Automatic Guitar Tuner

Alex Shin and Jinwoo Kim

Introduction/Project Description

The goal of this project, as described by the name, was to create an embedded system that automatically tunes any guitar (for this project, we used an acoustic guitar). Our system employs two sound sensors to capture the audio signals generated by plucking the guitar strings. We also have a button along with six LEDs for the user to click through and select the string that they want to tune. Upon selecting a string to tune, the user can attach the motor to the guitar peg of that string, pluck the string, and the motor will automatically rotate to tune the guitar. When the string is in tune, an indicator LED will turn green to indicate that the selected guitar string is in tune (will be off when the string is out-of-tune). By automating the tuning process, our project seeks to offer guitarists a convenient and efficient solution for achieving optimal tuning accuracy without the need for manual adjustment, especially for enthusiasts without musically trained ears.

Materials

The following list provides the supplies necessary for completing this project

- Arduino Uno Microcontroller
- RGB Led
- 6 LEDs
- Button (we used a rotary encoder push switch but any button will do)
- Sound Sensors
    - KY-037
    - GY MAX4466
- Geared DC Motor
    - Torque @ Max. Efficiency (g-cm) = 850
    - 30:1 Gear Ratio
- 12V DC Power Supply
- 7 Resistors (330Ω)
- Tuning Peg Attachment/Gripper (3D printed)
- Jumper Wires

Design Process

Note Selection: For selecting the note, we originally had a rotary encoder and an I2C display, where the user would be able to turn the rotary encoder to select a particular string they want to tune which would be displayed on the I2C display. However, when combining this logic with our Fast-Fourier Transform (FFT) for frequency detection, it posed an issue where we did not have enough memory for the rest of our program, as we were using around 85% of the dynamic

memory (RAM) on our Arduino Uno. (ran into a lot of stack overflow errors and also seemed as if we had insufficient memory for interrupts - see Problems Encountered). Given this, we initially attempted to use smaller variable types where it was possible, but ultimately we decided to change the hardware. We opted for instead using 6 leds, which the user could select through using a button, to select the note they want to tune for. While we did not end up using the I2C display and rotary encoder logic in our final implementation, we included it in final directory ("Component_Unit_Tests/TestNoteSelection/TestNoteSelection.ino") as we believe it is still very interesting and someone else would gain inspiration from it.

Tuning Motor + Attachment: The design process for the tuning motor and attachment was very straightforward (although we did encounter a lot of different problems). The general idea was to use a stepper motor and a motor driver to precisely turn the motor, which would turn the pegs and tune the guitar string based on our audio signal processing. Furthermore, we designed our own guitar peg gripper, 3D printed it, and attached it to our motor (see Figures 1 below). Later into our engineering process, we encountered an issue with the stepper motor not having enough torque to turn the guitar pegs, and so we shifted to a motor with more torque (see Problems Encountered).

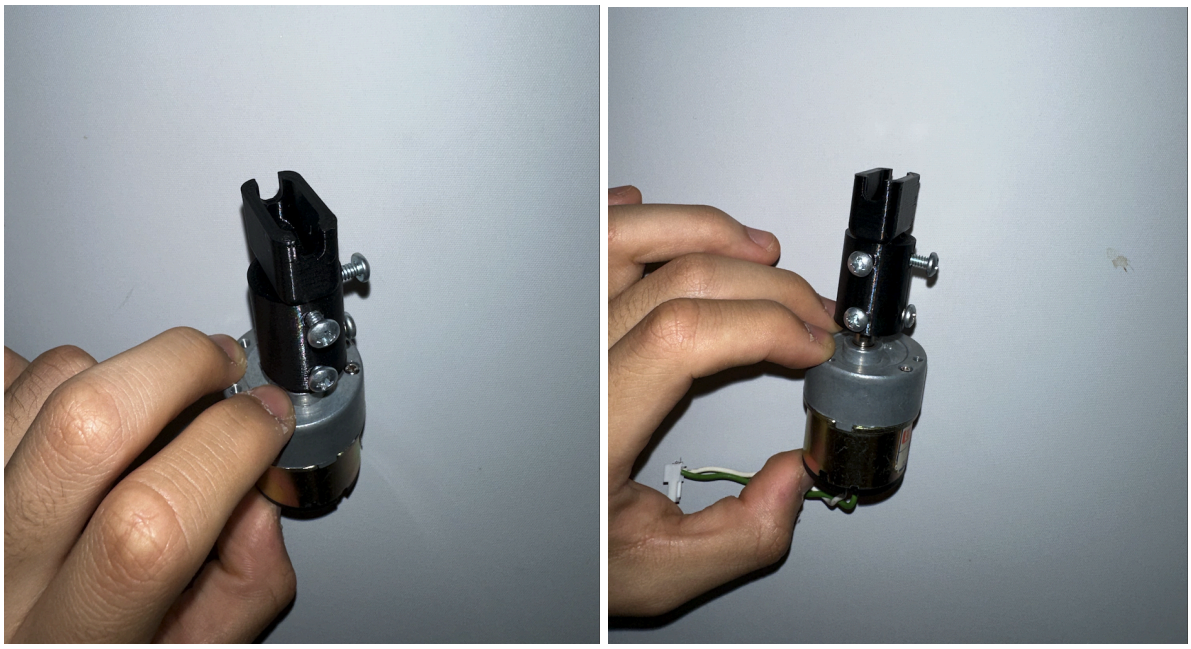


Figure 1: Guitar Peg Attachment mounted on motor

Audio Signal Detection:
We used a KY-037 High Sensitivity Sound module to detect the presence of sound, and a GY MAX4466 Sound Module for a precise analog reading of the sound. By sampling the analog signal and applying a Fast Fourier Transform, we were able to retrieve the frequency of the note being played by the guitar. However, the strings of our guitar did not produce a pure tone, and the sound module displayed some bias in the results. In order to account for this error, we played

each string of our tuned guitar and sampled the frequencies we got from our sound sensor, taking the average frequency of ten samples for each string. Most of the frequency recordings we recorded were very similar to the expected values. As an example, for the D-string we received an average frequency of 148.85Hz (the actual frequency of D3 is 146.83Hz). Thus, when tuning the D-string, we set the target frequency to our sampled value.

Significant Problems Encountered

Insufficient Memory: 1) As aforementioned, with our program taking up so much dynamic memory (RAM) on the Arduino Uno, we ran into issues when attempting to combine our rotary encoder and I2C display logic with the rest of our project code. Although the note selection display worked independently, it posed memory issues and so we went in a different direction. Maintaining the functionality of the display we elected to use a button and 6 leds, which used less memory to control note selection. 2) Because of the memory limitations of the Arduino Uno, we could only use a maximum sampling size of 128, significantly decreasing the precision of our sound module. Although our project still does a great job tuning, the frequency detection of the sound module could be a lot more precise if we had more memory to use a great sampling size for our FFT.

Torque Issues: When originally mounting our stepper motor (w/ the peg attachment), and writing code to turn the stepper motor, the stepper motor did not have enough torque to actually turn the guitar pegs, and would simply skip steps. We also tried this with a servo motor, which also did not have enough torque to turn the guitar pegs. After several attempts and iterations to maximize the torque outputs from both the servo motors and stepper motors, we opted to use a different motor with more torque. The highest torque motor we were able to get our hands on was a 12V geared DC motor with Torque @ Max. Efficiency of 850 (g-cm) with a 30:1 gear ratio (2550 g-cm max torque), which had about 3x the torque of the stepper motor. The issue however, was we need to use a DC motor for very precise tuning. Therefore, in order to accomplish this, we used a motor driver with an H-bridge (See Figure 2) so that we would be able to control the direction of the motor (CW, CCW, STOP), the speed of the motor, and the rotations of the motor (essentially creating a “step” with pulse-width-modulation). Referencing the pinout of the motor driver (See Figure 3): When In1 is high and In2 is low, the motor turns counterclockwise, when In1 is low and In2 is high, the motor turns clockwise, and when both are low the motor stops (all when PWM is high). Furthermore, we could no longer use our 9V battery if we wanted to optimize the torque, so we used a 12V DC Power Supply to fit the voltage specifications of the motor.

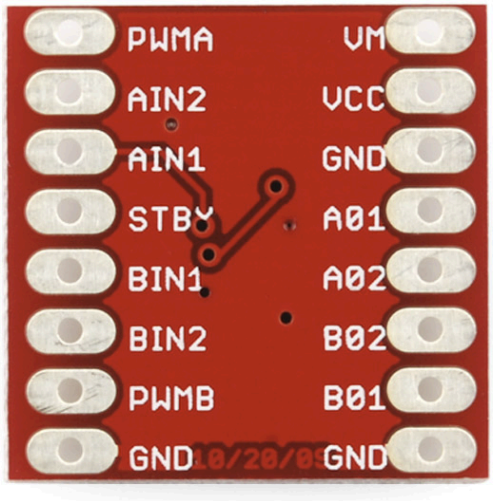


Figure 2: TB6612FNG Motor Driver

In1	In2	PWM	Out1	Out2	Mode
H	H	H/L	L	L	Short brake
L	H	H	L	H	CCW
L	H	L	L	L	Short brake
H	L	H	H	L	CW
H	L	L	L	L	Short brake
L	L	H	OFF	OFF	Stop

Figure 3: TB6612FNG Motor Driver Truth Table

Audio Signal: It required quite a bit of time to find and tune the correct sensitivity settings of the sound modules (the potentiometers), but with a lot of testing we were able to find the correct sensitivity settings (needed to be sensitive enough to pick up the audio from the guitar - test code is included in 'Component_Unit_Tests/SoundSensorTest/SoundSensorTest.ino'). Furthermore as mentioned before, because of the memory limitations of our Arduino microcontroller our FFT is not as good, decreasing the precision of our frequency detection. Thus, we have a "deadzone range" such that when a string is within this range of the target frequency, it is considered in tune. Moreover, we also have a listening range for each string, so very erroneous readings don't cause our motor to spin in an unexpected behavior.

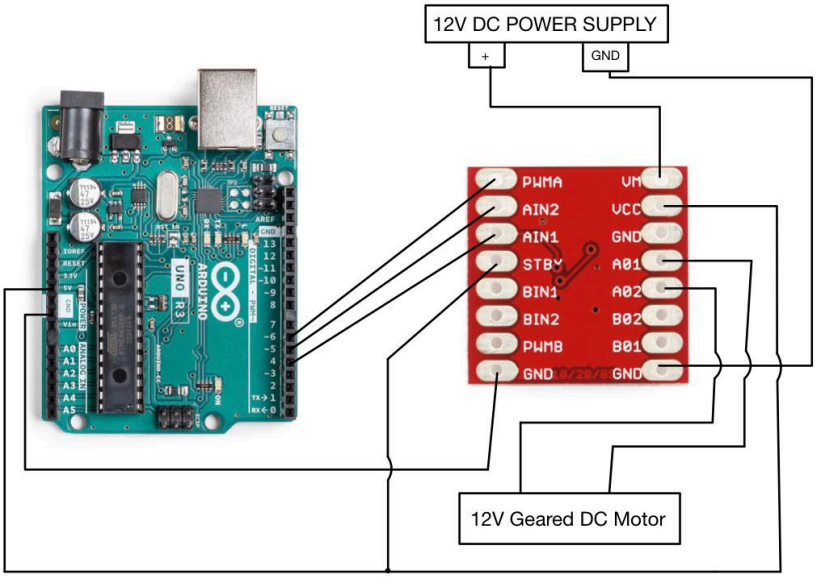
Limitations

The biggest limitation of our project is our frequency detection, which comes down to the memory limitations of our Arduino Uno. While our project still does a good job of tuning the guitar strings, more accurate frequency detection would significantly increase the precision of

our project as we would be able to use a much greater sample size and sampling frequency for our FFT. If we were to do this project again, we would use a microcontroller with more memory as well as more precise sound sensors.

Schematic

Figure 4: Motor + Motor Driver Schematic



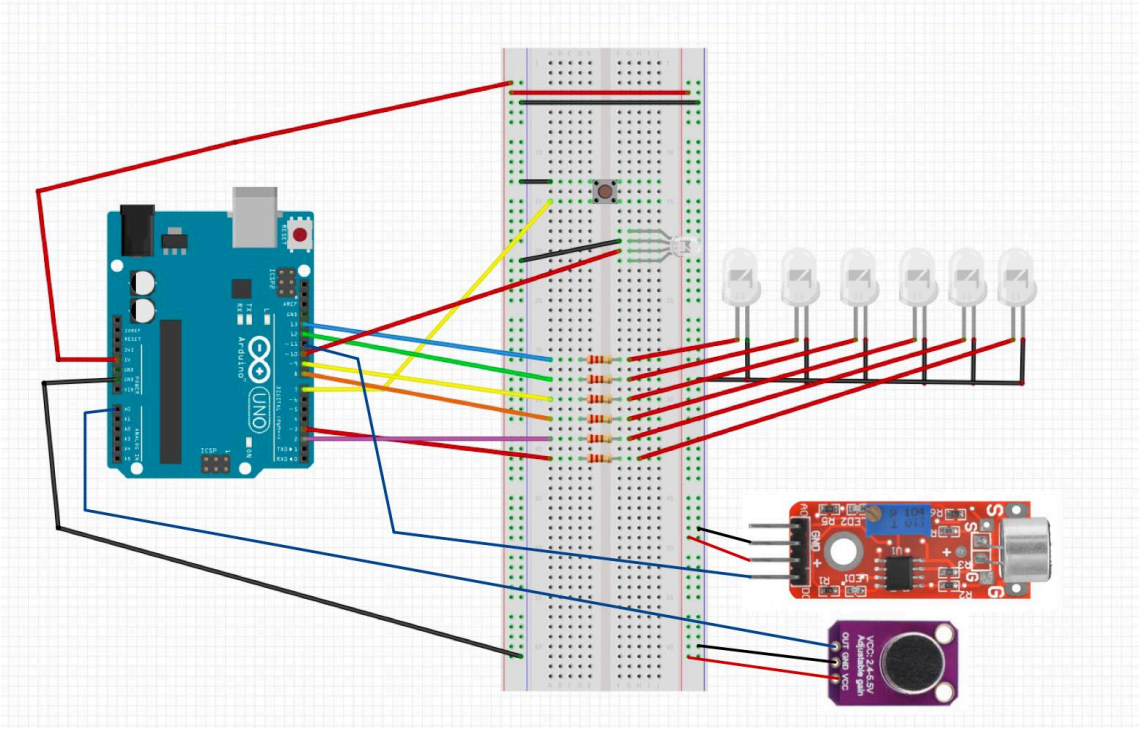


Figure 5: Sound Sensors, Button, LED schematic