**Q-Tune: Release Candidate**

Yenessa Avalos-Maldonado, Timothy Chiles, William Trigg, Giovanni Cornejo, and Dakota Cappel

Github Link: <https://github.com/avalosy8/qtune>

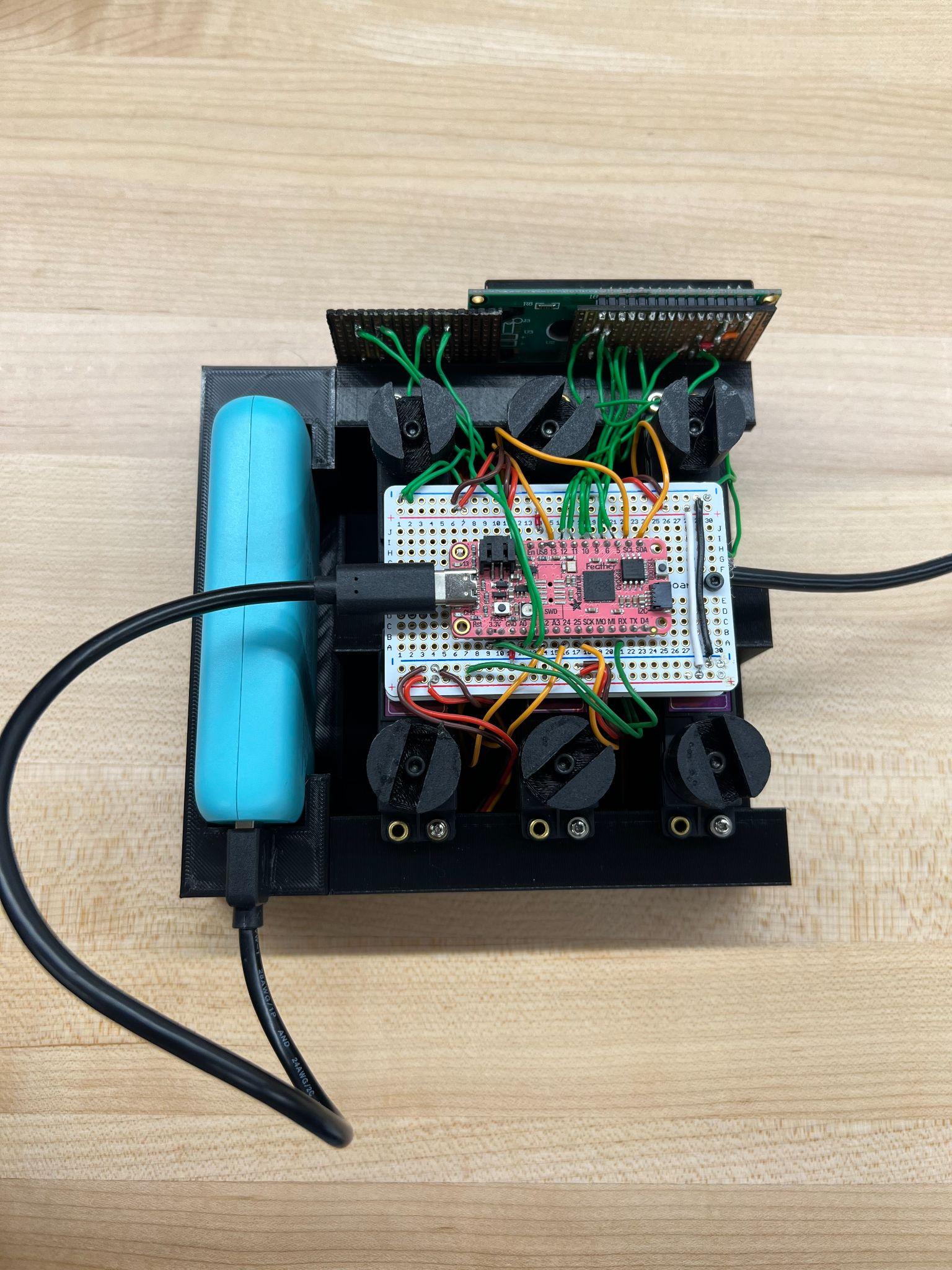


Figure 1: Top down view of the completed Release Candidate (left). Side view of the device clamped onto the guitar, with the motor tuning attachment properly fitting onto the guitar’s tuning pegs (right).

**External Interface**

Our automatic guitar tuner’s external interface consists of a piezo vibration sensor, which connects as an input to the op-amp comparator circuit, and a servo motor that is driven by an output pin from the Adafruit RP 2040. The output of the comparator circuit is the input to the RX pin of the RP 2040, which is polled to start the frequency calculations. The external interface connects to the persistent state by the use of state variables and constants in memory, which are used to store the data obtained from the vibration sensor and comparator circuit. Additionally, all the hardware components have been condensed to fit within a 3D case as shown in Figure 1. The circuitry components have been moved and soldered onto two perfboards, one containing the Adafruit board and another containing the charge amplifier, filters, and comparator circuit. The entire device is attached to the guitar’s head using two clamps.

**Persistent State**

The guitar tuner stores its data via state variables and constants in memory. The constants are stored in onboard flash memory and are necessary for our internal tuning systems to run properly. The state variables are stored in RAM on the Adafruit board and are updated in response to input from the frequency-detecting components of our external interface. The persistent state connects to the external interface of our build through pins on the RP2040 microcontroller. The RX pin from the external interface is polled to start the frequency calculations. The persistent state stores this information to determine if the motors from our external interface need to turn the tuning pegs on the guitar. The persistent state connects to the interface system by also utilizing stored variables in order to process the necessary data to allow each component of our internal system to properly process data.

**Internal Systems**

Our three main internal systems are our frequency-detection system to determine the tuning of the guitar, our motor-turning system to alter the tuning of the guitar, and our UI system to allow users to interface with our guitar tuner. More advanced versions of the data processing involved in our three systems have been implemented. At present, the frequency-detection system can calculate the frequency of a guitar string via the external piezo sensor circuitry to some degree of accuracy. The motor-turning system can turn the motors in response to the frequency calculated with some degree of precision. The UI system allows users to select the string they want to tune and reports the tuner’s progress as a percentage. The internal system connects to the persistent state by the use of state variables and constants in memory to process the gathered data from each component.

**Communication**

The user will be interfacing with the LCD screen and two push buttons as a means of interacting with the tuner. The push buttons will be accessible to the user near the LCD screen, on the left and right side. These buttons accurately detect a selection made by the user and displays the correct string that was selected and its targeted frequency. Thus, the user will be able to intuitively select the desired string to tune. After making a selection,the user will just have to view the LCD screen to be notified about whether the guitar string needs to be strummed again based on tuning percentage. When the measured frequency isn’t close to the target frequency within an acceptable range then the motors move the tuning peg as expected and the user is prompted to strum the guitar string again.

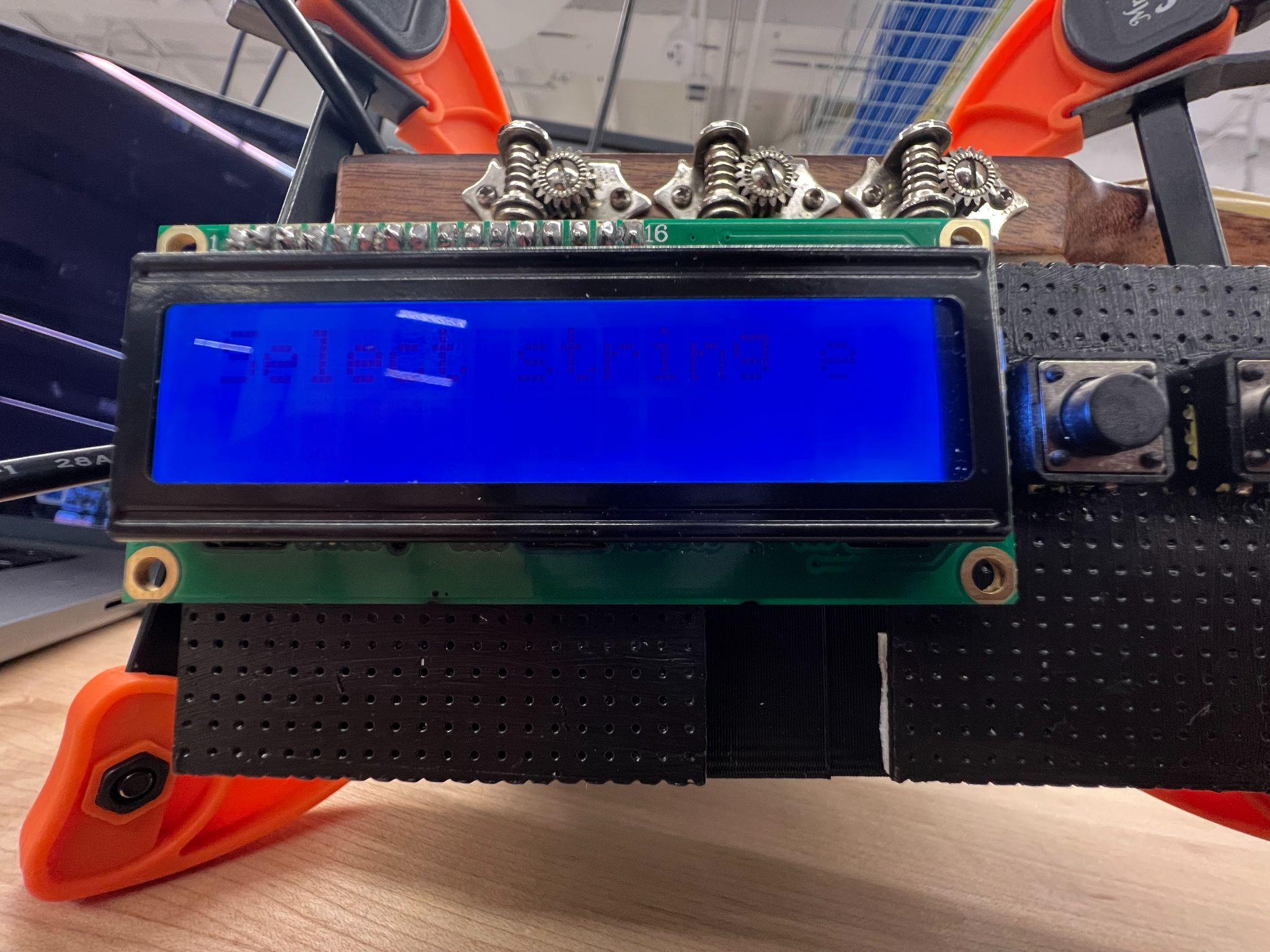


Figure 2: User is first prompted to select a string.



Figure 3: After the string is selected, its target frequency is displayed.



Figure 4: Percentage of how tuned the selected string is. This is displayed after the frequency is measured.

**Integrity & Resilience**

When running the program normally, the device is able to run through menu options, display the desired frequency, display the measured frequency and tuning progress percentage, and turn the motor without any crashing or system failures. There are multiple filters in-place, both in hardware and software, to ensure that the system does not pick up unwanted frequencies and produce undesired results such as improper tuning. When the second harmonic of the strings appears more strongly at the input than its fundamental frequency, our code has the infrastructure in place to account for the doubled frequencies. All random instances of crashes have been debugged and dealt with as well, creating a smooth and consistent user experience. Furthermore, the responsiveness of the buttons is immediate, without bouncing or false input detections, and accurately selects the string to be tuned. Once the motors move, the LCD screen immediately notifies the user whether the target frequency was reached.

The measured frequencies remain consistent as we have the signal fed through a charge amplifier and a Schmitt Trigger. The charge amplifier makes the resultant sine wave reflect the vibration of the guitar more accurately because the piezo produces a proportional charge to mechanical stress, not a proportional voltage. The charge amplifier also nulls the effects the piezo sensor and its cable’s internal capacitance could potentially have on our signal. The charge amplifier eliminates noise by only allowing frequencies within the guitar’s expected operating range (~60-360Hz) to continue through the circuitry. The Schmitt Trigger can produce a clean and accurate square wave representation from potentially noisy sinusoidal signals via its two threshold voltages.