UNIVERSITY of WASHINGTON | BOTHELL

B EE 484 Sensors and Sensor Systems

Automatic Guitar Tuner

Group Members: Will Song, Jacob Barrett, Saeid Amoozgar, An Vu, Launnie Ginn

Fall 2019

Instructor: Seungkeun Choi, Ph.D

Date of Submission: December 6, 2019

Table of Contents

ABSTRACT

The average person enjoys automatic controls for devices within the home. Despite advancements, the average person still manually tunes their guitar. Moreover, guitars require frequent tuning when strings change tension and frequency with environmental changes. Beginners struggle with tuning and need instructor assistance, particularly when strings break. Automatic guitar tuners help beginners tune independently and practice more often. However, previous automatic guitar tuners experience torque, detection, and acoustic support limitations. This Arduino-based automatic guitar tuner project adds acoustic-only support, better motor control, and LCD display. Tuning occurred within 4 cents with a simple DC geared motor and compact acoustic microphone.

Introduction

Tuning a guitar can be confusing for a novice guitar player, and time-consuming for even the more experienced musicians. Ideally, a guitar would never go out of tune. Since a perpetually tuned guitar is not currently possible, the best alternative is a device that can tune a guitar automatically. There are a few existing guitars tuners that make this possible, but they are expensive and experience issues with torque and tuning accuracy. Our team has developed a guitar tuner that uses an electret condenser microphone to detect the frequency of a played string, and then automatically adjusts the tuning peg to bring it in tune. Additionally, the tuner also contains a quarter inch jack that can be used instead of the microphone, making this device versatile for different types of guitars.

Motivation

This section outlines existing automatic guitar tuners on the market, basic operating principles, and motivation for proposed improvements.

Discussion of existing products

There are two main categories of automatic guitar tuners. The first category replaces traditional guitar tuning pegs with powerhead tuners consisting of servo motors (ex: Tronical Tune system)(16). These systems are fixed to one instrument, costly, heavy, and lack flexibility for the guitar player with more than one instrument.

The second category of guitar tuner is an external DC motorized tuner which turns the tuning pegs one-by-one based on the pitch of the string. Commercial versions of the external DC motorized tuner include Roadie (9) and JOWOOM (Figure 1)(7). Common issues with the external DC motorized tuners include excess torque that breaks guitar strings. Other issues include poor frequency detection that leads to inaccurate tuning. Most iterations of the DC motorized tuner incorporate acoustic or electric-only support, but not both. This project was derived from an Arduino-based version (2) of the external DC motorized tuner with more flexible acoustic and electric guitar support, in addition to an enhanced user interface.

Figure 1 - From left to right: Roadie, JOWOOM, and Arduino-based tuner using SIMULINK

Operating principles

Using filtered and amplified output from the guitar, the sound of the string is captured. The frequency of the audio is determined, and then compared with a reference frequency. The motor connected to the tuning keys then turns the guitar peg, which changes the tuning, thus the frequency recorded. This effectively makes a simple control system. In order to change the string you would like to tune, a button connected to the microcontroller is sampled and the microcontroller cycles through the standard tuning reference frequencies.

Pros and cons of current systems

The tuning key replacement type systems are semi-permanent, heavy, and difficult to install. If the consumer decides to have the device installed professionally, there is a cost added to the initial investment of approximately $300 - $800 (2). Although this type of device is costly, consumer reviews indicate it works as expected. It tunes all 6 strings at the same time, so the time to tune is faster than tuning each string one-at-a-time.

In contrast to tuning key replacement, the handheld type tuner adjusts only one string at a time. It can be argued that the point of this type of tuner is not to be fast, but to be easier and less expensive. The handheld tuner does not require any installation, and the cost of the device is less than $100; a reasonable amount for a beginning musician. Despite the advantages of low cost and portability, consumers have noted that handheld tuners are inaccurate or have bugs. Most of these devices don't work with both electric-acoustic and acoustic-only guitars either.

Proposed improvements

In order to improve upon the existing guitar tuning systems, this project focuses on the more cost-effective, portable handheld system with accurate tuning, user-friendly interface, and flexible acoustic guitar compatibility. Accuracy was achieved using a tuned and robust frequency detection algorithm. To provide more meaningful user-interface to the guitar player, an LCD display was implemented. In addition to input from an electric guitar pickup, a microphone was installed to support acoustic-only guitars.

System Overview

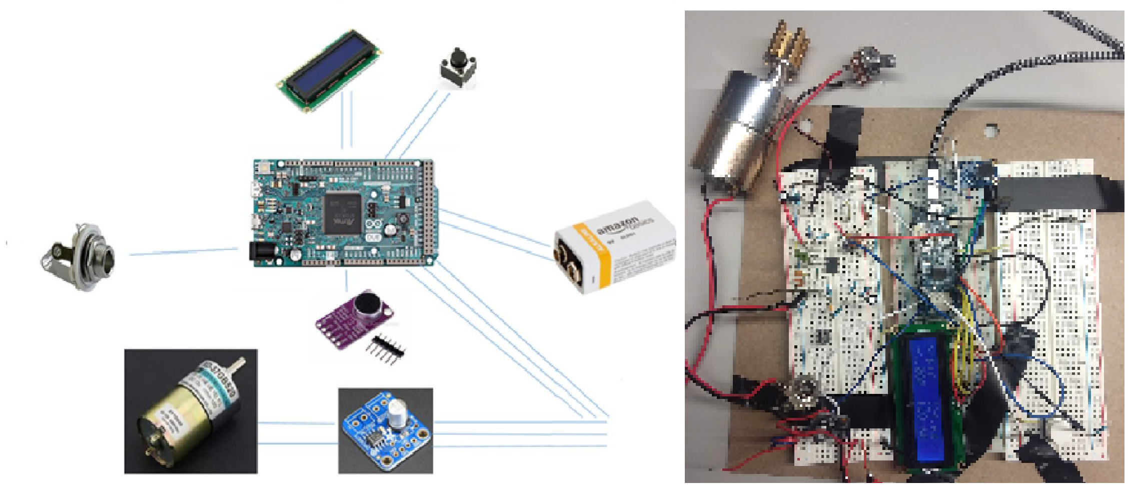
This section provides a breakdown of the components used and a detailed operation of the major subsystems.

Equipment used

- Arduino Zero
- MAX9814 Analog Microphone w/Gain

- Great Divide 38”, 6-string, acoustic-electric guitar
- Guitar cable (¼ “ Jack)
- Audio Jack: 1/4" Mono Jack
- Op amp TL972
- Wires: 20 AWG stranded
- LEDs
- LCD Display
- Push button
- 9V batteries & connector
- 6mm Flange Shaft Coupling Guide Shaft
- M2.5 Hex Standoff Assortment Kit
- Body 4-lead W/ 1m Cable and Connector
- Adafruit DRV8871 DC Motor Driver
- DFRobot DC Gear Motor 12V 100RPM
- Arduino IDE
- Digital caliper
- Korg CA-50 Guitar tuner w/Cent display
- Tektronix Oscilloscope MSO 3012 MIxed Signal Oscilloscope
- Tektronix AFG 3022B Dual Channel Arbitrary/Functional Generator
- Tektronix DMM 4020 5-½ Digit Multimeter

Figure 2 - Automatic guitar tuner system overview



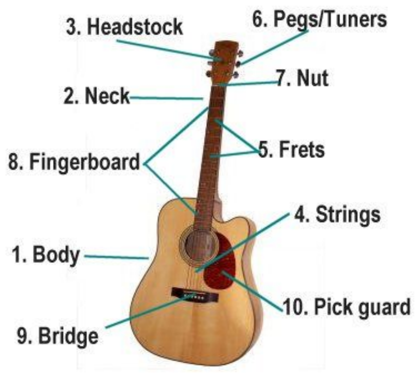
Subsystems

This section explains the hardware, software, user-interface, and sensor subsystems implemented in the automated guitar tuner project.

Guitar

For the context of this project, two guitar types were considered: acoustic-electric and acoustic-only. An acoustic-electric guitar looks like a standard, 6-string, acoustic guitar with an electric pickup for plugging the guitar into an amplifier or sound board without the need for external microphones. A standard, 6-string guitar with part names is shown in Figure 3 for reference (12).

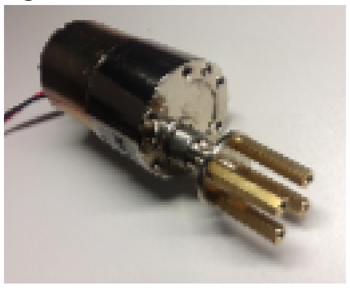
Figure 3 - Guitar part names



Motor

To manipulate the tuning pegs on the guitar and adjust tension on the guitar strings, the guitar tuner uses a geared DC motor with a screw hub and hex standoffs (spaced 3 mm apart) to connect the motor to the tuning pegs. The motor was powered by a 9V battery, and controlled by the DRV8871 DC motor driver (1).

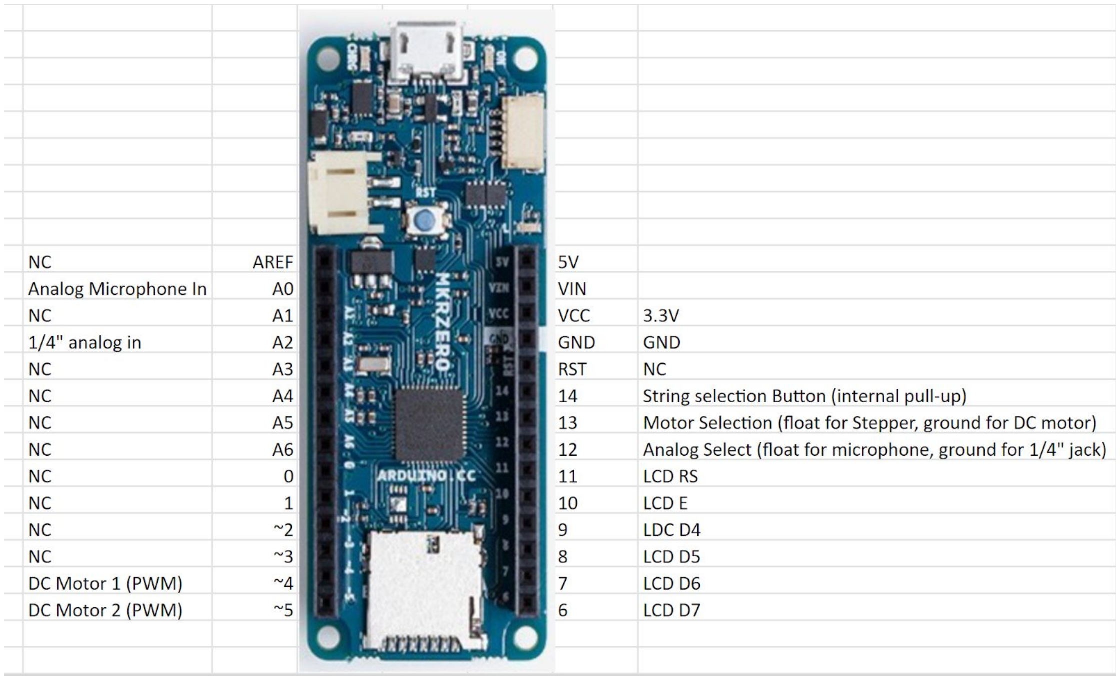
Figure 4 - Geared DC motor and guitar peg manipulator



Micro-controller

The DC motor and motor driver are connected to an Arduino MKRZero microcontroller, which receives input, outputs display information, analyzes the audio frequency, and outputs commands to the DC motor. The MKR Zero has SAMD21 Cortex-M0+ 32bit low power ARM MCU, 3.3 V operating voltage, and 48MHz clock speed (5). The Arduino processes digital and analog input (with ADC converter). The Arduino samples audio at a constant rate, senses voltage levels (0 - 5V), and converts the signal to 0 - 1024 (10-bit) values. It then determines the guitar string's fundamental frequency by detecting maximum voltage change and calculating the period.

Figure 5 - Arduino MKRZero and pin diagram



NC	AREF	5V	
Analog Microphone In	A0	VIN	
NC	A1	VCC	3.3V
1/4" analog in	A2	GND	GND
NC	A3	RST	NC
NC	A4	14	String selection Button (internal pull-up)
NC	A5	13	Motor Selection (float for Stepper, ground for DC motor)
NC	A6	12	Analog Select (float for microphone, ground for 1/4" jack)
NC	0	11	LCD RS
NC	1	10	LCD E
NC	~2	9	LDC D4
NC	~3	8	LCD D5
DC Motor 1 (PWM)	~4	7	LCD D6
DC Motor 2 (PWM)	~5	6	LCD D7

Audio-input

To sample acoustic guitar audio, a capacitive microphone with analog output was used. To capture electric guitar audio, a ¼" audio jack was used. The two inputs allow the user to tune more than one type of guitar. Because the average guitar audio output amplitude is small, approximately 100 mV, the signal was amplified using 30 - 60 dB gain to allow the Arduino to sense voltage change. The ¼" jack audio circuit used a TL972 op-amp with 29.5 dB gain to amplify audio signals from electric guitars. The ¼" jack audio circuit also contains a potentiometer to adjust gain and eliminate clipping. The acoustic microphone has built-in 60 dB gain and connects directly to the Arduino Zero. The gain on the acoustic microphone can also be adjusted between 40, 50, and 60 dB to mitigate clipping.

Figure 6 - ¼" jack audio input amplifier circuit schematic (2)

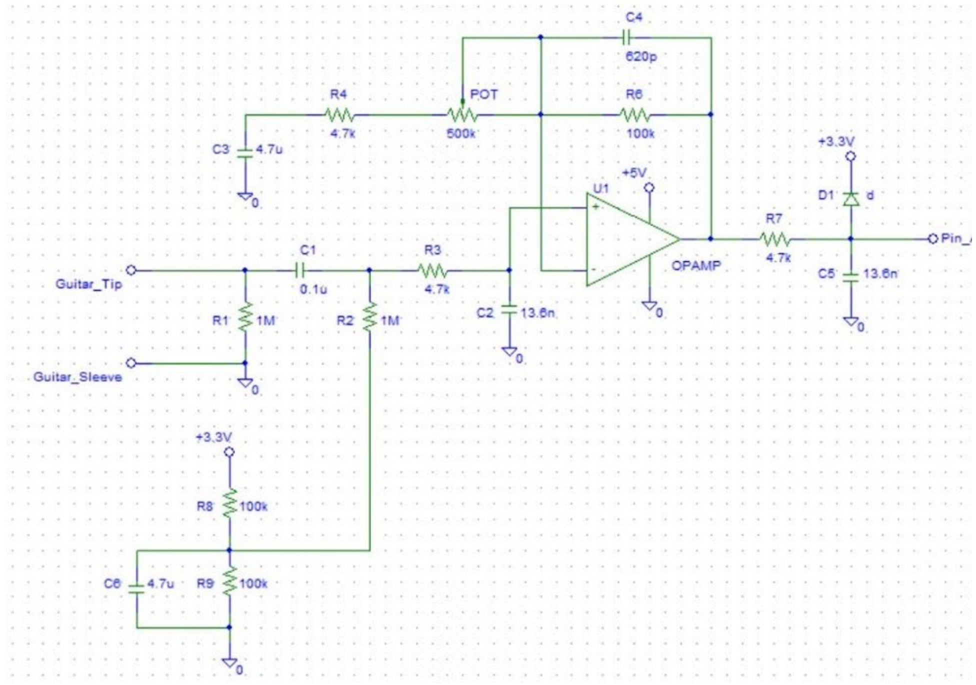
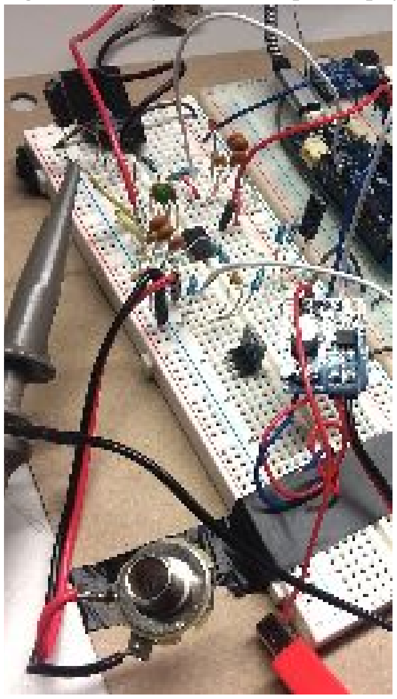


Figure 7 - ¼" jack audio input amplifier circuit



User-interface

The user-interface includes a push button and a small LCD display (3). The push button allows the user to select which string they want to tune (6). The LCD displays the frequency detected, desired frequency, error messages, and the selected string number 1 - 6.

Figure 8 - String selector circuit schematic (2)

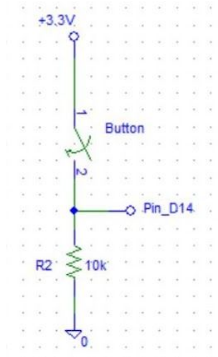


Figure 9 - LCD display circuit schematic

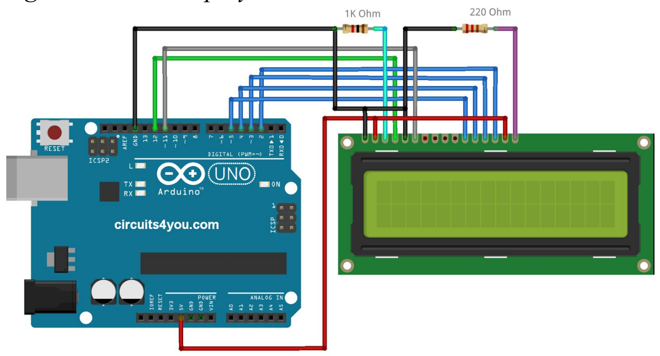
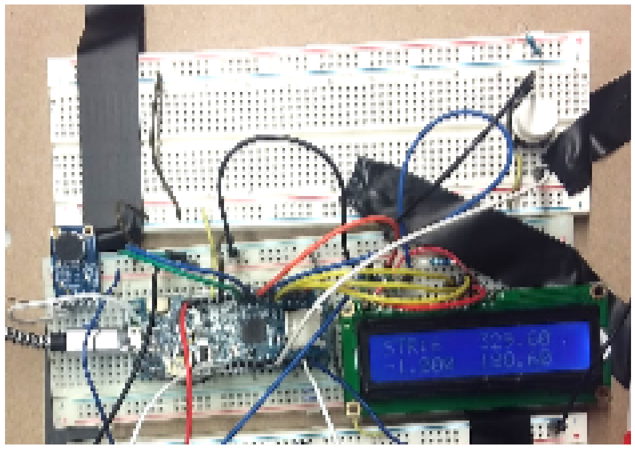


Figure 10 - String selector and LCD circuit



Software

Using captured audio, the frequency of the string is found. We used a standard arduino library (AudioFrequencyMeter) to do this (4). The frequency is run through the software filter shown in Figure 12. As Figure 12 shows, if the difference between the recorded frequency and the ideal frequency less than 1 Hz, the motor will not turn. If the frequency is less than ½ the accepted frequency or more than twice the accepted frequency, the tuner assumes the wrong string was selected or the string was not installed correctly and the motor will not turn. If the frequency is low and within the acceptable range, the motor turns clockwise. If the frequency is high and within acceptable range, the motor turns counterclockwise. If the string has not recently been plucked, the motor should not turn. This is achieved by setting the audio input amplitude threshold above average room noise levels.

Figure 11 — Control System Overview

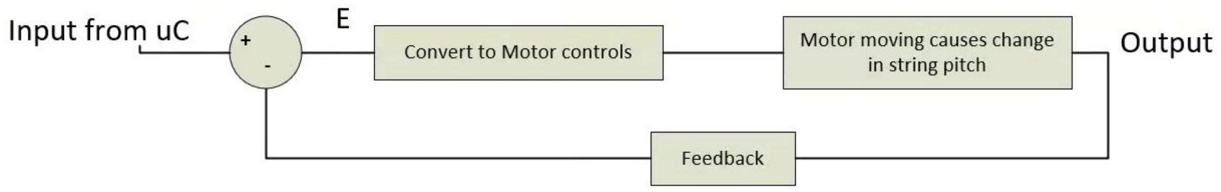

Input from uC
+
-
E
Convert to Motor controls
Motor moving causes change in string pitch
Output
Feedback

Figure 12 — Filter Overview

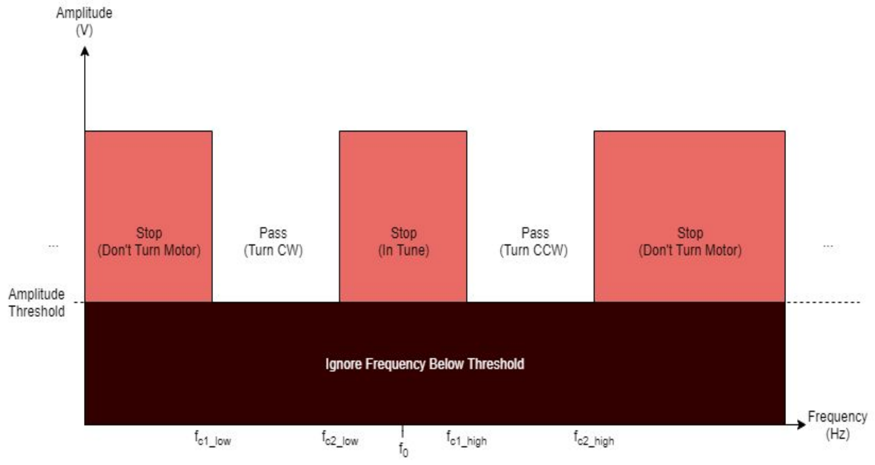
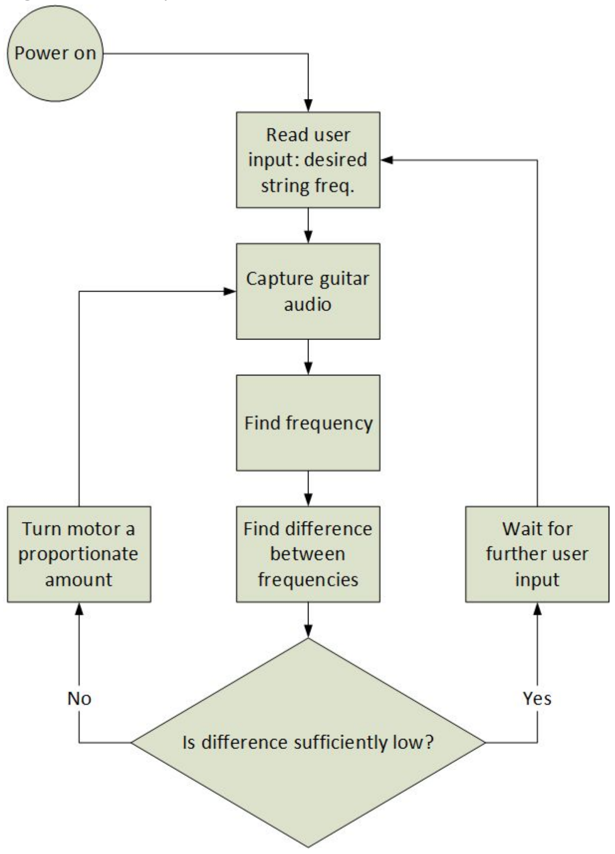


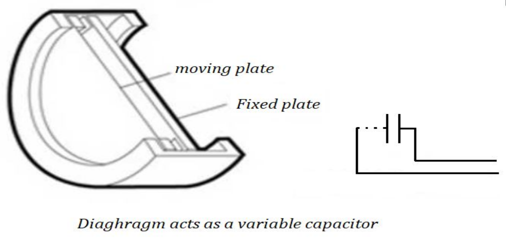
Figure 13 — Software Flowchart


Power on
Read user input: desired string freq.
Capture guitar audio
Find frequency
Find difference between frequencies
Turn motor a proportionate amount
Wait for further user input
No
Yes
Is difference sufficiently low?

Microphone as a Sensor

A microphone is a sensor which converts sound input (air pressure) to an electrical signal. When the moving plate (diaphragm) vibrates in time with the sound wave, the distance between the plates, and hence the capacitance, is changed. The change the capacitance results in current flow as the sensors output signal. The MAX9814 implements an electret condenser microphone, which is a type of condenser microphone with permanently charged capacitor plates (18). While the electret condenser microphone is pre-charged, the device requires an external voltage source to power the amplifier that provides built-in gain.

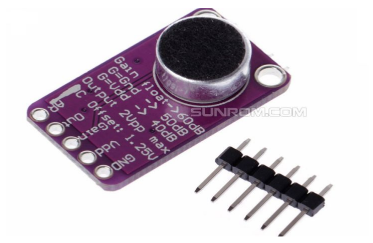
Figure 14 - Condenser microphone as a flexible, sound sensitive capacitor

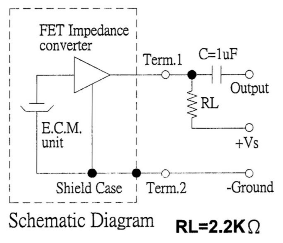


MAX9814 Analog Microphone

This microphone amplifier module was chosen due to its wide frequency range of 20Hz to 20KHz and built in amplifier. The amplifier is a Maxim Integrated MAX9814 with Automatic Gain Control (AGC), three gain settings (40dB, 50dB, and 60dB), and input and output ranges that are compatible with the Arduino MRK Zero microcontroller (8). The AGC adjusts gain to prevent clipping at the output when the input level is too high. Adjusting gain prevents distortion and cleans the input signal to the microcontroller ADC converter.

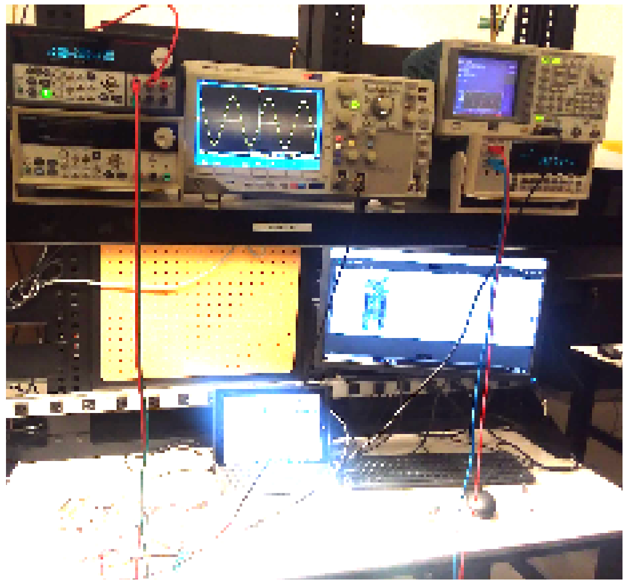
Figure 15 - MAX9814 analog electret microphone & electret microphone schematic

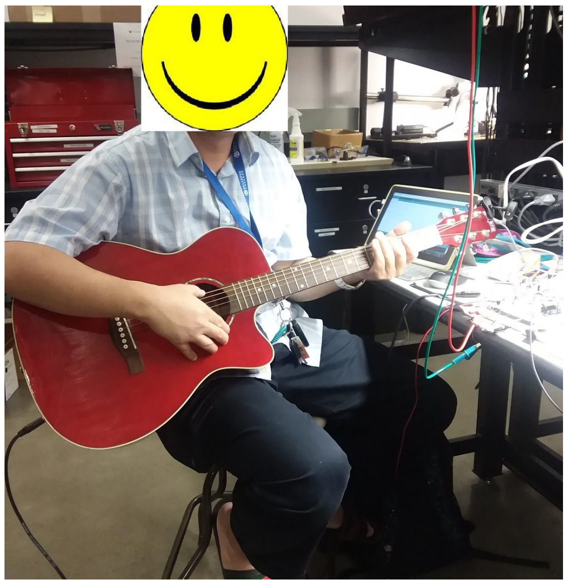




Results

Figure 16 - Left to right: Measurement setup and live acoustic guitar signal testing





Guitar peg manipulator measurement for design

The guitar pegs are manipulated by four M2.5 hex standoffs attached to a 6 mm screw hub that is affixed to the DC motor shaft. The hex standoffs must grip the guitar pegs without excess space to maximize torque and tuning accuracy. Hex standoffs are spaced 3 mm apart according to standard acoustic guitar peg thickness measurements in Table 1. Guitar pegs were measured with a digital caliper. Because peg thickness was common between varying guitar sizes, hex standoffs were spaced according to this dimension.

Figure 17 - Guitar tuning peg diagram

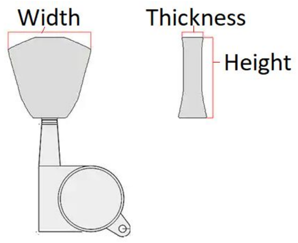


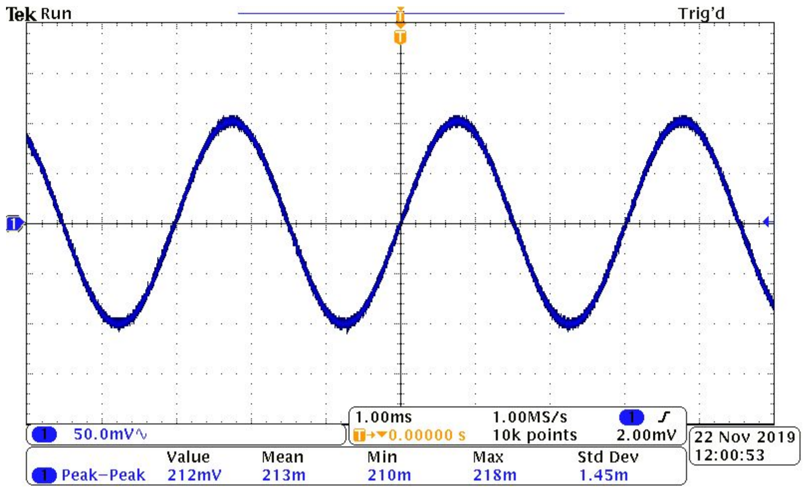
Table 1: Adult and kid-sized acoustic guitar peg measurements

Guitar Type	Peg Length (mm)	Peg Thickness (mm)	Peg Height (mm)
Adult	19.1	2.8	12.9
Kid	17.4	2.7	9.4

Measured input/output signals of audio circuits with oscilloscope

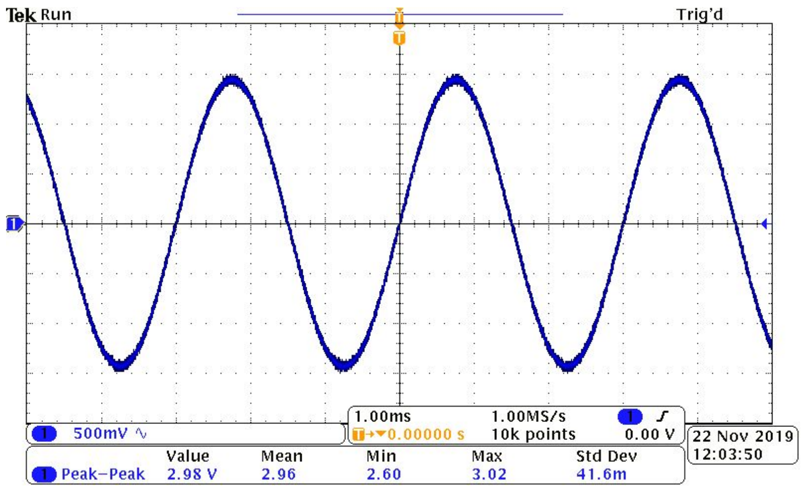
Audio signals required gain adjustment to eliminate clipping and ensure optimal input waveform for frequency detection. Output signals were measured with an oscilloscope. The ¼” jack audio input guitar signal (Figure 18 & 19) was simulated with a 100 mVpp, 333 Hz sine wave. Figure 19 demonstrates an amplified guitar signal without clipping. The acoustic microphone audio output (Figure 20) and gain was tested using a human voice at approximate guitar volume. The signal quality of the human voice signal may be partially attributed to physical fluctuations from the source.

Figure 18 - ¼" jack input waveform: 100 mVpp, 333 Hz



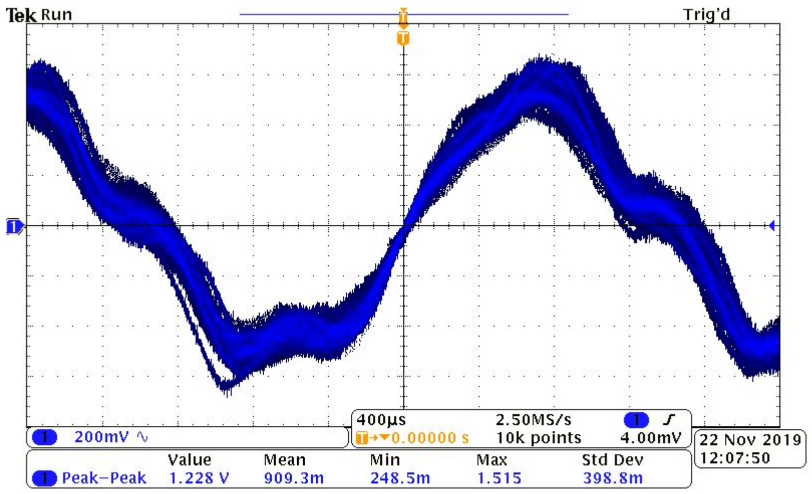
```
             Value    Mean    Min     Max     Std Dev
Peak−Peak    212mV    213m    210m    218m    1.45m
```

Figure 19 - ¼" jack output waveform: 2.98 Vpp, 333 Hz, Gain: 29.5 dB



```
             Value    Mean    Min     Max     Std Dev
Peak−Peak    2.98 V   2.96    2.60    3.02    41.6m
```

Figure 20 - Analog Mic Output Signal (Human Voice), Gain: 60 dB



```
             Value    Mean     Min      Max     Std Dev
Peak−Peak    1.228 V  909.3m   248.5m   1.515   398.8m
```

String tuning outcome

The 6 guitar strings on an electric-acoustic guitar were tuned using the automatic guitar tuner in acoustic-only mode (with acoustic microphone). The frequency of the guitar string was measured before and after tuning with a Korg CA-50, and the resulting measurement in cents was converted to Hertz using the following formula:

$$1\ cent\ =\ 1200\frac{\ln\left(\frac{f_2}{f_1}\right)}{\ln 2}$$

, where f_1 and f_2 are detected and accepted frequencies in Hertz (15). The results in Table 2 indicate the automatic guitar tuner adjusted the guitar strings within 0 to 4 cents, or 0.57 to 2.07 Hz, of the accepted frequencies (13). For reference, the just noticeable difference (JND) in pitch is 5 cents (14). The 4 cent offset may be attributed to ambient noise, frequency detection, and motor turning variation.

Table 2 - Measured string frequency before and after tuning

Note	**Note Frequency (Hz)**	**String Frequency Before Tuning (Hz)**	**String Frequency After Tuning (Hz)**
Low E	82.4	81.83 (-12 ¢)	82.4 (0 ¢)
A	110	109.37 (-10 ¢)	109.87 (-2 ¢)
D	146.8	145.95 (-10 ¢)	146.96 (2 ¢)
G	196	194.87 (-10 ¢)	195.89 (-1 ¢)
B	246.9	245.47 (-10 ¢)	247.47 (4 ¢)
High E	329.6	326.77 (-15 ¢)	328.84 (-4 ¢)

Conclusion

The automatic guitar tuner developed in this project uses an Arduino MRK Zero microcontroller to convert analog inputs into a detected frequency, which is compared against the defined frequency of a selected string. It then controls a DC motor through a motor controller to adjust the tension of the guitar string. Information on the detected frequency and the offset from the defined frequency are displayed on the LCD for the user. Overall, the results were as expected with the exception of noise tolerance. While the automatic guitar tuner adjusted strings within the JND range utilizing the analog microphone, it was susceptible to high-noise environments. Suggested improvements include:

- Rather than using a single microphone, a microphone array could isolate the target sound while reducing noise interference.
- More advanced and accurate frequency detection algorithms could be implemented with a faster microcontroller.
- Improving the frequency detection and motor feedback loop so the motor runs according to the distance from the desired frequency.

***Q&A from the demonstration and presentation*:**

- How long do you adjust the string successfully?
  - For most strings, 3 to 5 strums resulted in tuning close to 99% of desired frequency.
- Regarding the LCD display: What is the meaning of the positive and negative number?
  - A: positive number indicates the reading frequency is higher standard frequency, negative number indicates the reading frequency is lower standard frequency,
- Regarding Results: Did we measure the time it took to tune each string?
  - We didn't measure the time but instead we counted how many times we need to play a string to be tuned.

    - The timing was also qualitatively observed. Overall, the amount of time it took to tune the guitar with the automatic guitar tuner was comparable to the time it takes to manually tune the guitar.

References

1. “Adafruit DRV8871 Brushed DC Motor Driver Breakout.” Adafruit Learning System, https://learn.adafruit.com/adafruit-drv8871-brushed-dc-motor-driver-breakout/usage. Accessed 2 Dec. 2019.
2. “Arduino-Based Automatic Guitar Tuner.” Hackster.Io, https://www.hackster.io/metrowest_aug/arduino-based-automatic-guitar-tuner-2093fe. Accessed 2 Dec. 2019.
3. Arduino Frequency Counter Tutorial with Circuit Diagrams & Code. https://circuitdigest.com/microcontroller-projects/arduino-frequency-counter-circuit. Accessed 2 Dec. 2019.
4. Arduino-Libraries/AudioFrequencyMeter. 2015. Arduino Libraries, 2019. GitHub, https://github.com/arduino-libraries/AudioFrequencyMeter.
5. Arduino Zero. https://store.arduino.cc/usa/arduino-zero. Accessed 2 Dec. 2019.
6. Button design: https://www.arduino.cc/en/Tutorial/StateChangeDetection.
7. “Electric String Winder | Online Digital Guitar Tuner.” JOWOOM, https://www.jowoom.com/jowoom-insider. Accessed 2 Dec. 2019.
8. Industries, Adafruit. Electret Microphone Amplifier - MAX9814 with Auto Gain Control. https://www.adafruit.com/product/1713. Accessed 2 Dec. 2019.
9. Industries, Band. Roadie Automatic Guitar Tuner. https://www.roadietuner.com. Accessed 2 Dec. 2019.
10. “Microphones” Compotech, https://en.compotech.se/products/loudspeakers-microphones/mikrofoner/. Accessed 2 Dec. 2019.
11. “Mixdown.” Mixdown, http://www.mixdownmag.com.au/. Accessed 2 Dec. 2019.
12. Music for Kids: Parts of the Guitar. https://www.ducksters.com/musicforkids/guitar_parts.php. Accessed 6 Dec. 2019.
13. Note Frequencies. https://www.seventhstring.com/resources/notefrequencies.html. Accessed 2 Dec. 2019.
14. Rossing, Thomas D., The Science of Sound 2nd Ed, Addison-Wesley 1990

15. The Use of Cents for Expressing Musical Intervals. http://hyperphysics.phy-astr.gsu.edu/hbase/Music/cents.html#c4. Accessed 6 Dec. 2019.
16. Tronical Tune - Tronical - GET THE GUITAR DNA - Autotunes Your Guitar in Seconds. https://www.tronicaltune.com/. Accessed 5 Dec. 2019.
17. What Do Guitar Tuner Accuracy Specs Mean? | Atlantic Quality Design, Inc. Musical Products. http://zerocapcable.com/?page_id=225. Accessed 3 Dec. 2019.
18. What Is an Electret Microphone? http://www.learningaboutelectronics.com/Articles/Electret-microphones. Accessed 6 Dec. 2019.

<u>Appendix A</u> - Automatic Guitar Tuner Arduino Sketch

```
#include <LiquidCrystal.h>
#include <Wire.h>
#include <AudioFrequencyMeter.h>

LiquidCrystal lcd(11,10,9,8,7,6);
AudioFrequencyMeter meter;

const int motorLeft = 5;
const int motorRight = 4;

const double string1Frequency = 82.4;
const double string2Frequency = 110;
const double string3Frequency = 146.8;
const double string4Frequency = 196;
const double string5Frequency = 246.9;
const double string6Frequency = 329.6;
    double sf=0,
    frequency = 0;
const int button = 1;
    int counter = 0;
    int buttonState = 0, buttonLastState =0 ;

void stringNumber(){ //Check for selected string and assign accepted frequency
  buttonState = digitalRead(button);
  if(buttonState != buttonLastState){
    if(digitalRead(button)==LOW){
```

```
    if (counter <6){
      counter = counter + 1;
    }
    else{
      counter = 1;
    }
  }
}
switch (counter){
    case 1:
    sf = string1Frequency;
    break;
    case 2:
    sf = string2Frequency;
    break;
    case 3:
    sf = string3Frequency;
    break;
    case 4:
    sf = string4Frequency;
    break;
    case 5:
    sf = string5Frequency;
    break;
    case 6:
    sf = string6Frequency;
    break;
    default:
    sf = 0;
```

```
      break;
    }
  delay(50);
  buttonLastState = buttonState;
} //end stringNumber function

int errorCheck(double f){ //check for accepted pass bands, if stop then trigger error
  int e = 0;
  switch (counter){
      case 1:
      if ( f < 40.7 | f > 166){
       e = 1;
      }
      break;
      case 2:
      if ( f < 54.5 | f > 222){
        e = 1;
      }
      break;
      case 3:
      if ( f < 72.9 | f > 295.6){
        e = 1;
      }
      break;
      case 4:

      if ( f < 97.5 | f > 394){
        e = 1;
        }
      break;
      case 5:

      if ( f < 245 | f> 495){
        e =  1;
      }
      break;
      case 6:
      if ( f < 164 | f > 661 ){
        e = 1;
      }
      break;
      default:
        e = 0;
      break;
  }
  return e;
}

void LCDrange(float freq, float noteFreq){ //calculate the difference between detected and
accepted frequencies
    double difference = freq - noteFreq;
    lcd.print(difference);

}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(button, INPUT_PULLUP);
  pinMode(motorLeft,OUTPUT);
  pinMode(motorRight,OUTPUT);

  lcd.begin(16, 2);
  lcd.setCursor(0,0);
  lcd.print("please select:");
  delay(3000);
  lcd.clear();
  //frequency detection parameters -> tunes frequency detection
  meter.setBandwidth(10.00, 800);
  meter.setAmplitudeThreshold(70);
  meter.begin(A0, 45000);           // Intialize A0 at sample rate of 45kHz
}

void loop() {
  // put your main code here, to run repeatedly:
    frequency = meter.getFrequency(); //detect frequency
    stringNumber();
    lcd.setCursor(0, 0);
    lcd.print("STR:");
    lcd.setCursor(4, 0);lcd.print(counter);
    lcd.setCursor(8, 0);lcd.print(sf);
    lcd.setCursor(0, 1);
    if(errorCheck(frequency) == 1){
      lcd.print("Error");
    }
    if(errorCheck(frequency) == 0) {
       lcd.print(frequency);
      lcd.setCursor(8, 1); LCDrange(frequency,sf);
      delay(500);
      if(frequency > 0 & counter > 0){
        if (frequency > (sf + 1)) {
          digitalWrite(motorLeft, HIGH);
          delay(100);
          digitalWrite(motorLeft, LOW);
        }
        else if (frequency < (sf - 1)){
          digitalWrite(motorRight,HIGH);
          delay(100);
          digitalWrite(motorRight,LOW);
        }
      }
    }
  }
```