**Q-Tune: Alpha Build**

Yenessa Avalos-Maldonado, Timothy Chiles, and William Trigg

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



Image 1: Completed Alpha Build

**Usability**

**Interface**

The LCD screen accurately displays the processed readings from the vibration sensor. There are two debounced push buttons that allow the user to select a string to tune. As the program is running, the frequency readings are displayed and notifies the user when the frequency is either too high or too low, which indicates that the tuning pegs will be adjusted and that the user needs to strum the guitar strings again to re-measure the frequency values.

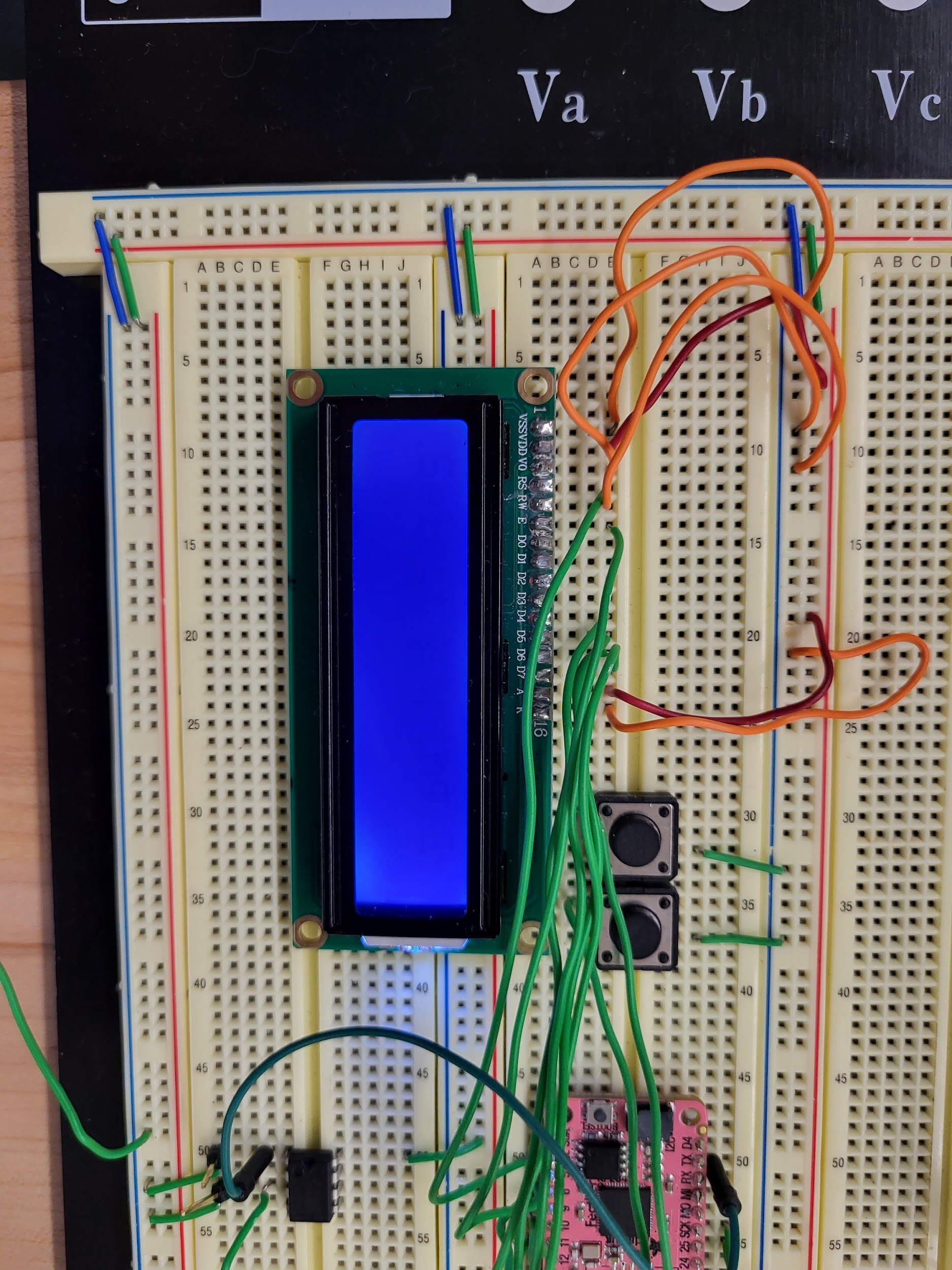


Image 2: UI for the automatic guitar tuner including the LCD screen and push buttons

**Navigation**

When the measured frequency isn’t close to the target frequency, within an acceptable range, then the motors move the tuning peg as expected. Additionally, the push buttons accurately detect a selection made by the user and display the correct string that was selected and the targeted frequency. The user is then notified via the LCD screen whenever they need to strum the strings again to retune the guitar string.

**Perception**

The push buttons will be accessible to the user near the LCD screen, on the left and right side. The users will be able to intuitively select the string they wish to tune by pressing these buttons. After the motors move, the user will just have to view the LCD screen to be notified about whether the guitar string needs to be strummed again.

**Responsiveness**

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. To reduce idle time length and conflicts between tasks, we use asynchronous functions for moving the motors and measuring/displaying the current frequency values on the LCD screen.

**Build Quality**

**Robustness**

When running the program normally, the device is able to run through menu options, display the desired frequency, display current frequency, and turn the motor without any crashing or system failures.

**Consistency**

At the current state, we are in the process of switching to a larger motor in order to ensure that we have the proper torque needed to turn the pegs of the guitar. This has caused some unexpected inconsistency when running the entire program, which can be expected when implementing a new component. Our design will function properly up until attempting to turn the motor, where the program may act unexpectedly and not turn the motor. Our design with the old motor worked consistently and we expect the same once we figure out the specific issues with the changes of our motor and its integration into our system. We plan to try different ways to power the motor which we suspect is the main cause of this issue. We also will evaluate the code and see if there are any necessary changes that need to occur there.

**Aesthetic Rigor**

The LCD screen properly and consistently produces the correct tuning options and frequencies corresponding to that tuning. The menu is simplistic and intuitive so that any user can pick up the device and use it. Current hardware will be condensed and hidden into a single 3D-printed device to make for a more aesthetic design.

**Vertical Features**

**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 RP 2040. The output of the comparator circuit is the input to the RX pin of the RP 2040, which is used to trigger an interrupt and 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.



Image 3: Piezo sensor that detects frequency of guitar strings



Image 4: Servo motor with 3D-printed tuning attachment used to tune the guitar

**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 configured as an interrupt pin, which starts 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. Preliminary 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 whether the string is tuned too high, too low, or in tune. 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.