_button) == LOW){
                //clear second line for RVs
//                lcd.setCursor(0,1);
//                lcd.print("                ");
//                lcd.setCursor(0,1);

                //Display time to find frequency for RVs
//                time_print = millis();
//                time_print = millis() - time_print;
//                lcd.setCursor(0,1);
//                lcd.print(time_print);
//                lcd.print(" ");

                //Display Target frequency for RVs
//                lcd.print("T: ");
//                lcd.print(getFreq(print_num));

                //Display Actual frequency for RVs
//                lcd.print(" A: ");
//                lcd.print(actual_freq);
                motor_direction = motorDirection(actual_freq, print_num);
                //output motor direction to motor controller (1 clockwise, 0 do nothing, -1 counter-cl
            }
            else{
                motor_direction = 0;
            }


            if (motor_direction == 0){
              digitalWrite(LPWM_Output, 0);
              digitalWrite(RPWM_Output, 0);
            }
            else if (motor_direction == 1){
              digitalWrite(LPWM_Output, 1);
              digitalWrite(RPWM_Output, 0);
            }
```

```
            else if (motor_direction == -1){
              digitalWrite(LPWM_Output, 0);
              digitalWrite(RPWM_Output, 1);
            }
    return;
}
```

Appendix F Piano Tuning Library

C++ code place on the ATmega2560 to assist the main control loop.

```
#include "piano_tuning_lib.h"
#include <math.h>
#include <avr/pgmspace.h>
#include "newfreqdetect.h"
#include <Arduino.h>
#include <avr/sleep.h>

const char A0N[] PROGMEM = "A0";
const char AS0N[] PROGMEM = "A#0";
const char B0N[] PROGMEM = "B0";
const char C1N[] PROGMEM = "C1";
const char CS1N[] PROGMEM = "C#1";
const char D1N[] PROGMEM = "D1";
const char DS1N[] PROGMEM = "D#1";
const char E1N[] PROGMEM = "E1";
const char F1N[] PROGMEM = "F1";
const char FS1N[] PROGMEM = "F#1";
const char G1N[] PROGMEM = "G1";
const char GS1N[] PROGMEM = "G#1";
const char A1N[] PROGMEM = "A1";
const char AS1N[] PROGMEM = "A#1";
const char B1N[] PROGMEM = "B1";
const char C2N[] PROGMEM = "C2";
const char CS2N[] PROGMEM = "C#2";
const char D2N[] PROGMEM = "D2";
const char DS2N[] PROGMEM = "D#2";
const char E2N[] PROGMEM = "E2";
const char F2N[] PROGMEM = "F2";
const char FS2N[] PROGMEM = "F#2";
const char G2N[] PROGMEM = "G2";
const char GS2N[] PROGMEM = "G#2";
```

```
const char A2N[] PROGMEM = "A2";
const char AS2N[] PROGMEM = "A#2";
const char B2N[] PROGMEM = "B2";
const char C3N[] PROGMEM = "C3";
const char CS3N[] PROGMEM = "C#3";
const char D3N[] PROGMEM = "D3";
const char DS3N[] PROGMEM = "D#3";
const char E3N[] PROGMEM = "E3";
const char F3N[] PROGMEM = "F3";
const char FS3N[] PROGMEM = "F#3";
const char G3N[] PROGMEM = "G3";
const char GS3N[] PROGMEM = "G#3";
const char A3N[] PROGMEM = "A3";
const char AS3N[] PROGMEM = "A#3";
const char B3N[] PROGMEM = "B3";
const char C4N[] PROGMEM = "C4";
const char CS4N[] PROGMEM = "C#4";
const char D4N[] PROGMEM = "D4";
const char DS4N[] PROGMEM = "D#4";
const char E4N[] PROGMEM = "E4";
const char F4N[] PROGMEM = "F4";
const char FS4N[] PROGMEM = "F#4";
const char G4N[] PROGMEM = "G4";
const char GS4N[] PROGMEM = "G#4";
const char A4N[] PROGMEM = "A4";
const char AS4N[] PROGMEM = "A#4";
const char B4N[] PROGMEM = "B4";
const char C5N[] PROGMEM = "C5";
const char CS5N[] PROGMEM = "C#5";
const char D5N[] PROGMEM = "D5";
const char DS5N[] PROGMEM = "D#5";
const char E5N[] PROGMEM = "E5";
const char F5N[] PROGMEM = "F5";
const char FS5N[] PROGMEM = "F#5";
const char G5N[] PROGMEM = "G5";
const char GS5N[] PROGMEM = "G#5";
const char A5N[] PROGMEM = "A5";
const char AS5N[] PROGMEM = "A#5";
const char B5N[] PROGMEM = "B5";
const char C6N[] PROGMEM = "C6";
const char CS6N[] PROGMEM = "C#6";
const char D6N[] PROGMEM = "D6";
const char DS6N[] PROGMEM = "D#6";
```

```
const char E6N[] PROGMEM = "E6";
const char F6N[] PROGMEM = "F6";
const char FS6N[] PROGMEM = "F#6";
const char G6N[] PROGMEM = "G6";
const char GS6N[] PROGMEM = "G#6";
const char A6N[] PROGMEM = "A6";
const char AS6N[] PROGMEM = "A#6";
const char B6N[] PROGMEM = "B6";
const char C7N[] PROGMEM = "C7";
const char CS7N[] PROGMEM = "C#7";
const char D7N[] PROGMEM = "D7";
const char DS7N[] PROGMEM = "D#7";
const char E7N[] PROGMEM = "E7";
const char F7N[] PROGMEM = "F7";
const char FS7N[] PROGMEM = "F#7";
const char G7N[] PROGMEM = "G7";
const char GS7N[] PROGMEM = "G#7";
const char A7N[] PROGMEM = "A7";
const char AS7N[] PROGMEM = "A#7";
const char B7N[] PROGMEM = "B7";
const char C8N[] PROGMEM = "C8";

const char *const noteTable[] PROGMEM = {
    A0N,
    AS0N,
    B0N,
    C1N,
    CS1N,
    D1N,
    DS1N,
    E1N,
    F1N,
    FS1N,
    G1N,
    GS1N,
    A1N,
    AS1N,
    B1N,
    C2N,
    CS2N,
    D2N,
    DS2N,
    E2N,
```

F2N,
FS2N,
G2N,
GS2N,
A2N,
AS2N,
B2N,
C3N,
CS3N,
D3N,
DS3N,
E3N,
F3N,
FS3N,
G3N,
GS3N,
A3N,
AS3N,
B3N,
C4N,
CS4N,
D4N,
DS4N,
E4N,
F4N,
FS4N,
G4N,
GS4N,
A4N,
AS4N,
B4N,
C5N,
CS5N,
D5N,
DS5N,
E5N,
F5N,
FS5N,
G5N,
GS5N,
A5N,
AS5N,
B5N,

```
    C6N,
    CS6N,
    D6N,
    DS6N,
    E6N,
    F6N,
    FS6N,
    G6N,
    GS6N,
    A6N,
    AS6N,
    B6N,
    C7N,
    CS7N,
    D7N,
    DS7N,
    E7N,
    F7N,
    FS7N,
    G7N,
    GS7N,
    A7N,
    AS7N,
    B7N,
    C8N
    };
const float freqTable[] PROGMEM = {
    27.50,
    29.14,
    30.87,
    32.70,
    34.65,
    36.71,
    38.89,
    41.20,
    43.65,
    46.25,
    49.00,
    51.91,
    55.00,
    58.27,
    61.74,
    65.41,
```

```
69.30,
73.42,
77.78,
82.41,
87.31,
92.50,
98.00,
103.83,
110.00,
116.54,
123.47,
130.81,
138.59,
146.83,
155.56,
164.81,
174.61,
185.00,
196.00,
207.65,
220.00,
233.08,
246.94,
261.63,
277.18,
293.66,
311.13,
329.63,
349.23,
369.99,
392.00,
415.30,
440.00,
466.16,
493.88,
523.25,
554.37,
587.33,
622.25,
659.25,
698.46,
739.99,
783.99,
```

```
    830.61,
    880.00,
    932.33 ,
    987.77 ,
    1046.50,
    1108.73,
    1174.66,
    1244.51,
    1318.51,
    1396.91,
    1479.98,
    1567.98,
    1661.22,
    1760.00,
    1864.66,
    1975.53,
    2093.00,
    2217.46,
    2349.32,
    2489.02,
    2637.02,
    2793.83,
    2959.96,
    3135.96,
    3322.44,
    3520.00,
    3729.31,
    3951.07,
    4186.01
    };

const char *const octave_minus_7[] PROGMEM = {A7N, AS7N, B7N, C8N};
const char *const octave_minus_6[] PROGMEM = {CS7N, D7N, DS7N, E7N, F7N, FS7N, G7N, GS7N};

const int minus_7_length = 4;
const int minus_6_length = 8;

/**
 * Quickly gets text name of the current_note
 *
 * @return The text name of the current_note
 */
const char* returnKey(){
```

```
    return note_text;
}

/**
 * Quickly gets the next key based on key passed in
 *
 * Returns the first key if the current key is the final key.
 *
 * @return Modifies global variables 'int current_note' and 'String note_text';
 */
int nextKey(){
    current_note += 1;
    if (current_note >= 88)
        current_note = 0;

    strcpy_P(note_text, (char *)pgm_read_word(&noteTable[current_note]));
    return current_note;
}

/**
 * Quickly gets the previous key based on the key passed in
 *
 * Returns the last key if the current key is the first key.
 *
 * @return Modifies global variables 'int current_note' and 'String note_text';
 */
int prevKey(){
    current_note -= 1;
    if (current_note <= -1)
        current_note = 87;

    strcpy_P(note_text, (char *)pgm_read_word(&noteTable[current_note]));
    return current_note;
}

/**
 * Quickly gets the key directly up an octave of the key passed in
 *
 * If on the final octave for the note, returns the first octave for the note
 *
 * @return Modifies global variables 'int current_note' and 'String note_text';
 */
int nextOctave(){
```

```
for (int i=0; i<minus_7_length; i++){
        strcpy_P(note_test, (char *)pgm_read_word(&octave_minus_7[i]));
        if (String(note_text) == String(note_test)){
            current_note -= 84; //12 * 7
            strcpy_P(note_text, (char *)pgm_read_word(&noteTable[current_note]));
            return current_note;
        }
    }

    for (int i=0; i<minus_6_length; i++){
        strcpy_P(note_test, (char *)pgm_read_word(&octave_minus_6[i]));
        if (String(note_text) == String(note_test)){
            current_note -= 72; //12 * 6
            strcpy_P(note_text, (char *)pgm_read_word(&noteTable[current_note]));
            return current_note;
        }
    }

    current_note += 12;
    strcpy_P(note_text, (char *)pgm_read_word(&noteTable[current_note]));
    return current_note;
}

// /**
//  * Get float table from microphone to input to actualFreq
//  *
//  * @return pointer to float table containing freq information
//  */
// float *getAudio(){
//     float floatTable[1] = {0.0};
//     return floatTable;
// }

/**
 * Quickly gets the frequency of key passed in
 *
 * @param current_note Integer of the current note
 * @return float of the frequency of the current note
 */
float getFreq(int current_note){
    float current_freq = pgm_read_float(&freqTable[current_note]);
    return current_freq;
}
```

```
/**
 * Sees if the given frequency is within an acceptable range of the given note
 *
 * @param current_note Integer of the current note
 * @param actualFreq Frequency to be tested against
 * @return bool: 1 if within range, 0 if outside range
 */
bool withinRange(float actualFreq, int current_note){
    //notes are different by 1.059463...
    float targetFreq = getFreq(current_note);
    if (actualFreq > targetFreq + targetFreq*.01)
        return false;
    if (actualFreq < targetFreq - targetFreq*.01)
        return false;
    return true;
}

/**
 * Sees if the given frequency is within an acceptable range of the given note
 *
 * @param actualFreq Frequency to be tested against
 * @param current_note Integer of the current note
 * @return int: 1 = turn clockwise, -1 = turn counter clockwise, 0 = in acceptable range - do nothing
 */
int motorDirection(float actualFreq, int current_note){
    if (withinRange(actualFreq, current_note) == 1)
        return 0;
    float targetFreq = getFreq(current_note);
    if (signbit(targetFreq - actualFreq))
        return -1;
    return 1;
}

/**
 * TODO: Get the current frequency from the audio input device
 * @param freqBuffer pointer to float array of mic input
 * @return float of the frequency
 */
float actualFreq(float *freqBuffer){
    return detectFrequencyNew(freqBuffer);
}
```

```
/**
 * Master test function, no params or return value
 */
// void testFunc(void){
//     string curNote;
//     curNote = "D2";
//     cout << curNote << '\n' << '\n';

//     // test nextNote
//     string nextNote = nextKey(curNote);
//     cout << nextNote << '\n';

//     string lastNote;
//     lastNote = "C8";
//     cout << lastNote << '\n';
//     string last2first = nextKey(lastNote);
//     cout << last2first << '\n' << '\n';

//     // test prevNote
//     string prevNote = prevKey(curNote);
//     cout << prevNote << '\n';

//     string firstNote;
//     firstNote = "A0";
//     cout << firstNote << '\n';
//     string first2last = prevKey(firstNote);
//     cout << first2last << '\n' << '\n';

//     //test nextOctave
//     string A7 = "A7";
//     string Asharp7 = "A#7";
//     string F7 = "F7";
//     string Fsharp7 = "F#7";
//     string nextOctC8 = nextOctave(lastNote);
//     string nextOctA7 = nextOctave(A7);
//     string nextOctAsharp7 = nextOctave(Asharp7);
//     string nextOctD2 = nextOctave(curNote);
//     string nextOctF7 = nextOctave(F7);
//     string nextOctFsharp7 = nextOctave(Fsharp7);

//     cout << nextOctC8 << '\n';
//     cout << nextOctA7 << '\n';
//     cout << nextOctAsharp7 << '\n';
```

```
//     cout << nextOctD2 << '\n';
//     cout << nextOctF7 << '\n';
//     cout << nextOctFsharp7 << '\n' << '\n';

//     //test getFreq
//     float freqCur = getFreq(curNote);
//     float freqFirst = getFreq(firstNote);
//     float freqLast = getFreq(lastNote);
//     cout << freqCur << '\n';
//     cout << freqFirst << '\n';
//     cout << freqLast << '\n' << '\n';

//     //test withinRange
//     float withinRangeCur = 74.02;
//     float outsideRangeCur = 76.32;
//     bool within_range_bool = withinRange(withinRangeCur, curNote);
//     bool outside_range_bool = withinRange(outsideRangeCur, curNote);
//     cout << within_range_bool << '\n';
//     cout << outside_range_bool << '\n' << '\n';

//     //test motorDirection
//     int motor_direction_tighten = motorDirection(40.1, curNote);
//     int motor_direction_loosen = motorDirection(outsideRangeCur, curNote);
//     int motor_good =  motorDirection(withinRangeCur, curNote);
//     cout << motor_direction_tighten << '\n';
//     cout << motor_direction_loosen << '\n';
//     cout << motor_good << '\n';

// }

void sleepFunc(void){
    sleep_enable();
    attachInterrupt(0, wakeFunc, HIGH);
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    delay(500);
    sleep_cpu();
    return; //after wakeup goes to this line
}

void wakeFunc(void){
    sleep_disable();
    detachInterrupt(0);
}
```