Auto-Guitar Tuner Design Document

Daniel Cho, Ritvik Patnala, Timothy Park

Team 25

TA: Eric Tang

October 14, 2025

Contents

Abstract

This document provides a comprehensive description of the Auto-Guitar Tuner, a handheld device that automates guitar tuning by integrating pitch detection, microcontroller-based control, and motorized peg adjustment. It outlines the system architecture, physical and functional designs, subsystem requirements and verifications, software flow, tolerance analyses, cost considerations, and safety protocols. The tuner is designed to operate as a standalone portable device compatible with standard six-string guitars.

1 Introduction

1.1 Problem and Solution

1.1.1 Problem

When playing guitar, being in tune is essential. When strings are not properly tuned to their correct pitches, the notes played can clash with each other, causing what listeners perceive as being "off" or "out of tune." Accurately tuning a guitar is a challenge for both beginners and experienced players. Traditional tuners require the musician to manually turn tuning pegs while reading pitch information, which can be inconsistent and time-consuming. An automatic solution that can both detect pitch and physically adjust the tuning peg would reduce errors, speed up tuning, and improve usability in practice and performance settings.

1.1.2 Solution

We propose a handheld automatic guitar tuner integrating pitch detection and motorized peg adjustment into one device. The system will capture string vibrations, process them using a microcontroller to identify the current pitch, and automatically rotate the tuning peg with a small motor until the string is in tune. Since the handheld device tunes one string at a time, it can be used on different guitars without worrying about the various spacing between pegs and strings.

A compact LED screen will display the detected pitch and tuning status, while four buttons (Power, String Select, Mode, Start) provide simple user control. The String Select button allows users to cycle through the six guitar strings. Each press moves the selection to the next string in order: low E, A, D, G, B, high E, then back to low E again. This circular navigation lets users easily choose which string to tune without confusion or the need for multiple buttons. The Mode button lets users toggle between preset tuning standards (Standard, Drop D, Open G, etc.) to accommodate various playing styles and preferences. The design will run on a rechargeable battery, with all subsystems integrated into a custom PCB for portability and reliability.

1.2 Visual Aid

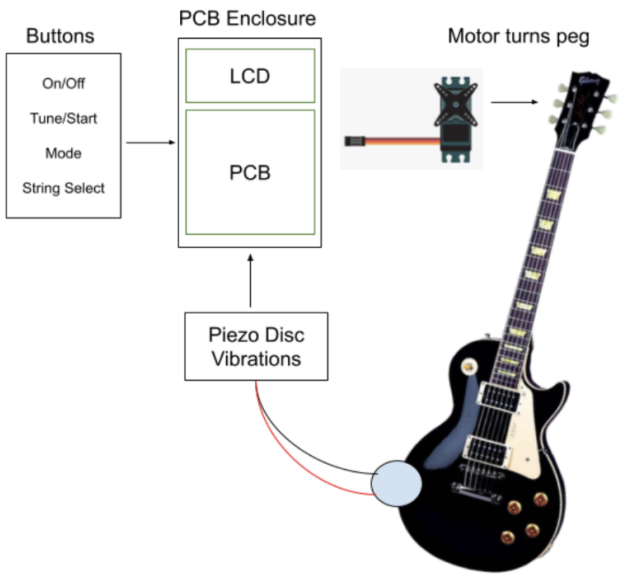

Figure 1: Visual aid diagram of the Auto-Guitar Tuner

1.3 High-Level Requirements

1. **Performance and Accuracy:** The guitar tuner shall accurately detect and display the fundamental frequency of each guitar string within ±2 Hz on the LCD display. The pitch detection algorithm must operate reliably across the full range of standard guitar tuning frequencies (73 Hz for low D to 330 Hz for high E). Real-time feedback must be provided with a maximum latency of 500 ms from string pluck to frequency display update.

2. **Compatibility and Safety:** The tuner shall automatically adjust the tuning peg to bring each string to within ±12 cents of the target pitch for the selected tuning standard. The motor subsystem must implement safety interlocks to prevent excessive torque that could damage the tuning mechanism or break strings, with automatic shutoff if motor stall or overcurrent conditions are detected. The device shall be compatible with standard six-string acoustic and electric guitars featuring conventional machine head tuners.

3. **Usability and Interface:** The user interface shall provide intuitive navigation through at least three tuning standards (Standard, Drop D, Open G) and all six guitar strings (E2, A2, D3, G3, B3, E4) via clearly labeled control buttons. The LCD display must provide clear visual feedback including the currently selected string, target frequency, measured frequency, tuning error in cents, and system status. All user inputs shall be registered within 200 ms, and mode changes shall be reflected on the display without flickering or interruption.

2 Design

2.1 Block Diagram

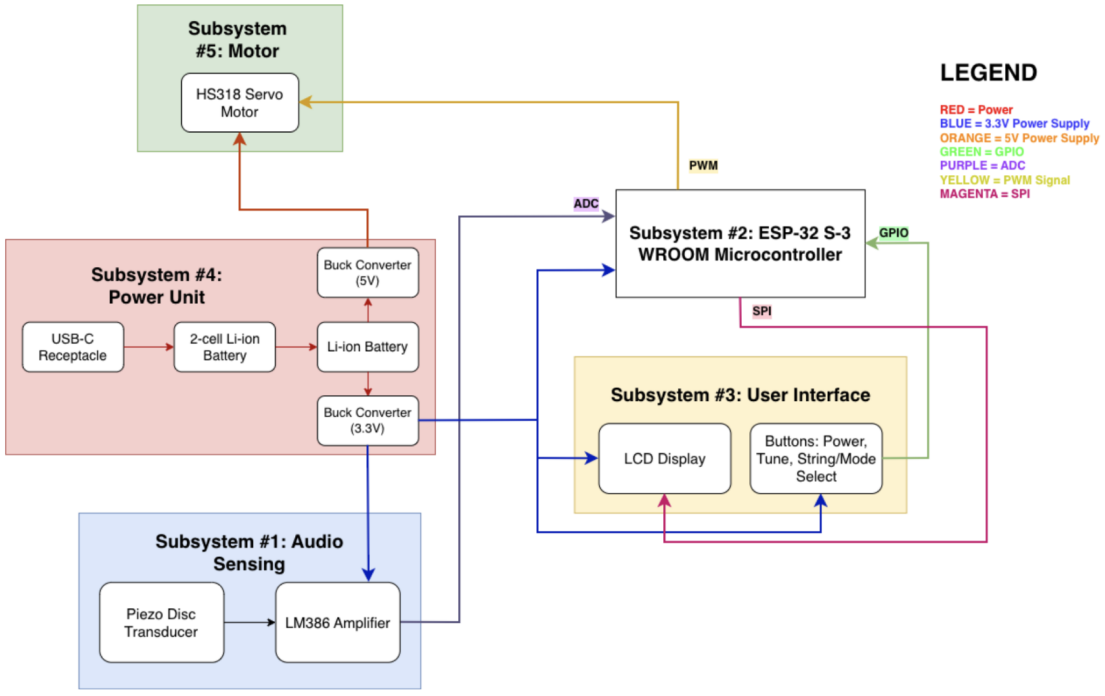

Figure 2: System block diagram showing five main subsystems

The system consists of five main subsystems: Power Management, Microcontroller, Audio Sensing, User Interface, and Motor. The ESP32 microcontroller serves as the central unit, interfacing with all subsystems to coordinate power regulation, audio sampling, digital signal processing, motor control, and user interaction.

2.2 Physical Design

The Auto-Guitar Tuner is contained in a small, portable, rectangular shell. Functionality, usability, and internal electronics protection are prioritized in the physical design. OpenSCAD was used to model and render the device's base and detachable lid in CAD. The enclosure is stable when placed close to the guitar headstock and is the right size to fit comfortably in the user's palm.

A 2.4-inch LCD on the front panel shows the system status and tuning feedback in real time. Four horizontally oriented tactile control buttons for power, string selection, mode switching, and tuning initiation are located beneath the display. During operation, this arrangement enables rapid access with little hand movement. On the side is a USB-C connection for charging.

The motor output shaft and a fork-style coupler that fits normal guitar tuning pegs are located in the top panel. To enable the motor to rotate the peg during automated tuning, this coupler firmly grasps the tuning key. To manage torque without jeopardizing the enclosure's structural integrity, the motor boss is reinforced.

Finally, to ensure that the components are properly aligned with the external features, the design incorporates mounting bosses for the PCB and battery on the inside.

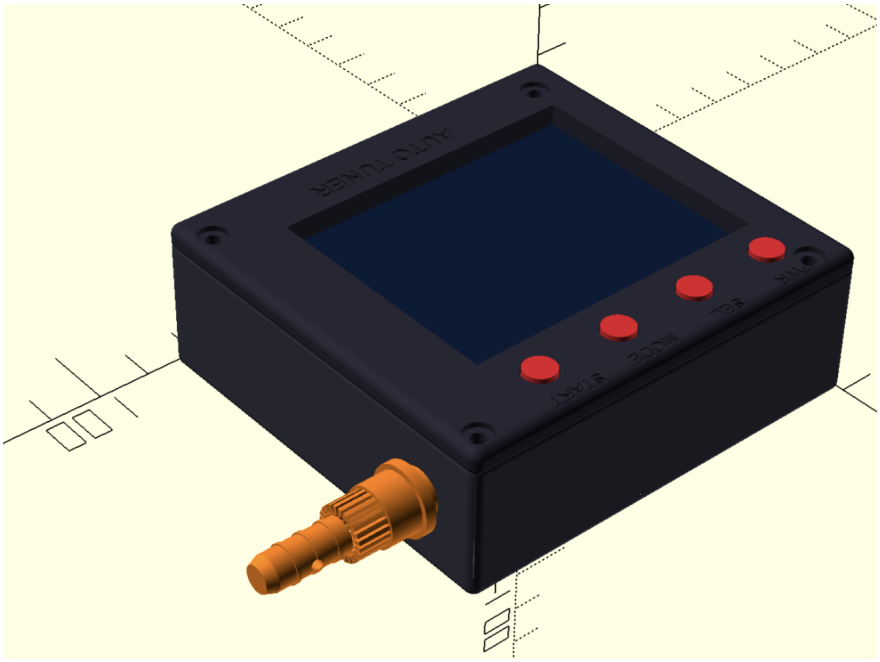

Figure 3: Isometric rendering of the enclosure, highlighting the display, buttons, and turning peg adapter on the motor shaft

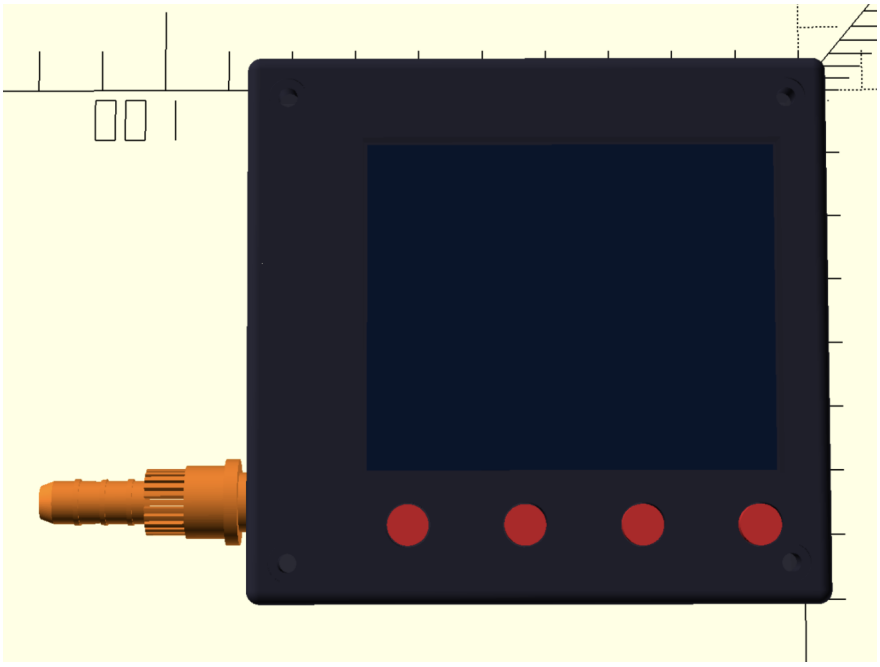

Figure 4: Top view rendering of the Auto-Guitar Tuner showcasing the button and display layout

2.3 Subsystems

2.3.1 Power Management Subsystem

The power management subsystem is responsible for supplying a stable DC voltage to all other subsystems on the PCB. For the initial prototype boards, a **9 V battery input** is stepped down to **5 V** and **3.3 V** using two AP62150WU-7 buck switching regulators. Each regulator is configured with different feedback resistor values to achieve the required output voltages, following the recommended application circuit from the manufacturer's datasheet [2].

The **5 V rail** powers the motor, micro USB-B receptacle, and the amplifier IC, while the **3.3 V rail** supplies the ESP32 microcontroller, the TFT LCD, and part of the audio/vibration subsystem. Since this subsystem provides stable regulated voltages, we bypass the ESP32's onboard voltage regulator and directly power the microcontroller from the 3.3 V line, minimizing power conversion losses.

Figure 5 shows the reference buck converter circuit used for both the 5 V and 3.3 V regulators. Each regulator uses an inductor, two capacitors, and a resistor divider network (R1 and R2) to set the output voltage. The table in Figure 6 summarizes the recommended component values from the AP62150 datasheet for various output voltages, which we used to determine the correct values for both rails. For example, selecting $\mathbf{R1 = 52.3\,k\Omega}$ and $\mathbf{R2 = 10\,k\Omega}$ yields a 5 V output, while $\mathbf{R1 = 31.6\,k\Omega}$ and $\mathbf{R2 = 10\,k\Omega}$ yields a 3.3 V output.

Finally, Figure 7 presents the **full PCB schematic** for the power subsystem, showing the two AP62150 regulators in parallel — one for each voltage rail — and their associated passive components. This configuration ensures each subsystem receives the required stable voltage while maintaining a compact layout and efficient power delivery.

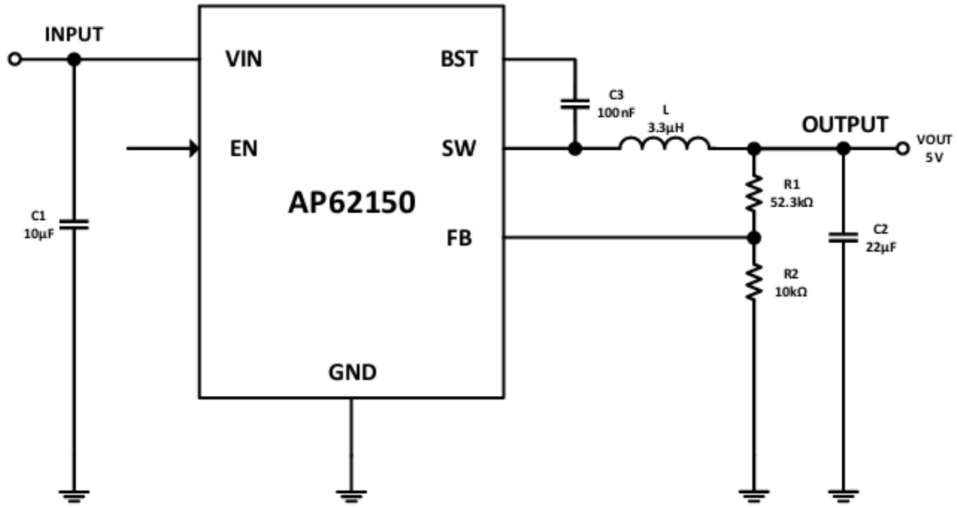

Figure 5: Reference buck converter circuit for AP62150 regulator

AP62150						
Output Voltage (V)	R1 (kΩ)	R2 (kΩ)	L (µH)	C1 (µF)	C2 (µF)	C3 (nF)
1.2	4.99	10	1.2	10	22	100
1.5	8.66	10	1.5	10	22	100
1.8	12.4	10	1.8	10	22	100
2.5	21.5	10	2.2	10	22	100
3.3	31.6	10	3.3	10	22	100
5.0	52.3	10	3.3	10	22	100

Figure 6: Recommended component values from the AP62150 datasheet for various output voltages

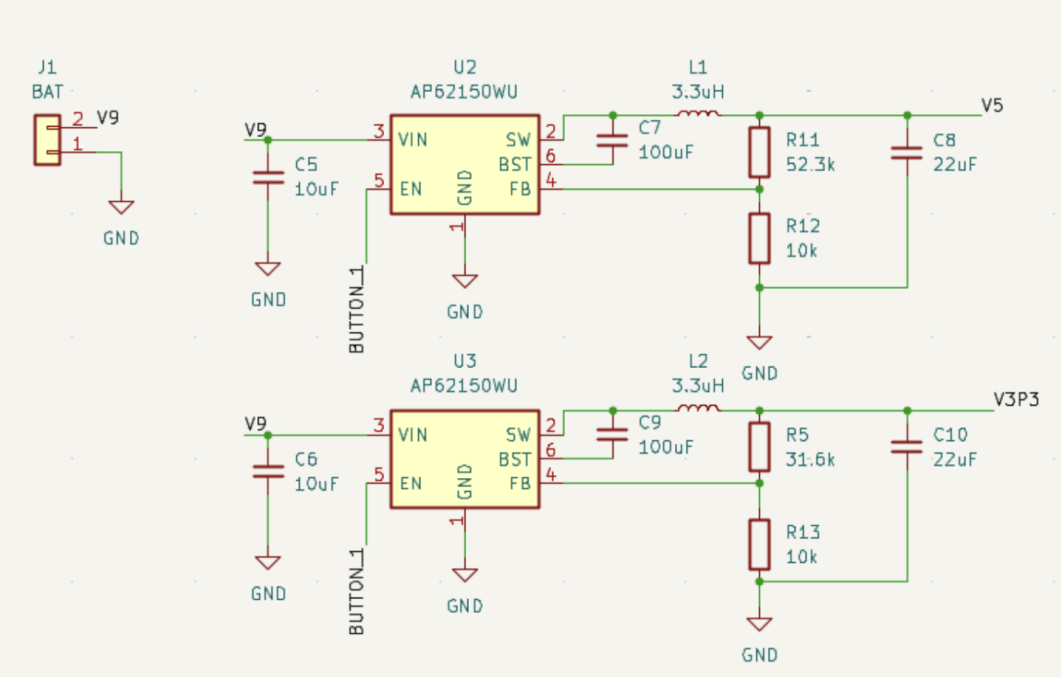

Figure 7: Full power subsystem schematic showing 5 V and 3.3 V buck converters on the PCB

Requirement	**Verification**
Stable 3.3V line with tolerance of 5% (3.135–3.465V)	Measure 3.3V line using oscilloscope
Stable 5V line with tolerance of 5% (4.75–5.25V)	Measure 5V line using oscilloscope
Maintain ±2–3% of nominal output voltage during a fast load step	Use an oscilloscope to measure both output voltage lines, and verify that it remains within 2–3% during a fast load step
Continuous, stable 5V line within ±5% when servo motor stalls	Monitor the 5V rail with an oscilloscope, and intentionally stall the motor

Table 1: Power management subsystem requirements and verifications

2.3.2 Control Subsystem

The control subsystem is essentially the microcontroller. For this project, we chose the ESP-32 S-3 WROOM microcontroller. The purpose of the microcontroller is to first process the audio signal received from the audio subsystem (vibration sensor and amplifier). This implies determining the pitch frequency of the audio signal. The next step is to compare the measured pitch frequency and desired frequency, and translate the difference to PWM commands for the servo motor to rotate the tuning peg. Apart from this, it should also be able to send data using the SPI interface to the display to showcase the string, pitch frequency, how far off the measured pitch frequency is to the desired frequency, and the status of the tuner. The inputs that it takes in are the string select and the mode select.

Parameter	**Value**
Memory	384KB ROM, 512KB SRAM
Peripherals	36 GPIOs, SPI, LCD Interface, UART
Operating Voltage	3.0V – 3.6V
Features	32-bit LX7 microprocessor

Table 2: ESP32-S3-WROOM microcontroller specifications

Specifications We chose this particular chip since it has a much higher computation power and comes with an SPI flash and a 40MHz crystal to program the ESP-32 chip. Apart from this, it also supports all the necessary interfaces, including SPI, GPIOs, and UART. We also had a potential idea to leverage the WiFi and Bluetooth features of this microcontroller by having the microphone or vibration sensor much closer to the sound hole of the guitar and send data over Bluetooth [8].

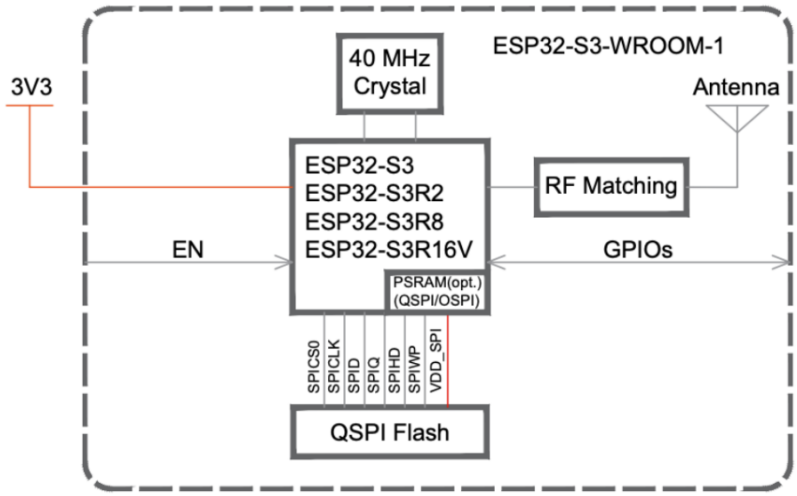

Figure 8: ESP-32 S-3 WROOM Chip

Control Unit: Inputs There are 3 primary inputs that the ESP-32 needs to process. The first 2 are buttons that choose the string and tuning standard/mode. These buttons are debounced and connected to the GPIO pins of the microcontroller. We plan on using polling to constantly read the pin. Based on the tuning standard and the string chosen to be tuned, a pitch frequency is already preset. The last input required is the audio signal from the audio subsystem.

Control Unit: Processing The audio signal from the vibration subsystem is connected to the ADC pin of the ESP32. It is sampled at 4096 Hz for 20 ms frames. A Hanning window is applied to reduce spectral leakage.

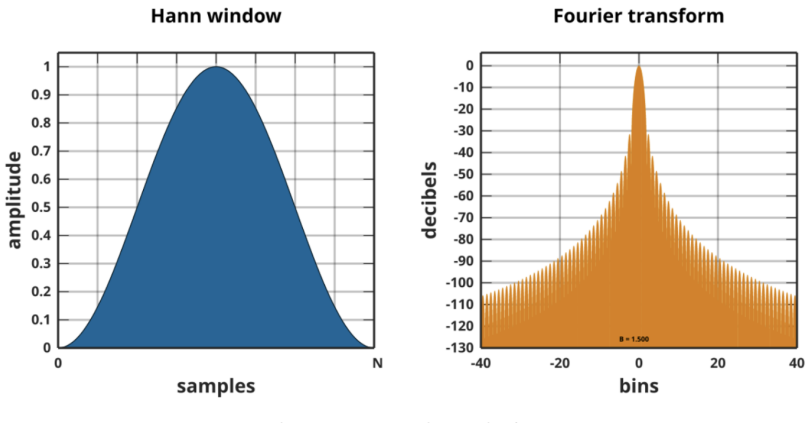

Figure 9: Hamming Window

The pitch detection algorithm will use autocorrelation. Autocorrelation is the correlation of a signal with itself at different delays:

$$R_{xx}[k] = \sum_{n} x[n]x[n+k] \tag{1}$$

where R_{xx} = Autocorrelation Function; $x[n]$ = Sampled Audio Signal, k = Lag.

The maximum is at lag $l = 0$ [9]. R_{xx} can also be calculated in the frequency domain by multiplying the FFT of the audio signal with its conjugate and computing the inverse FFT of the entire signal:

$$R_{xx}[k] = \text{IFFT}(\text{FFT}(x[n]) \times \text{FFT}^*(x[n])) \tag{2}$$

This autocorrelation function is then normalized to $\rho_{xx}[k]$:

$$\rho_{xx}[k] = \frac{R_{xx}[k]}{R_{xx}[0]} \tag{3}$$

This normalized autocorrelation function then goes through zero crossing. This is essentially to see how similar the function is to the time-shifted version of itself by k samples. The value of k that leads to the autocorrelation function being most similar with the time-shifted versions is the lag that we use to compute the pitch frequency:

$$\text{Pitch Frequency} = \frac{F_s}{k} \tag{4}$$

where F_s = Sampling Rate; k = Lag.

Once the pitch frequency is detected using the method above, it needs to be compared to the desired frequency of the string in that particular tuning standard. We plan on using a Proportional Integral Derivative (PID) controller to achieve this. A PID controller is an instrument that receives input data, calculates the difference between the actual value and the desired point, and adjusts the output to control variables, in our case, the motor [10].

Control Unit: Output There are 2 primary outputs from the microcontroller. As mentioned earlier, the output of the PID controller can be used to determine the number of degrees to rotate the motor. This output is sent in the form of a pulse width modulation signal from a GPIO pin to the servo motor. See section 2.3.5 for more on this.

The next output is to the display. The display showcases the string being tuned at the moment, the frequency, and how far off the current frequency is to the desired frequency. The data is sent to the display using the SPI interface. Other pins on the display such as the chip select, clock, reset and backlight are all connected to GPIO pins on the ESP32.

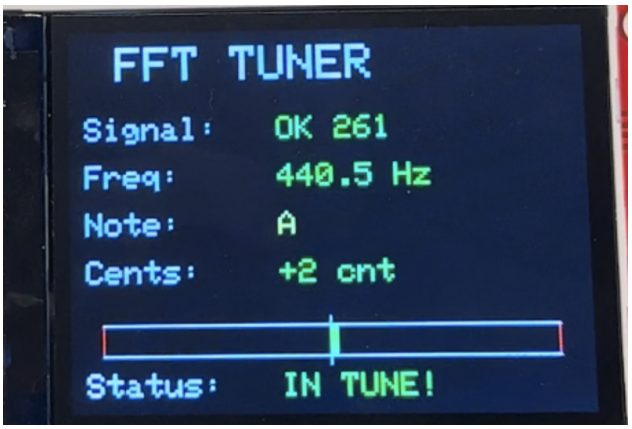

Figure 10: LCD Display

Programming To program the ESP32, we have to set up a USB Receptacle. We chose to use the USB 2.0 Micro B receptacle. The receptacle uses a differential pair for signal integrity to program it. The D+ and D- pins need to be connected to the USB_D+ and USB_D- of the microcontroller. [11]

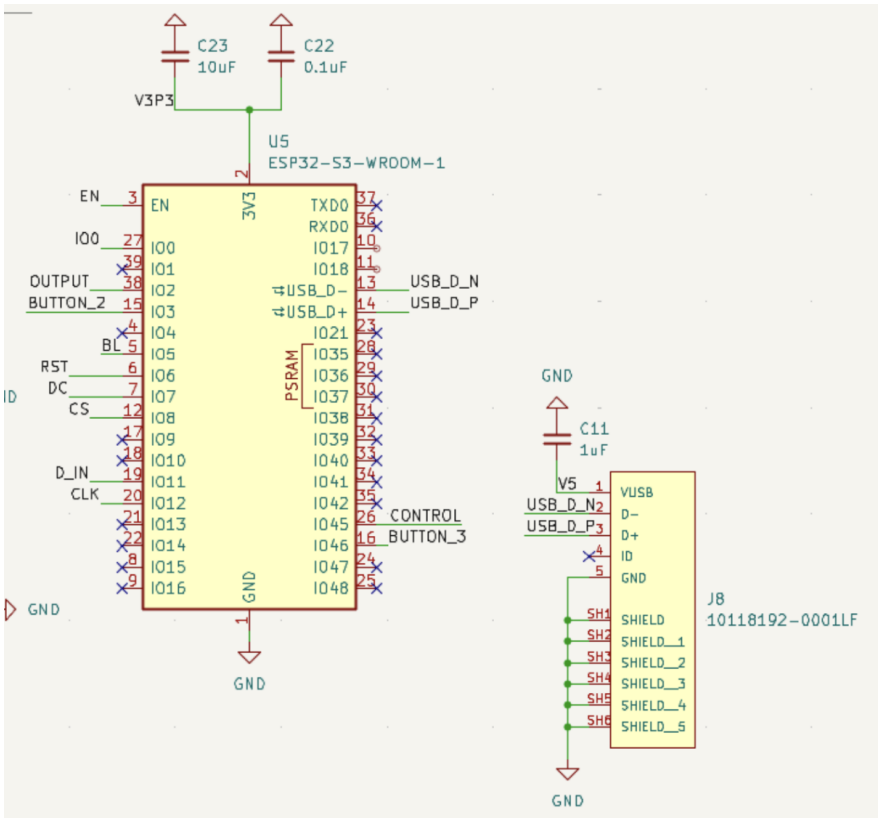

Figure 11: Control Subsystem Circuit Diagram

Requirements	**Verification**
Pitch to Rotation: ESP32 shall detect the string's fundamental frequency and compute the required tuner rotation to reach the target pitch.	A function generator can be used to replicate the strumming of the string. An oscilloscope can be used to look at the PWM signal coming out of it to see if it is the correct amount of rotation.
LCD Status &UX: ESP32 shall render real-time status to the LCD without disrupting sensing and control.	Run tuning cycles to see if the display showcases the correct string, how far off the pitch is, and when the guitar string is in tune.
Safety Interlocks: Block motor commands when unvoiced/noisy frame or overcurrent and stall occurs.	Trigger each safety condition (unvoiced frame, overcurrent, stall) and on a scope, timestamp the fault event edge, the PWM disable edge, and the LCD fault flag. If the PWM is off for approximately 100ms and the LCD shows the fault within 100ms of the fault.

Table 3: Control subsystem requirements and verifications

2.3.3 Audio/Vibration Subsystem

The vibration subsystem is responsible for recognizing the current pitch of a guitar string. We plan to use a piezo disc sensor, and place it onto the guitar so it can pick up the mechanical vibrations generated when a string is plucked. To amplify the piezo signal, we're using the LM386-MX1 amplifier. This amplifier seems to be perfect for this application since it is meant for low-voltage applications, and the signal generated from the piezo will be in the 100–500mV RMS range.

At the V_{out} pin, we've connected the output in between two 10kΩ resistors, so that the 3.3V gets cut and the signal centers around ∼1.65V. This is to ensure the signal will fit within the 0–3.3V range for the ESP32 ADC input and maximizes the frequency resolution. From there, we designed a low pass filter with the R9 (10kΩ) resistor and C12 (0.01μF) capacitor to attenuate the higher frequencies.

Using the formula:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \times 10000 \times 0.01 \times 10^{-6}} \approx 1591 \text{ Hz} \tag{5}$$

This filter has a cutoff frequency around 1591Hz, which is perfect for our application. The highest fretted note on the guitar (24th fret, E-string) is ∼1318Hz, which is within this range. But since we will be tuning with open strings, the open E string should have a frequency of around 329Hz. Both of these frequency points are less than 1591Hz, so this filter will be good for eliminating noise and some higher partial harmonics.

The C19 (0.1μF Capacitor) will be used to stabilize the V_S voltage used to power the amplifier, and the C1 (0.1μF Capacitor) will be used to eliminate any DC offset from the piezo signal, and only allow the AC signal to pass through into the amplifier. The 1MΩ resistor is used to provide high impedance at the input, to make sure the full piezo signal can be captured and carried into the amplifier.

Both the gain pins (pins 1 and 8) are left unconnected in our schematic since the default gain value for this amplifier is 20 (26 dB).

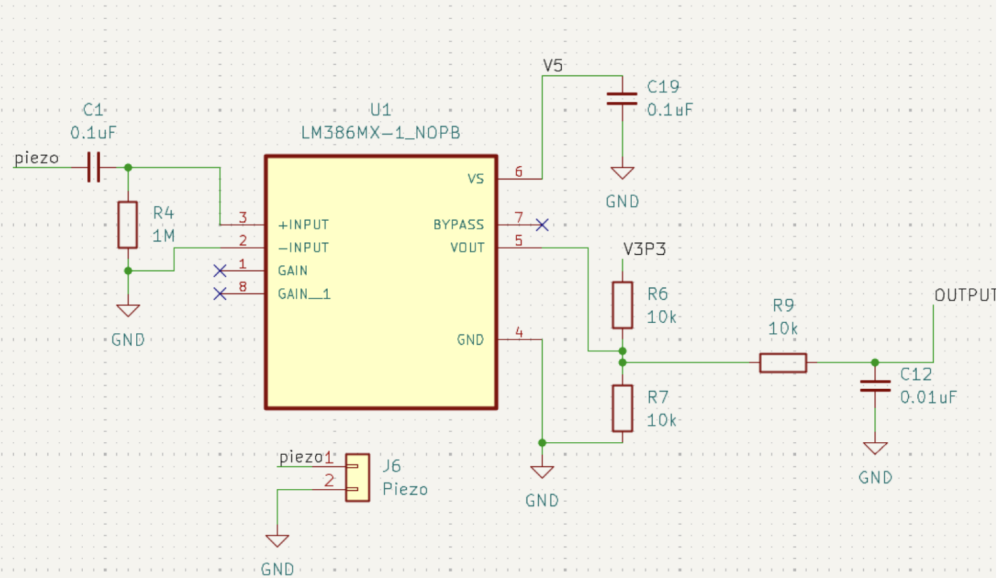

Figure 12: Audio/vibration subsystem schematic

Requirement	Verification
Piezo input signal $\geq$ 10mV	Measure the voltage across the two piezo disc wires to ensure signal measures above 10mV
Output signal amplitude must be 20x as large as the piezo input amplitude	Measure the amplitude of the input piezo signal as well as the output of the LM386 amplifier using an oscilloscope under the same conditions
Current drawn from power supply pin should be less than 4mA	Insert a multimeter to measure current between the 5V rail and the V_5 pin

Table 4: Audio subsystem requirements and verifications

2.3.4 User Interface Subsystem

Purpose and Overview Throughout the tuning process, the User Interface (UI) subsystem controls all user interactions and offers visible feedback in real time. Four tactile buttons (Power, String Select, Mode, and Start) plus a small LCD display coupled to the ESP32 microcontroller via SPI make up the interface. Users can choose target strings, set up tuning modes, start tuning operations, manage device power, and monitor system status with this subsystem's clear visual feedback. Rapid, distraction-free tuning is made possible by the design's emphasis on straightforward operation, which includes few controls and clear feedback.

Hardware Components

LCD Display

- Operating voltage: 3.3V
- Interface: SPI
- Operating current: ∼30mA
- Display content: Current string selection, target frequency, measured frequency, tuning error

Control Buttons

- Type: 4x momentary switches
- Connection: ESP32 GPIO pins with internal pull-up resistors
- Logic: Active-low
- Debouncing: Software-based

UI State Machine The subsystem manages the display and the user input by utilizing a Finite State Machine (FSM) as follows.

State	Description	Transition
Boot	Initialize GPIO, SPI peripherals, and display	→ Ready (after system initialization)
Ready	Display current string and mode selection; await user input	→ Tuning (Start button pressed) → Standby (timeout) → Fault (system error)
Tuning	Show real-time frequency, error magnitude (cents), and motor direction	→ Success (in-tune condition met) → Aborted (Start button pressed) → Fault (system error)
Success	Display "In Tune" confirmation	→ Ready (after 2s or button press)
Aborted	Show cancellation message	→ Ready (after 1s)
Fault	Display error message (motor stall, input clipping, low battery)	→ Ready (after acknowledgement)

Table 5: UI Finite State Machine states and transitions

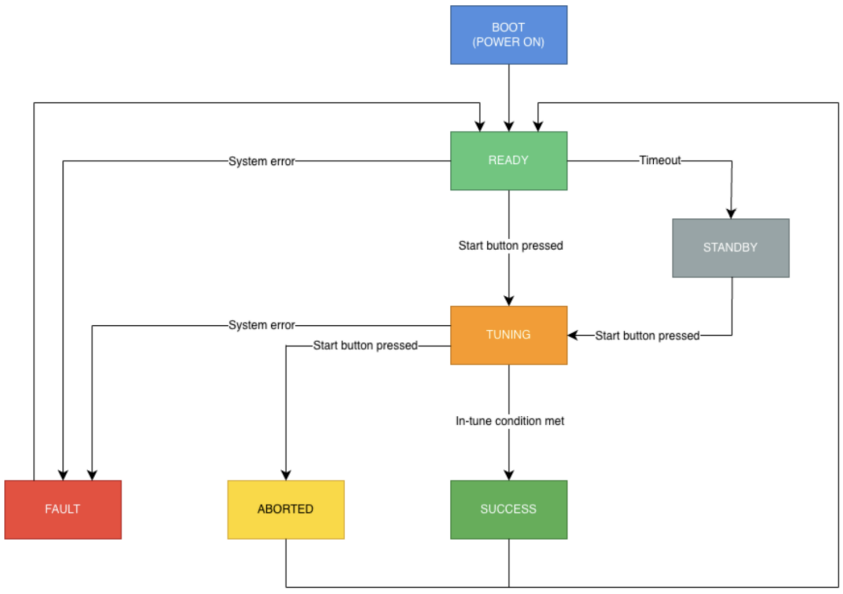

Figure 13: Diagram depicting the logic flow of the UI Finite State Machine (FSM)

Button Functions Through a combination of short and long push operations, the four control buttons offer full user interaction capabilities. Long presses necessitate holding the button for more than 1.5 seconds, and short presses are defined as any button activation lasting less than 1.5 seconds. This dual-purpose method preserves intuitive functioning while reducing the amount of physical controls.

The Power button only reacts to lengthy presses to turn the device's power on or off; it lacks a short-press capability. Unintentional power cycling during regular use is avoided thanks to this thoughtful design decision.

With each brief click of the String Select button, the six standard guitar strings are cycled through in ascending sequence (E2 → A2 → D3 → G3 → B3 → E4), returning to E2 after E4. Users can swiftly move backward through the string sequence without continuously cycling forward by long-pressing this button, which reverses the cycle direction.

Standard tuning, drop D, and open G are among the tuning modes that can be selected by short pushes on the Mode button. When navigating between modes that are not contiguous, a long press reveals a separate mode selection menu that shows every mode that is available at once for direct selection.

Depending on the status of the system, the Start button's functionality varies depending on the situation. A quick press starts the tuning process when the device is in the Ready state. Pressing Start again while active tuning instantly ends the process and puts the device back in Ready mode. Long presses from any state bring up a help overlay with basic usage instructions and button functionalities; this overlay automatically disappears after a few seconds or when a button is touched.

Button	**Short Press**	**Long Press (> 1.5s)**
Power (Toggle switch)	–	Toggle device power
String Select	Cycle forward (E2 → A2 → D3 → G3 → B3 → E4 → E2)	Cycle backward
Mode	Cycle through tuning modes	Open mode selection menu
Start	Begin / Stop tuning	Display help overlay

Table 6: Operations of the four momentary switches

Requirement	**Verification Method**
Button press must be registered within 200ms	Measure response with a logic analyzer
LCD must display the updated string selection	Cycle through strings and verify the display
Mode switching must change target frequencies	Toggle modes, and verify the displayed reference
Display must refresh without flickering	Observe the continuous tuning display during operation

Table 7: UI subsystem requirements and verifications

2.3.5 Motor Subsystem

The motor subsystem is responsible for rotating the tuning pegs on the guitar. This subsystem consists of the servo motor (HS318) and is controlled by sending an electrical pulse of variable width or pulse width modulation (PWM). This motor is made up of a small DC motor, a potentiometer, and a control circuit. Gears are attached to the control wheel. As the motor starts to rotate, the potentiometer's resistance changes so the control circuit can precisely regulate how much movement there is to be done and in which direction. The potentiometer essentially provides live feedback about the shaft's angle. The control circuit compares where the shaft is (potentiometer reading) and where it should be (PWM signal). The difference tells whether the motor should rotate in one direction or the other, and when to slow down and stop at the target angle [1].

Parameter	Value
Operating Voltage (Volts DC)	4.8V – 6.0V
Max Torque Range (kg cm)	3.0–3.7
Current Draw at Idle/No-Load/Stall (mA)	8/180/800
Circuit Type	Analog
Physical Specifications	Standard 25 Output Shaft; Plastic Casing; Cored Metal Brush, Nylon Gear
Range of Motion	180°

Table 8: HS318 servo motor specifications

Design Choice Our initial choice was to use a stepper motor over a servo motor. The differences between the 2 motors are highlighted in Table 9 [6].

	Stepper Motor	Servo Motor
Control Method	Moves in fixed steps	Uses potentiometer feedback
Torque	Torque drops at higher speeds	High torque across the range
Precision	Depends on the step size	Continuous position feedback allows for higher precision tuning
Flexibility	Cannot react to changes in load	Better choice for variable load systems

Table 9: Comparison between stepper motor and servo motor

One of our high level requirements is for the tuner to be able to get the desired pitch within 12 cents. For such high precision, a stepper motor isn't ideal without the feedback loop of the servo motor. Moreover, different strings have different tensions and therefore loads making it difficult to work with a stepper motor.

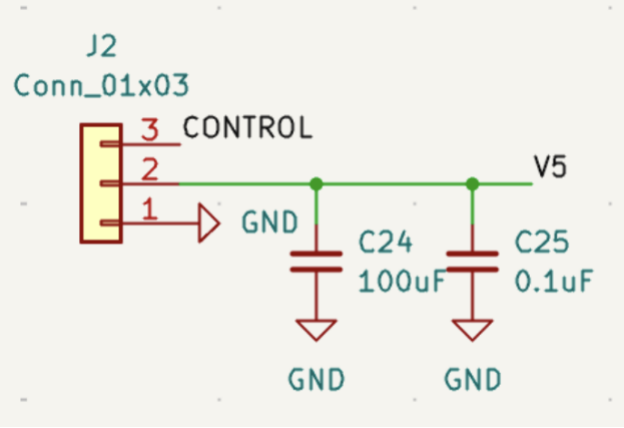

Figure 14: Motor subsystem schematic

The circuit diagram above showcases how we are connecting the servo motor to the PCB and the ESP-32 microcontroller. The control signal is essentially the PWM signal that the motor will receive from a GPIO pin on the microcontroller. The ceramic capacitor is placed closer to the servo's power pins to filter out high-frequency noise and voltage spikes that could be caused by nearby digital circuitry and PWM motor switching. The electrolytic capacitor is used to store much more charge and respond to low frequency dips and surges in current draws [5].

Servo Motor Control As mentioned, the PWM signal sent to the motor will determine the position of the shaft, and based on the duration of the pulse, the rotor will turn to the desired position. 3 parameters characterize the waveform: pulse voltage, pulse width, and pulse period. The pulse voltage for the HS318 servo motor is approximately 3.3V to 5V. For the most part, the signal amplitude does not matter much, as long as it is high enough for the servo to register the pulses. The pulse width is what controls the motor. The servo motor expects a pulse every 20ms [4]. We can control how much the servo moves by setting a new pulse width that corresponds to the desired end position, using its internal potentiometer for feedback.

Pulse Width	**Behavior**
1.5ms	**Neutral:** HS318 servo motor has a neutral angle at 90°. This implies that the motor can move 90° in either direction.
$<$ 1.5ms	**Counter-Clockwise:** The servo motor will move counter-clockwise for pulse widths lesser than 1.5ms.
$>$ 1.5ms	**Clockwise:** The servo motor will move in the clockwise for pulse widths greater than 1.5ms.

Table 10: Servo motor pulse width control

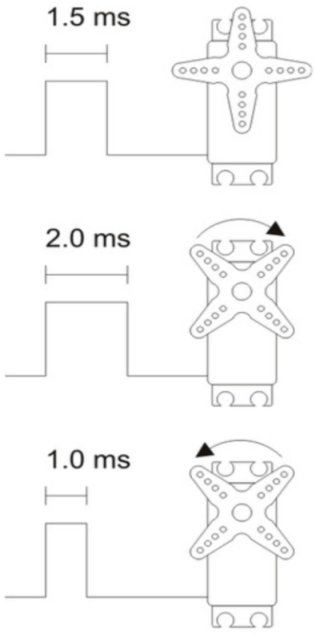

Figure 15: Relationship Between Control Pulse Width and Servo Motor Shaft Position

Torque Analysis Based on the specifications of the HS318 servo motor, the maximum available torque to us is 3.7 kg cm. The first step is to determine the torque required at the string post. The tension of a string ranges from 60N – 80N, and the typical radius of a post is 3mm = 0.003m.

$$\tau_{\text{post}} = T \times r_{\text{post}} = 80\text{N} \times 0.003\text{m} = 0.24\text{Nm} \tag{6}$$

where τ = Torque; T = Tension of String; r_{post} = Radius of the string post.

Our motor is going to be connected to the tuner knob/tuning peg. Hence, the torque required from the motor to rotate this knob is given by the formula below:

$$\tau_{\text{motor}} = \frac{\tau_{\text{post}}}{\text{ratio} \times \eta} = \frac{0.24\text{Nm}}{14 \times 0.7} = 0.0245\text{Nm} = 0.25\text{kg} \cdot \text{cm} \tag{7}$$

where η = efficiency.

Most guitar tuning machines use a worm and gear mechanism, so when we rotate the knob, the post turns more slowly. The ratio of the peg to post is 14:1 for typical acoustic guitars. The Mechanical efficiency takes into account friction and other forces that may affect the tuning and is typically 0.7.

Based on these calculations, the torque required from the motor is 0.25 kg cm, which is much lesser than the maximum available torque for the motor (3.7 kg cm).

Requirement	**Verification Method**
Angular Resolution: Servo motor should be able to rotate the tuning peg with an accuracy of 1°, which translates to approximately 5–7 cents on an acoustic guitar.	Command smaller movements to random angles and measure the actual movement using an external reference such as a shaft encoder.
PWM Command: The servo motor should be able to receive the correct PWM based on the difference between the current and desired frequency.	Measure the PWM signal using an oscilloscope at the control pin of the servo motor or the GPIO pin of the microcontroller. The pulse width should mathematically translate to the necessary rotation.
Motor Position: Servo motor should rotate to the desired position (±1°).	Once the motor rotates to the position it was commanded to, the microcontroller should register that the string is in tune when it is strummed again. No further rotations should be seen as well. The console/Display should show that the string is in tune.

Table 11: Motor subsystem requirements and verifications

2.4 Tolerance Analysis

One of our biggest concerns for our project is the vibration system. Being able to accurately capture the current guitar string's pitch is the single most crucial aspect of our system, since it affects how the motor turns the peg so that the motor can make the correct adjustments to the tuning pegs. Our piezo sensor detects mechanical vibrations through the body of the guitar and converts the mechanical energy into an electric signal that can be processed by the microcontroller. This approach is more feasible compared to using a microphone, since we won't have to worry about ambient noise or other loud environments.

However, the piezo sensors we're using (B0B2QS8VK5 35mm piezo disc) have material and manufacturing tolerances. The resonance frequency is specified around 3–5 kHz ±0.5 kHz. Luckily,

the guitar string frequencies we are interested in are in the range from a low D2 (73 Hz) to a high E4 (330 Hz). Since the resonance frequency sits far above the target frequencies we desire, the frequencies we desire below this range can be connected with linearity. So we know our piezo sensors will be able to capture all the notes we need.

One of the key reasons we chose to use the LM386MX-1 amplifier IC is that it has adjustable gain values. This way, if a specific piezo sensor has a varying sensitivity, we can adjust the gain so that we won't have to rework our whole vibration amplification system. For now, we're planning to use a gain of 20, but we might plan to change the gain value if we need to compensate for changes in mounting strategy or part tolerances.

3 Cost and Schedule

3.1 Cost Analysis

3.1.1 Parts Cost

Table 12: Bill of materials with costs

Component	**Part Number**	**Manufacturer**	**Qty**	**Price**	**Use**
ESP32 Microcontroller	ESP32-S3-WROOM-1-N16R8	Espressif Systems	1	$6.56	Breadboard (chosen before PCB design)
ESP32-S3 DevBoard	ESP32-S3-DevKitC-1-N32R16V	Espressif Systems	1	$17.00	Breadboard demo 2, prototyping
Mic Amplifier	MAX9814	Adafruit	3	$9.99	Initial breadboard, before piezo
35mm Piezo Disc	B0B2QS8VK5	YQBOOM	10	$9.99	PCB
2.4inch LCD Display	B08H24H7KX	Waveshare	1	$18.99	PCB
2.2inch LCD breakout	B01CZL6QIQ	HiLetgo	1	$14.49	Breadboard demo 1, UI prototyping
Buck Switching Regulator	AP62150WU-7	Diodes Inc.	2	$0.66	PCB
USB micro B Receptacle	10118192-0001LF	Amphenol ICC	1	$0.43	PCB
Amplifier IC	LM386MX-1	Texas Instruments	1	$0.85	PCB
Tactile Switch	KMR232NGULC LFS	C&K	2	$1.40	PCB
100μF Electrolytic Capacitor	35ZLH100MEFC6.3X11	Rubycon	1	$0.32	PCB
3.3μH Fixed Inductor	VLS6045EX-3R3N	TDK Corporation	2	$0.68	PCB
31.6kΩ Resistor	ERA-6AEB3162V	Panasonic	1	$0.10	PCB
52.3kΩ Resistor	ERA-6AEB5232V	Panasonic	1	$0.10	PCB
			Total:	**$81.56**	

3.1.2 Labor Costs

According to the Grainger Engineering career services salary and hiring data portal, the average graduate makes $96,270 a year [7]. Assuming approximately 2080 hours a work week, that is approximately $46.28 per hour. Since this is a 16-week course, and we can expect about an average of 15 hours a week, the labor cost is:

$$\$46.28 \times 16 \text{ weeks} \times 15 \text{ hours/week} = \$11,107.20 \tag{8}$$

And since our group has 3 team members:

$$\$11,107.20 \times 3 \text{ members} = \$33,321.60 \tag{9}$$

3.1.3 Total Cost

$$\text{Total Cost} = \$33,321.60 \text{ (labor)} + \$81.56 \text{ (parts)} = \mathbf{\$33{,}403.16} \tag{10}$$

3.2 Schedule

Table 13: Project schedule with weekly goals and team assignments

Week	Goals	Assigned
9/29	• Pick components, verify parts can handle power and current levels • Design PCB • Get feedback at the PCB review • Prototype motor and basic pitch processing on the ESP Dev-board	• Tim, Ritvik • Tim, Ritvik • All • Daniel
10/6	• Finalize PCB • Finalize working motor and microphone for breadboard demo 1 • Layout, routing • Breadboard demo 1	• Tim, Ritvik • Tim, Ritvik • Daniel • All
10/13	• Finish Design Doc • Evaluate Teamwork • Submit parts order • Talk to Greg to finalize machine shop order for PCB casing • Begin working/prototyping on UI for LCD display • Second Round PCB order	• All • All • Tim • Ritvik • Daniel • All

Continued on next page

Table 13 – *Continued from previous page*

Week	Goals	Assigned
10/20	• Prototype audio/vibration subsystem • Prototype UI subsystem on LCD module • Test different standard tuning modes to make sure pitch is properly recognized • Verify amplifier circuit on breadboard	• Daniel • Daniel • All • Tim, Ritvik
10/27	• Finalize microcontroller I/O and programming to make sure algorithms work and data is handled properly • Breadboard demo 2 • Review subsystem results and discuss changes to the PCB	• Daniel • All • All
11/03	• Submit third wave PCB order (if needed) • Individual progress report • Begin assembling PCB • Test PCB	• Daniel • All • Tim • Ritvik
11/10	• Test PCB • Unit test all subsystems	• Daniel • Tim
11/17	• Final assembly and testing • Test and verify subsystem requirements • Accomplish all 3 high-level requirements • Mock demo	• All • All • All • All
11/24	**Fall Break**	–
12/01	• Edits, final changes/bug fixes • Final Demo • Begin preparing for the final presentation • Start write-ups for final report	• All • All • All • All
12/08	• Submit final papers • Give Final Presentation • Make final updates to individual lab notebooks	• All • All • All

4 Discussion of Ethics and Safety

The IEEE and ACM Codes of Ethics, which place a strong emphasis on putting the public's welfare first, guaranteeing safety, being truthful about capabilities, and accepting responsibility for the results of design choices, are followed in the creation of the Auto-Guitar Tuner. Our objective is to create a device that is both morally and technically sound.

4.1 Ethical Considerations

4.1.1 Transparency and Honesty

We fully explain the tuner's performance features, such as its single-string operation, estimated battery runtime, and tuning precision of ± 12 cents. In our lab reports, we will record all testing findings, including failures, and refrain from inflating performance.

4.1.2 Accessibility and Fairness

The UI is designed with readable fonts, intuitive layouts, and clear audio cues so that it can be used by beginners and experienced players alike. Simple controls and immediate feedback make the tuner accessible without requiring prior technical knowledge.

4.1.3 Intellectual Property and Environmental Impact

For components like the servo motor, we exclusively use publicly available datasheets and appropriately referenced open-source libraries (e.g., Arduino FFT). No proprietary designs or code are utilized without authorization. The tuner minimizes waste from disposable batteries by utilizing a rechargeable Li-ion battery. When feasible, components are chosen from reliable vendors who comply with RoHS.

4.2 Safety Considerations

The motor subsystem, which directly transfers torque to tuning pegs, is the main focus of the tuner's mechanical safety. Several safeguards are in place because excessive torque could harm the pegs or strings. By controlling the PWM duty cycle, the firmware limits motor torque and keeps it below levels that could damage the instrument. The motor reduces the possibility of overshooting the intended pitch by operating in tiny, gradual steps between frequency measurements. Firmware-imposed rotation constraints stop the peg from turning past safe limits in a single tuning operation, and a slip coupler acts as a passive mechanical safety measure by disengaging under excessive force.

In terms of electrical safety, the gadget makes use of a Li-ion battery that has built-in overcurrent and overdischarge protection as well as a certified charger integrated circuit. Both a fuse and a TVS diode protect against short circuits and reversed connections, and the USB-C charging input has ESD protection. The interior wiring is completely insulated and sealed to reduce the possibility of electrical shock or user contact, and all circuits run at a safe low voltage of 3.3V.

Because the firmware directly controls the motor, software safety is essential. If the firmware hangs, uncontrollable behavior is prevented by tweaking timeouts and watchdog timers. Only after stable pitch detection does the motor actuate; if the signal is loud or unstable, the motor stays inactive. The system instantly stops working and notifies the user of an error if it detects a low battery, ADC saturation, or motor stall.

Finally, explicit instructions for correct coupler installation, tuning, and charging are provided to ensure user safety. The user interface warns of abnormal conditions, including low battery, unstable pitch, or high mechanical resistance. The tuner is made for conventional six-string guitars. Together, these safeguards guarantee that the instrument and user can use the equipment securely.

4.3 Risk Analysis

If not handled correctly, the mechanical, electrical, and software parts of the auto-guitar tuner could endanger the player as well as the instrument. The main mechanical risk is that the tuning pegs, strings, or the tuner's coupler could be damaged if the motor applies too much torque to them. This risk is considerably decreased by a slip coupler, incremental motor adjustments, and firmware torque restrictions. The control logic's rotation limitations further guarantee that the motor never turns over safe limits in a single operation.

Li-ion batteries and power regulation circuitry are the primary electrical hazards. Damage or overheating may result from flaws such as reverse connections, short circuits, or overcharging. We incorporate a fuse and TVS diode, utilize a certified charger IC with current and thermal protection, and adhere to standard PCB design procedures for isolation and grounding to reduce these hazards. To reduce potential risks, all electronics run at low voltages.

The firmware's control over motor action is linked to software and control risks. The motor could spin constantly or in the wrong direction if the firmware hangs or misinterprets pitch signals. These situations are avoided via signal validation checks, tuning timeouts, and watchdog timers. The firmware stops the motor and shows a clear error message when there is any malfunction, such as motor stall, low battery, or ADC clipping.

Risk	**Potential Consequence**	**Mitigation Strategy**
Excessive motor torque	Damaged turning peg or string	Firmware torque limit, incremental steps, rotation bounds
Battery fault or short	Overheating, fire hazard, circuit damage	Certified charger IC, fuse, proper PCB layout
Firmware hang or incorrect control	Continuous or unsafe motor rotation	Watchdog timer, timeouts, safe GPIO defaults
User misuse	Mechanical stress, poor tuning	Clear instructions, physical alignment features
Low battery during tuning	Brownout or device instability	Battery monitoring, lockout under critical voltage

Table 14: Risk analysis for the Auto-Guitar Tuner

References

[1] Jameco Electronics, "How Servo Motors Work," https://www.jameco.com/Jameco/workshop/Howitworks/how-servo-motors-work.html

[2] Diodes Incorporated, "AP62150 Datasheet," https://www.diodes.com/assets/Datasheets/AP62150.pdf

[3] Texas Instruments, "LM386 Low Voltage Audio Power Amplifier," https://www.ti.com/lit/ds/symlink/lm386.pdf

[4] Pololu, "Servo Control Interface in Detail," https://www.pololu.com/blog/17/servo-control-interface-in-detail

[5] EE Power, "Electrolytic Capacitor Guide," https://eepower.com/capacitor-guide/types/electrolytic-capacitor/

[6] EE Power, "Stepper Motor vs Servo Motor," https://eepower.com/capacitor-guide/types/electrolytic-capacitor/

[7] Grainger College of Engineering, "Salary and Hiring Data Portal," https://ecs.grainger.illinois.edu/salary-data/data-portal

[8] Espressif Systems, "ESP32-S3-WROOM-1 Datasheet," https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf

[9] SciCoding, "Pitch Detection Using Autocorrelation," https://scicoding.com/pitchdetection/

[10] National Instruments, "PID Theory Explained," https://www.ni.com/en/shop/labview/pid-theory-explained.html

[11] Amphenol ICC, "USB Micro B Receptacle Datasheet," https://cdn.amphenol-cs.com/media/wysiwyg/files/documentation/datasheet/inputoutput/io_usb_micro.pdf