Auto-Guitar Tuner

Final Report

Team 25

Daniel Cho, Ritvik Patnala, Timothy Park

TA: Eric Tang

ECE 445 Senior Design

University of Illinois at Urbana-Champaign

December 10, 2025

Contents

Abstract

This report presents the design, implementation, and verification of an automatic guitar tuner that integrates pitch detection with motorized tuning peg adjustment. The system utilizes a piezo vibration sensor and LM386 amplifier to capture string vibrations, an ESP32-S3 microcontroller for digital signal processing and control, and an HS318 servo motor for automated peg rotation. The device features a 2.4-inch TFT LCD display and tactile button interface for user control across multiple tuning standards.

1 Introduction

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.

1.2 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 two buttons (Toggle, Select) provide simple user control. The buttons allow users to cycle through the six guitar strings, or auto tune each string. The toggle button allows the user to cycle through different strings or tuning standards, while the select button allows users to select or move back from a menu. The design will run on a 9V battery, with all subsystems integrated into a custom PCB for portability and reliability.

1.3 High-Level Requirements

1. **(Performance)** Our guitar tuner should be able to accurately capture each string's pitch, and display it on the OLED.
2. **(Compatibility & Safety)** The tuner should accurately rotate the tuning peg, and get the desired pitch within 12 cents.
3. **(Usability and Interface)** The user should be able to cycle through the tuning standards, and strings on the LCD display.

2 Design

2.1 Design Procedure

Our project procedure started with clear team expectations, which were covered in a team contract. This document was crucial for establishing responsibilities, defining our objectives, and ensuring we had a clear communication system. We delegated work based on software/-control and hardware (PCB) design. The software duties were closely tied to the pitch recognition, UI, and control, which involved a lot of prototyping on a breadboard and da ev board. The hardware team focused more on the core PCB and hardware design, including power distribution, I/O between subsystems, and achieving functionality in accordance with our high-level requirements.

During the initial prototyping stage, our group communicated and worked closely together while designing and testing subsystems on the breadboard. For example, the hardware design choices and findings involving noise greatly affected the software configurations for filtering and note recognition. On top of that, the limitations of the final PCB affected our overall design choices (footprint sizes, trace widths, and overall spacing). Once our PCB was soldered, we all had to take charge of various subsystems to make sure they met core functionality requirements. Some unexpected challenges or issues forced us to be flexible when integrating the system as a whole. But having clear goals and open communication allowed us to achieve our goals for the semester.

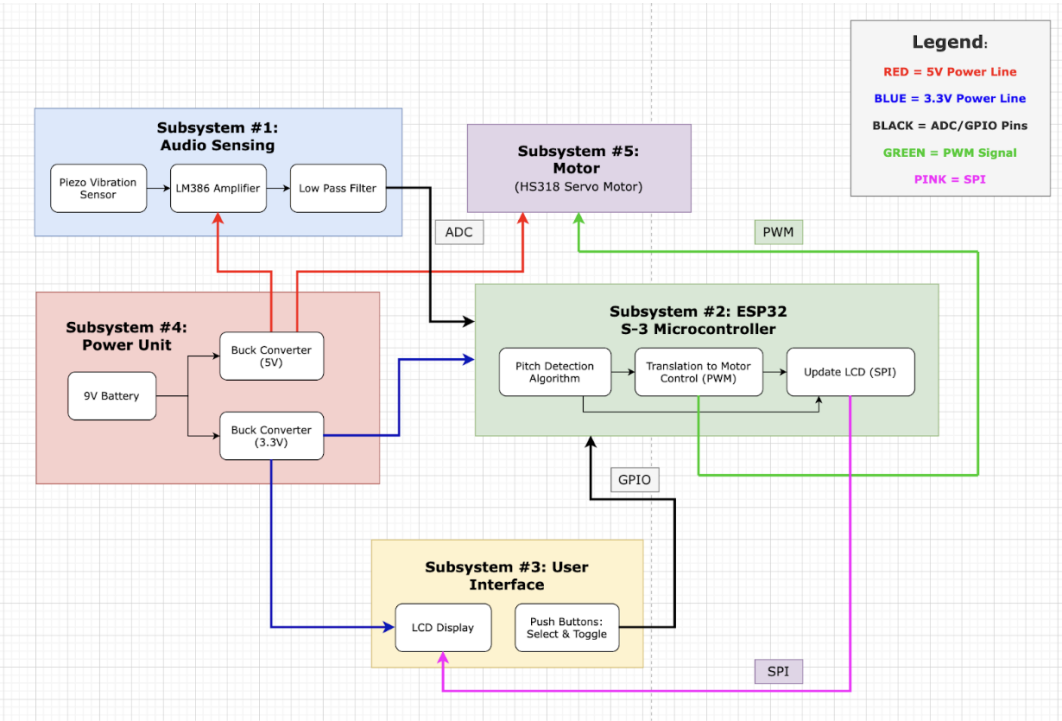

Figure 1: System Block Diagram showing all subsystems and their interconnections. Red = 5V Power Line, Blue = 3.3V Power Line, Black = ADC/GPIO Pins, Green = PWM Signal, Pink = SPI.

2.2 Design Details

2.2.1 Audio Sensing Subsystem

Design: For the actual amplification, we used the LM386 IC from Texas Instruments. It works well as a low-voltage audio power amplifier. By default, the IC has a gain of 20, but we designed it to have a gain of 50. This was done by using the capacitor and resistor values provided in the datasheet.

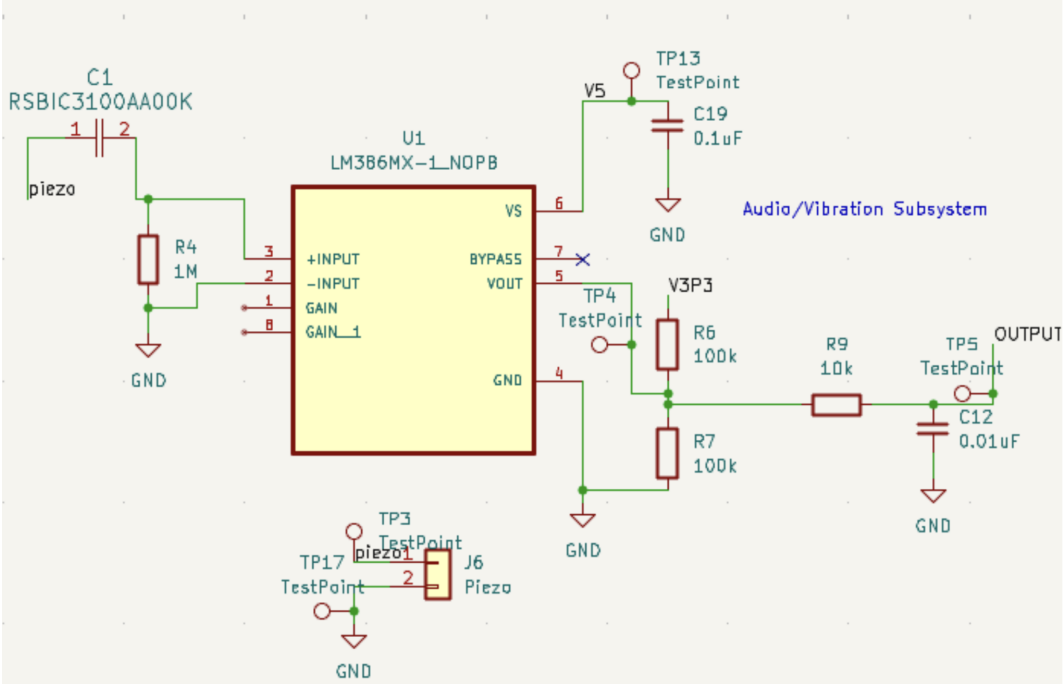

Figure 2: Audio/Vibration subsystem schematic showing piezo input, LM386 amplifier, biasing network, and low-pass filter.

From the output of the amplifier, we designed the PCB to use 2 resistors to bias the voltage at 1.65V, as the ESP32's ADC input ranges from 0 to 3.3V. This allows the circuit to take advantage of the full frequency resolution available from the ADC, as well as protect the ADC from voltage levels too high if amplified too much. We designed a passive low-pass filter using a 10kΩ resistor and 0.01μF capacitor, to prevent aliasing or unwanted high-frequency noise. The cutoff frequency was calculated based on these values shown in the equation below:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi(10\text{k})(0.01\mu\text{F})} \approx 1591 \text{ Hz} \tag{1}$$

The highest note we need to recognize (on the high E open string) is approximately 659 Hz. Since the cutoff frequency is 1591 Hz, this allows the open string frequency and its lower harmonics to pass through without getting attenuated. But it strikes a nice balance since

it also doesn't allow too much noise, while using a pretty common resistor value for the low-pass filter.

Design Justification: For the audio/vibration subsystem, we had to decide whether to use a microphone or a piezo sensor to pick up the pitch of the guitar notes. As our group was prototyping the dielectric microphone for our first breadboard demo, the biggest concern was that there was a lot of ambient noise, and it would require a lot of processing power to handle the filtering and FFT algorithm to accurately measure the pitch. The piezo sensor would be directly mounted on the guitar to pick up mechanical vibrations. This would be less susceptible to changes in environmental conditions or ambient noise, making the piezo a better choice for our audio sensing.

2.2.2 Control (ESP32) Subsystem

The control subsystem serves as the central processing unit of the Auto-Guitar Tuner, responsible for coordinating all other subsystems, including audio sampling, digital signal processing, motor control, display management, and user input handling. We selected the ESP32-S3-WROOM microcontroller as the core component due to its powerful dual-core architecture, extensive peripheral support, and integrated memory.

Hardware Architecture: The ESP32-S3 interfaces with all major subsystems through dedicated peripherals:

- **Audio Input:** GPIO pin configured as ADC1_CH0 samples the amplified piezo signal at 8192 Hz with 12-bit resolution
- **Motor Control:** GPIO pin generates 50 Hz PWM signal with variable duty cycle (5–10% corresponding to 1–2ms pulse width)
- **Display Interface:** Hardware SPI (VSPI) communicates with TFT LCD at 40 MHz clock speed
  - MOSI (GPIO23): Data output to display
  - SCK (GPIO18): SPI clock
  - CS (GPIO5): Chip select
  - DC (GPIO2): Data/command selection
  - RST (GPIO4): Display reset
  - BL (GPIO5): Backlight
- **User Buttons:** Two GPIO pins with internal pull-up resistors (active-low configuration)
  - Toggle Button (GPIO15): String/mode selection
  - Select Button (GPIO16): Confirmation/start tuning

- **USB Programming:** USB D+/D- connected to GPIO19/GPIO20 for firmware upload via micro-USB receptacle

Pitch Detection Algorithm: We implemented time-domain autocorrelation for pitch detection after evaluating multiple approaches, including FFT-based spectral analysis. The autocorrelation method proved superior for guitar tuning applications for several critical reasons:

Why Autocorrelation Over FFT:

1. **Harmonic Rejection:** Guitar strings produce complex harmonic content where the second harmonic (octave above) often has higher energy than the fundamental frequency. When using FFT-based magnitude spectrum analysis, the frequency bin with maximum energy frequently corresponded to the second harmonic rather than the fundamental. Autocorrelation, by contrast, naturally identifies the fundamental period because it measures the signal's self-similarity at different time lags.
2. **Computational Efficiency:** Direct time-domain autocorrelation on the ESP32-S3 proved significantly faster than FFT-based approaches:
   - Time-domain autocorrelation: ∼18–22ms for 2048 samples
   - FFT + inverse FFT + peak detection: ∼45–50ms for same sample size
3. **Frequency Resolution:** For guitar tuning requiring ±2 Hz accuracy across 73–330 Hz range, time-domain autocorrelation provides inherently better resolution in the low-frequency range where guitar fundamentals reside.
4. **Simplicity and Robustness:** The autocorrelation peak-finding algorithm is straightforward and less sensitive to noise than FFT magnitude thresholding.

Autocorrelation Implementation:

Our time-domain autocorrelation algorithm follows these steps:

1. **Windowing:** Apply Hann window to reduce edge effects:

$$w[n] = 0.5 \left(1 - \cos\left(\frac{2\pi n}{N}\right)\right) \tag{2}$$

   where $N = 2048$ samples

2. **Time-Domain Autocorrelation:** Compute for relevant lag range:

$$R_{xx}[k] = \sum_{n=0}^{N-k-1} x[n] \cdot x[n+k] \tag{3}$$

   We only compute lags corresponding to the guitar frequency range:
   - Min lag: $\lfloor f_s/330 \rfloor = 24$ samples (highest guitar note)

- Max lag: $\lfloor f_s/73 \rfloor = 112$ samples (lowest note we tune)

3. **Normalization:**
$$\rho_{xx}[k] = \frac{R_{xx}[k]}{R_{xx}[0]} \tag{4}$$

4. **Peak Detection with Zero-Crossing:** Start at lag $k = 24$, detect first zero-crossing where $\rho_{xx}[k]$ transitions from positive to negative, continue past zero-crossing to find first maximum where $\rho_{xx}[k] > 0.5$.

5. **Sub-sample Interpolation:** Apply parabolic interpolation around the detected peak:
$$\delta = \frac{\rho_{xx}[k_p + 1] - \rho_{xx}[k_p - 1]}{2(\rho_{xx}[k_p + 1] - 2\rho_{xx}[k_p] + \rho_{xx}[k_p - 1])} \tag{5}$$
$$k_{\text{refined}} = k_p + \delta \tag{6}$$

6. **Frequency Calculation:**
$$f_0 = \frac{f_s}{k_{\text{refined}}} \tag{7}$$
where $f_s = 8192$ Hz (sampling rate)

PID Controller Tuning:

The motor control PID gains were experimentally tuned:

- $K_p = 0.15$: Proportional gain provides immediate response to frequency error
- $K_i = 0.02$: Integral gain eliminates steady-state error
- $K_d = 0.05$: Derivative gain dampens oscillations

Safety Interlocks:

Multiple safeguards prevent damage to the guitar or device:

1. **Voicing Detection:** Only enable motor when autocorrelation peak exceeds threshold (0.6)
2. **Frequency Bounds Check:** Reject detected frequencies outside 60–350 Hz range
3. **Rotation Limits:** Enforce maximum single-step rotation of $\pm 5^\circ$
4. **Timeout Protection:** Abort tuning after 30 seconds if target not reached
5. **Stall Detection:** Stop if no progress for 5 seconds

Memory Management:

The ESP32-S3 memory allocation:

- Audio sample buffers: 8KB (2048 samples $\times$ 4 bytes per float)
- Autocorrelation working buffer: 4KB

- Display framebuffer: 153.6KB (320×240 pixels × 2 bytes RGB565)
- RTOS stack: 32KB total across 5 tasks
- Remaining ∼310KB SRAM available for code execution

2.2.3 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. Two tactile buttons (Select and Toggle) 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: 2x 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 1: UI Finite State Machine states and transitions

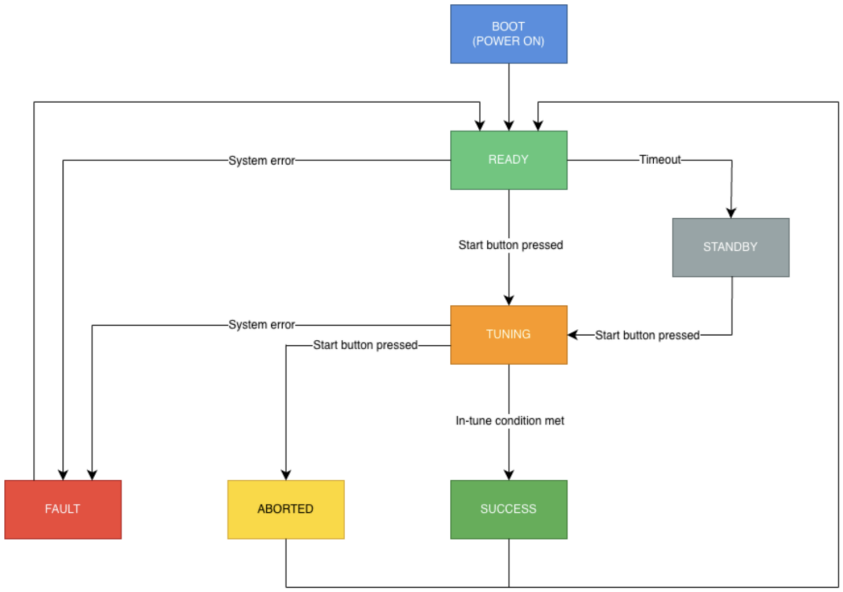

Figure 3: Diagram depicting the logic flow of the UI Finite State Machine (FSM)

Button Functions Through a combination of short and long button presses, the toggle and select buttons allow a user to navigate through the full UI. Having long and short presses allows full control without requiring a lot of additional buttons.

With each click of the select button, the six standard guitar strings are cycled through in this sequence (E2 → A2 → D3 → G3 → B3 → E4), returning to E2 after E4. Users can move backward through the string sequence by long-pressing this button, which reverses the cycle direction.

Standard tuning, Eb Standard, drop D, and open G are the tuning modes that can be selected by short pushes on the button. A long press opens a separate mode selection menu.

2.2.4 Power Subsystem

Design: The power management subsystem is responsible for controlling the power delivery to all other subsystems of the PCB. We need stable 3.3V and 5V rails, so we used AP62150WU-7 Buck Switching Regulators. One of the big reasons we chose to use this specific buck switching regulator is that it has up to 1.5A of continuous output current. Although we're planning to use the same IC for both power rails, we are using various resistor values to adjust the voltage drops and produce the stable 5V and 3.3V rails for our subsystems.

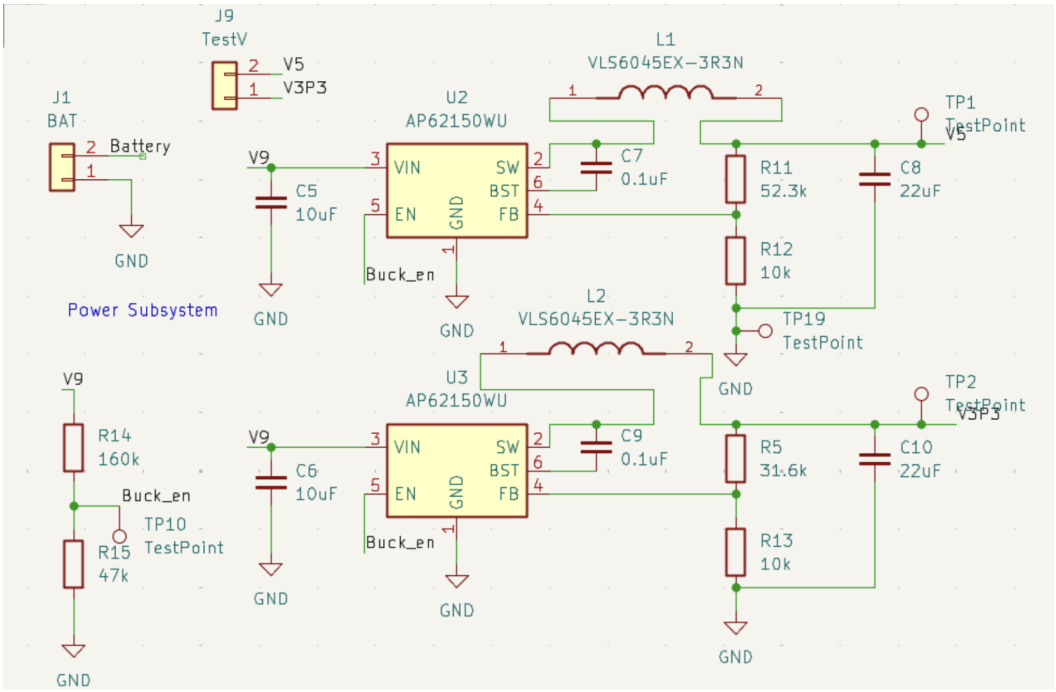

Figure 4: Power subsystem schematic showing dual AP62150WU buck converters for 5V and 3.3V rails.

We also used two VLS6045EX-3R3N 3.3μH inductors, since they can support up to 4.95A. In our worst-case scenario when the motor would stall, the motor would draw up to 800mA. So we chose these large SMD inductors since they support a much higher current rating.

Design Justification: The decision for our group was between an LDO (Low-Dropout Voltage Regulator) and a buck converter. The power dissipated as heat for an LDO can be calculated as follows:

$$P_{loss} = (V_i - V_o) \times I_{out} \tag{8}$$

For example, looking at our HS-318 servo motor alone, it can draw 800 mA if the motor is stalled. Using the above equation, this results in:

$$P_{loss} = (9\text{V} - 5\text{V}) \times 800\text{mA} = 3.2\text{ W} \tag{9}$$

This is a significant amount of power lost to heat, which could also affect LDO device behavior. Since our 5V power rail drives many different subsystems, we decided that buck converters (AP62150) would be better fit for our system since they can supply more current, and have higher efficiency (∼90% efficiency at 1.0 A).

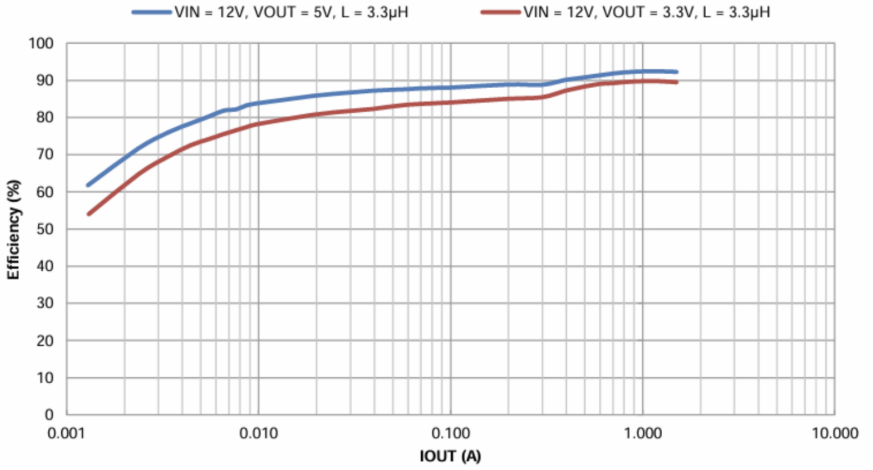

Figure 5: Efficiency vs Output Current for the AP62150. Even at 12V input, the buck converter achieves ∼90% efficiency at 1.0A load.

2.2.5 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). The motor has a potentiometer that 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. This correction is highly useful for our system since we need fine adjustments.

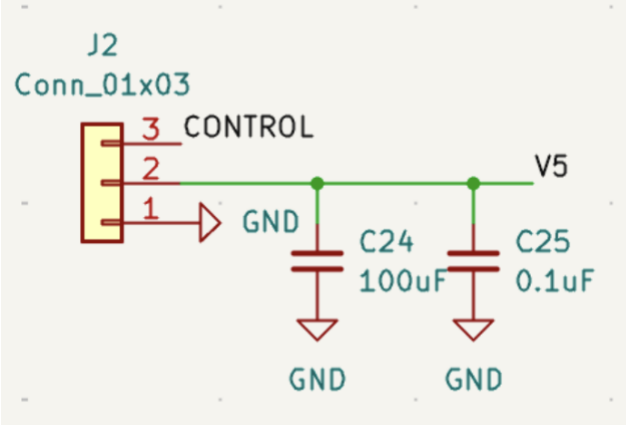

Figure 6: Motor subsystem schematic

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.

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

Table 2: HS318 servo motor specifications

Design Justification: In the early stages of our design, our group was torn on whether to utilize a stepper motor or a servo motor. When looking at the trade-offs, we noticed two issues with a stepper motor. First, stepper motors have an open loop, assuming all their movements are correct. This can be problematic, especially when strings are tight and the motor skips steps. Meanwhile, a servo motor has a closed-loop, correcting errors in its adjustments. Secondly, stepper motors lose torque at lower speeds. Since our tuning peg adjustments are small angular adjustments, it would be far more useful to have a servo motor since it maintains torque across its full speed range.

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.

$$\tau_{\text{post}} = T \times r_{\text{post}} = 80\text{N} \times 0.003\text{m} = 0.24\text{Nm} \tag{10}$$

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:

$$\tau_{\text{motor}} = \frac{\tau_{\text{post}}}{\text{ratio} \times \eta} = \frac{0.24\text{Nm}}{14 \times 0.7} = 0.0245\text{Nm} = 0.25\text{kg} \cdot \text{cm} \tag{11}$$

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).

3 Verifications

3.1 Audio Verification

To verify the amplification of the audio subsystem, our main verification experiment was to compare the input piezo signal and the output signal of the amplifier. Based on the first audio sensing requirement in appendix A, we need the input signal of the piezo to be greater than 10 mV to ensure that the signal is reliable and not just noise. As seen in Figure 7, the yellow piezo channel has an amplitude of 84 mV.

Second, we want to see that there has been a gain of 50 applied to the piezo signal. Looking at the raw output (green channel in Figure 7), we can see that the amplitude of the post-amplified signal is 2.2V. But looking at figure 2, we cut the voltage of the amplified in signal in half using the voltage divider circuit. So this means the pre-biased amplifier output sits around 4.4V. We can calculate the gain below:

$$\frac{2.2V}{84mV} =\sim 52. \tag{12}$$

This is approximately a gain of 50, which is what we were expecting.

Figure 7: Pre and Post amplification piezo signal on oscilloscope. Channel 1 (yellow) shows raw piezo signal (∼84mV), Channel 2 (green) shows amplified output (∼2.2V p-p).

3.2 Control Verification

When testing the overall functionality of our project, we were able to verify our control subsystem requirements. First, the motor was able to accurately detect each string's current pitch and control the motor to rotate until the string was right in tune. This worked for all 6 strings. Second, the control would continue sensing and control as the motor was working. We found that we had an average of 22ms of auto-correlation latency between piezo input data and having our frequency result availible. This tight feedback loop allowed our tuner to continue adjusting the motor without requiring the user to pluck each string over and over. We also had a warning indication on the UI where the ESP32 would recognize when the servo would reach its limit, telling the user to remove the motor before controlling the motor to reset into a neutral position.

3.3 UI Verification

We were able to verify a couple of our UI requirements from appendix A by controlling the device operation with our buttons. When testing our system, we got our auto-guitar to cycle through all E2, A2, D3, G3, B3, and E4 strings correctly. We also tested various tuning standards (Eb Standard, Drop D) and our UI adapted to target those different frequencies. We also saw that the tuning bar indicator would continue shifting towards the center, indicating that the display continued to refresh even as the motor was making fine adjustments.

3.4 Power Subsystem Verification

DC Output Voltage Testing:

To test our buck converters, we applied a controlled 9V input using a power supply, ensuring a stable and precise voltage feed to the system. We then utilized test points on the PCB to measure the output voltages of both buck converters. Using a digital multimeter (Keysight 34461A), we confirmed that the outputs were approximately 5V and 3.3V, respectively.

Figure 8: 3.3V Buck output reading via DMM showing 3.353V output.

Figure 9: 5V Buck output reading via DMM showing 4.995V output.

3.5 Motor Subsystem Verification

To verify the motor's basic movements, we powered the motor using the 5V output from our buck converter, ensuring a stable supply. We applied a square wave pulse generated by a function generator directly to the PWM test point on our PCB at a frequency of 50 Hz. By varying the duty cycle of this PWM signal, we were able to control the angle position of the servo motor, verifying its responsiveness. This test was to ensure that the motor could accurately be controlled, though we didn't get to measure the PWM signals sent by the ESP32 using an oscilloscope.

We also monitored the 5V power rail close to the motor through a test point to observe the voltage stability while the motor was running under different duty cycle conditions. The 5V rail remained stable during multiple duty cycle levels. We were also able to verify the last motor requirement in appendix A by seeing the motor stopping all motion once the LCD indicated a note was in tune. The motor wouldn't start moving again until the user connected the motor to the next string.

4 Costs

4.1 Parts Cost

Component	Part Number	Quantity	Price
ESP32 Microcontroller	ESP32-S3-WROOM-1-N16R8	1	$6.56
ESP32-S3 DevBoard	ESP32-S3-DevKitC-1-N32R16V	1	$17.00
Mic Amplifier	MAX9814	3	$9.99
35mm Piezo Disc	B0B2QS8VK5	10	$9.99
2.4inch LCD Display	B08H24H7KX (Waveshare)	1	$18.99
2.2inch LCD breakout	B01CZL6QIQ (HiLetgo)	1	$14.49
Buck Switching Regulator	AP62150WU-7	2	$0.66
USB micro B Receptacle	10118192-0001LF	1	$0.43
Amplifier IC	LM386MX-1	1	$0.85
Tactile Switch	KMR232NGULC LFS	2	$1.40
100μF Capacitor	35ZLH100MEFC6.3X11	1	$0.32
3.3μH Inductor	VLS6045EX-3R3N	2	$0.68
31.6kΩ Resistor	ERA-6AEB3162V	1	$0.10
52.3kΩ Resistor	ERA-6AEB5232V	1	$0.10
Acoustic Guitar	First Act	1	$35.00
0.1μF Film Capacitor	KEMET	1	$0.62
		Total Parts Cost:	**$81.56**

4.2 Labor Costs

According to the Grainger Engineering career services salary and hiring data portal, the average graduate makes $96,270 a year. Assuming approximately 2080 hours per year, 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:

$$\$46.28 \times 16 \text{ weeks} \times 15 \text{ hours} = \$11,107.20 \tag{13}$$

Since our group has 3 team members:

$$\$11,107.20 \times 3 \text{ members} = \textbf{\$33,321.60} \tag{14}$$

4.3 Total Costs

$$\$33,321.60 \text{ (labor)} + \$81.56 \text{ (parts)} = \textbf{\$33,403.16} \tag{15}$$

5 Conclusions

5.1 Accomplishments

Overall, we satisfied all of our high-level requirements. All the subsystems on our PCB were functional, and the system was able to tune our guitar exceptionally well. Specifically, the latency between plucking a guitar string and the frequency showing on the LCD was much lower than expected. Since extracting the correct frequency from a vibration signal is computationally complex, we were expecting at least a few seconds before the user would be able to see how the string is tuned. But as we tested our PCB, we found that the note would be recognized instantaneously. This helped with motor control, as the tighter feedback loop would allow our ESP to recognize the new adjusted frequency before the string stopped vibrating. This makes it easier for users, as they won't have to re-strum each string after every fine motor adjustment.

5.2 Uncertainties

One of the main uncertainties for this project was the mechanical design of the casing. Throughout the course, as we worked with the machine shop to design our PCB casing, we didn't account for certain wire spacings in our designs, which made our final testing difficult. Our casing was crammed, and actually caused a short between a power rail and our buck converters when we tried closing the lid.

Another major uncertainty was our power subsystem. There were multiple cases where our bucks became damaged due to having no overcurrent protection or isolated grounds issues. Adding a fuse at the battery input would've been a good idea to prevent some damage.

5.3 Ethical Considerations

Guitar strings are designed to operate under a specific range of tension. If a string is tightened too much, the string snaps and can recoil—potentially striking a user and causing injury to the face or hands. It could also damage the guitar itself.

Throughout the design and implementation of our auto guitar tuner, we were guided by the foundational principle of IEEE Code of Ethics, Section 1.1: "to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose promptly factors that might endanger the public or the environment." We designed our system using a servo

motor with a 210° limit. This directly prevents continuous rotation of the tuning peg. But precise motor control was also key to achieving accurate tuning while maintaining safety. The ESP32 controlled the motor to make small angular adjustments so that the tuning pegs would never be overrotated, while also allowing more accurate tuning verification.

5.4 Future Improvements

While our project was highly successful, there are many areas where we can improve our current design. A primary goal would be enhancing performance and compactness. We could replace the current 210-degree servo motor with a continuous rotation servo motor. This change would eliminate the mechanical rotational limit, allowing for much faster tuning peg rotations, which would cut down tuning time considerably. Concurrently, we should focus on reducing the size of the PCB and enclosure by utilizing smaller components and footprints. Using other push buttons and connectors would also make the wires easier to fit into our mechanical casing.

The user experience could be significantly improved by adding a wireless or app-based interface. This would allow users to define, store, and quickly load custom tuning profiles (like Drop D or Open G, or Other), increasing the utility. Finally, we could design and implement different tuning sockets or an adaptable mechanism to support the varying peg and tuner head sizes found across multiple guitar types, including classical and 12-string guitars.

References

[1] Texas Instruments, "LM386 Low Voltage Audio Power Amplifier," Datasheet, 2017.

[2] Espressif Systems, "ESP32-S3 Technical Reference Manual," 2022.

[3] Diodes Incorporated, "AP62150 2A, 1.2MHz Synchronous Buck Converter," Datasheet, 2021.

[4] Waveshare, "2.4inch LCD Module User Manual," 2020.

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

A Appendix: Requirements & Verification Table

Requirement	Verification	Done?
Power Subsystem		
Stable 3.3V line with tolerance of 5% (3.135–3.465V)	Measure 3.3V line using oscilloscope	Yes
Stable 5V line with tolerance of 5% (4.75–5.25V)	Measure 5V line using oscilloscope	Yes
Maintain ±2–3% of nominal output voltage during fast load step	Use oscilloscope to verify voltage remains within 2–3% during fast load step	Yes
Continuous, stable 5V line within ±5% when servo motor stalls	Monitor 5V rail with oscilloscope and intentionally stall motor	No
Control Subsystem		
Pitch to Rotation: ESP32 shall detect fundamental frequency and compute required rotation	Use function generator to replicate strumming; verify PWM signal with oscilloscope	Yes
LCD Status & UX: ESP32 shall render real-time status without disrupting sensing and control	Run tuning cycles and verify display shows correct information	Yes
Safety Interlocks: Block motor commands when unvoiced/noisy frame or stall occurs	Trigger each safety condition and verify PWM disable and LCD fault display	Yes
UI Subsystem		
Button press must be registered within 200ms	Measure response with logic analyzer	No
LCD must display the updated string selection	Cycle through strings and verify display	Yes
Mode switching must change target frequencies	Toggle modes and verify displayed reference	Yes
Display must refresh without flickering	Observe continuous tuning display during operation	Yes
Audio Subsystem		
Piezo input signal $\geq$ 10mV	Measure voltage across piezo disc wires	Yes
Output signal amplitude must be 50× piezo input amplitude	Measure input and output amplitudes with oscilloscope	Yes
Current drawn from power supply pin $<$ 4mA	Insert multimeter between 5V rail and V5 pin	No

Requirement	Verification	Done?
Motor Subsystem		
Angular Resolution: Servo motor accuracy of 1° (≈5–7 cents)	Command movements to random angles and measure with encoder	No
PWM Command: Correct PWM based on frequency difference	Measure PWM signal with oscilloscope; verify pulse width	No
Motor Position: Servo rotates to desired position (±1°)	Verify string is in tune after rotation; no further rotations needed	Yes