Washington University in St Louis

ESE 498 Senior Design Project

# Automatic Guitar Tuner

*Authors:*

Stephen Gower

Mark Jajeh

*Supervisor:*

Robert Morley

*Submitted in Partial Fulfillment of the Requirement for the BSEE Degree, Electrical and Systems Engineering Department, School of Engineering and Applied Science, Washington University in St. Louis*

May 2, 2016

## Contents

## List of Figures

*Special thanks to our Adviser Robert Morley*

## Student Statement

*During this project, Mark Jajeh and Stephen Gower have certified that they have applied ethics in the process and selection of our final design. We also certify that we followed the Washington University in St Louis code of Academic Integrity and Honor.*

## Abstract

In the outset of this project, our goal was to design a device to handle the tuning of a guitar in an almost-entirely automatic process. Following this, we have designed a hand-held device that connects via audio chord to the user's phone and that when applied to the tuning keys of a guitar, will tune the strings to the proper tone following the user's stimulation of the corresponding strings. This process is semi-automatic in the sense that the user must handle the device and strum the strings, but the tuning is left to automation. Our device is designed to tune the tone of the strings to within 0.3 Hz of the ideal tones and the app is designed to set for alternative tunings, increasing the utility of the design.

## Problem Formulation

### Problem Statement:

In the realm of music and in particular, guitars, automatic tuners are still a luxury. Early on in their lifetime, these tuners have existed as part of the individual guitar designs, guitars which at the time cost upwards of $10000. That's not to say the tuning itself was the driving component of the expense, just that it was initially restricted to more expensive niche models. Then tuners started to pop up that could be installed into the head of the guitar, again these are very inflexible tuners and if you wanted to tune 5 guitars you needed to purchase and install 5 of these tuners and that's assuming they could fit the head style of the guitar. There are few examples of a better solution to this problem; a hand held tuner that can be manually placed on keys.

### Problem Formulation:

We set out to design a tuner that could effectively tune multiple guitars or that didn't require an installation to operate. This would appeal to casual users who don't want to modify their guitar, it would also appeal to bands who often need to keep multiple instruments in tune. Further, we aim to make the product cost effective to appeal to a consumer that doesn't want to spend a lot of money on guitar models that include a built in tuner. Thus our solution needs to have these traits:

1. Cost effective (Retail price less than 100 dollars)
2. Ease of use
3. Variety and value produced at low cost wherever possible

This final point speaks to the potential of our solution, as we set out to design a tuner that integrates seamlessly with an iPhone. This way, we can develop a quality user interface and produce value for the consumer with zero production cost.

## Project Specifications

In developing our specifications for our design, we reach a list of crucial aspects that the product must address to be successful. To begin:

- The device must be capable of detecting a guitar string that is out of standard tuning and specifying the movement needed to bring the string into tune.

A layout of standard tuning is shown below:

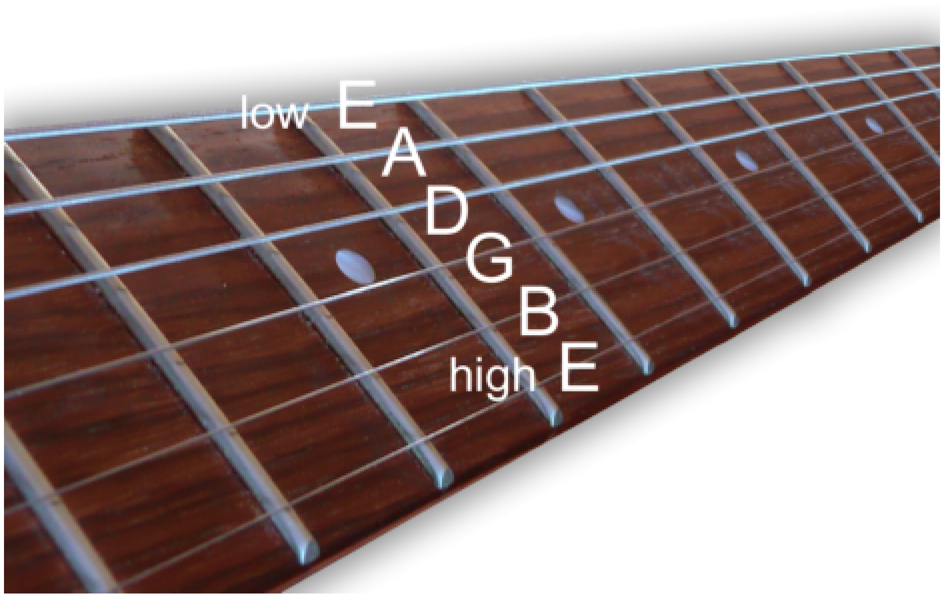

Figure 1: Standard Tuning Layout[1]

The corresponding frequencies for these strings goes as: Low E is 82.41 Hz, A is 110 Hz, D is 146.83 Hz, G is 196.0 Hz, B is 246.94 Hz and high E is 329.63 Hz [2]. So with these targets in mind, our specification now needs an accuracy component to shoot for, thus our next specification is:

- To be competitive in the market, our tuner must be capable of tuning to within a cent of the target frequency.

Most manual tuners in our research specified an accuracy of within a cent of the ideal fundamental frequency. For low E a cent is about 0.3 Hz and as a cent is proportional to the frequency, this is the smallest bounds we have on the accuracy of our tuner. Our next specification is:

- The device must be capable of automatically manipulating the tuning keys to bring the strings into tune. The user input must be minimal here, with the most required of the user being their compliance in placing the device on the correct keys.

This is one of the requirements that specifies the user input. The other user input required that will be implicit, is that the user will need to stimulate the string they are tuning during the tuning process.

# Concept Synthesis

## Literature Review

Over the course of our project, there were a few key resources that helped inform us on how to tackle the problem at hand. First, we found an online resource made by an enthusiast that provided code and discussion on the control of the BYJ48 stepper motor, which can be found in the references[3]. Similarly, in our processing of the guitar tones, Mark pulled heavily from his coursework in Digital Signal Processing and the textbook used in that course[4]. Similarly, many of our solutions and approaches came from our weekly discussions with our adviser, Robert Morley. Below is a list of the most relevant literature we refereed to over the course of this design process.

- WUSTL ESE 482: Signal Processing/ FFT Analysis
- WUSTL ESE 232: Circuit Design/Analysis
- Dennis Mell: 3D printing information
- vDSP Library: Signal Processing on iOS[5]
- Roadie Tuner: Similar Design/Competition

## Concept Generation

We can summarize our design concept with the following list of parts and necessary function.

- Hardware
  - Motor
    - Size: Must fit comfortable in the hand.
    - Cost: Need to keep cost low
    - Torque: Must have enough power to turn pins.
  - Pin Clamp
    - Attachment to motor

    - Grip on pins
    - Ease of attachment/detachment
  - Microcontroller
    - Cost
    - Computational Power
    - I/O ports
    - User Interface capability
- Software
  - User Interface
    - Ease of use
    - Functionality
    - Cost
  - FFT Computation
    - Accuracy
    - Filtering
    - Development time
  - Audio processing
    - Required hardware
    - Signal quality
    - cost
  - Motor Control
    - Cost
    - Compatibility
    - Accuracy

## Concept Reduction

The one design that was constant through all possible ideas was the use of a stepper motor. We decided to use a BYJ48 stepper motor and ULN2003 stepper motor driver. We chose the BY4J specifically due to its small size and torque rating and the ULN2003 due to its compatibility with the motor

We then needed to find a easy for the motor to turn a pin on the guitar. Our initial search came up empty and we quickly decided designing our own would be the best course of action. The design would be very simple to create in CAD and we had access to a 3D printer thanks to Professor Mell. This also kept cost and development time down.

Once we settled on a motor we turned our attention to a microcontroller to control its motion. We almost immediately threw out the idea of using a raspberry pi; the extra computing power it provides is not necessary in this application. We then had to choose between an arduino or some type of handmade controller. Although the arduino is a little pricey and provides much more computation then we need; we felt that our previous experience with the arduino ecosystem would provide an advantage moving forward in the design process.

We then tried to build a microphone interfaced with the arduino to record audio. We used an elected condenser microphone with a basic pre-amp circuit to successfully get audio on the arduino. Once we had this working we then tried to compute the FFT on the arduino board. We were not able to find an effective and time efficient way of doing this. It was at this point that we decided to move several parts of our design onto an iPhone. We would use the iPhone to handle all of our software needs. This was not only beneficial in terms of cost but also provided very robust features for future design.

The iOS app did not require many design decisions itself. The user interface and FFT computation were relatively straight forward and handled with some standard objective-c code. We used Apples built in accelerate framework to aid with all FFT related code [5]. Recording audio through the microphone was accomplished using the EZAudio library[6], which is available through the MIT Liscence.

The one design of the app that remained up for debate was how to communicate between the phone and arduino. Our advisor Professor Morley had experience connecting a design to a phone in the past using the standard 3.5mm audio jack. We considered this as well as bluetooth but ultimately settled with the audio jack due to its reliability.

Overall we are confidant that our final design choice satisfies all necessary requirements while minimizing cost and maximizing possible functionality. The iOS app provides a robust means of user control as well as a very accurate measurement system. It also provides an easy way for a user to adjust system properties as they see fit. The 3D printed tuning grip can be manipulated to work with almost any guitar. The control we have over the motor is very precise.

The one area we can improve is our circuit design.If we had time we would focus on completely removing the arduino from our design. and building our own controller. The arduino is by far the most expensive part of our system and we do not require the computing power it provides.

## Detailed Engineering Analysis and Design Presentation

Here is where we get to the meat of our design, after many iterations and choices made, our final design is shown below:

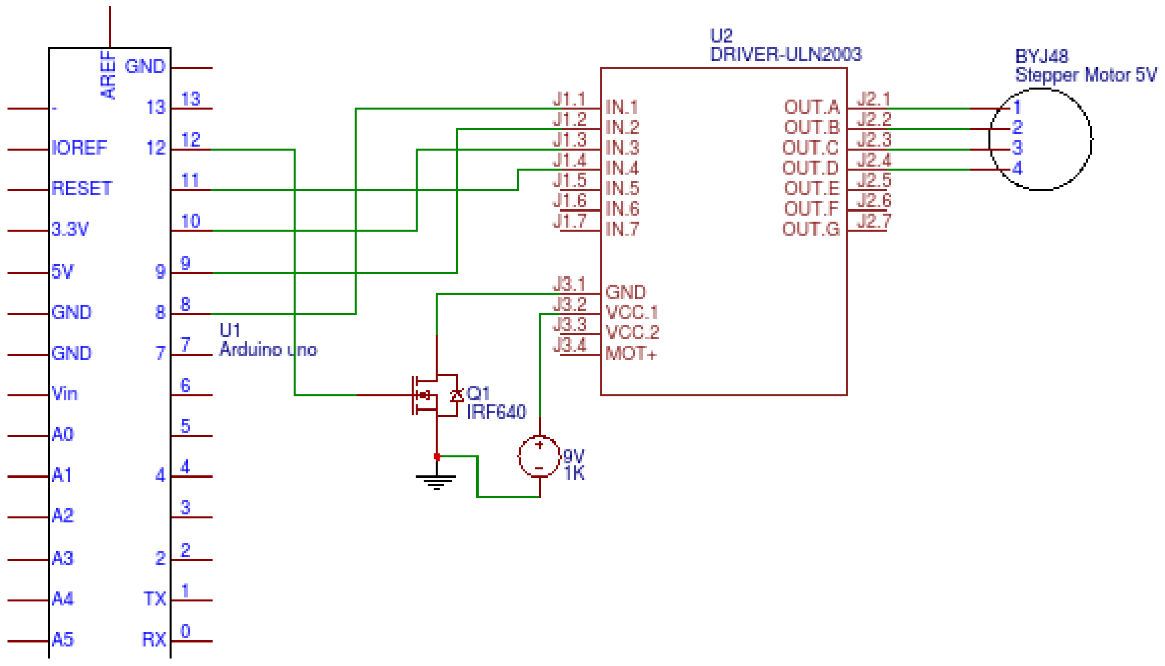

Figure 2: Project Device Layout

The only part not pictured, is the aux port and the iphone. The aux port is connected into the ANALOG IN pins on the arduino with a seperate pin for the left and the right signals. This allows us to connect to the iPhone which will be controlling the system by performing the FFT and deciding how to turn the motor. In this design, the arduino acts as a decoder taking in signals from the phone and interpreting them to control the motor. Otherwise with a left and a right signal at our disposal we could not possibly control the 4-phase stepper motor which required 4 different signals that were unique. However including the phone instead of doing everything on the arudino afforded the design a much better user interface as well as a new dimension of possibility in features.

Our source being considered, guitar strings do not produce a pure tone by nature, they are instead riddled with harmonics as shown:

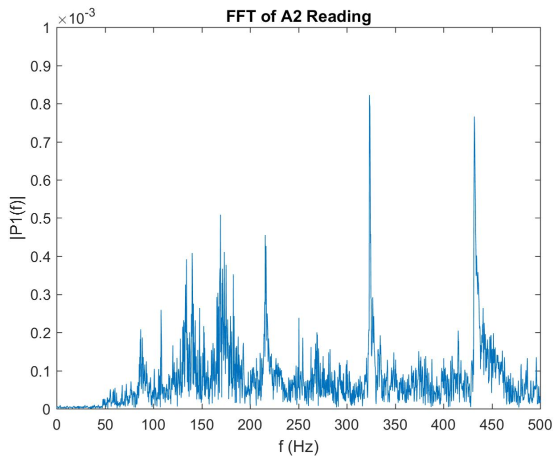

Figure 3: FFT of the A2 String

Recall that the A2 string's ideal fundamental frequency is placed at 110 Hz. Indeed in the above figure we see that frequency represented, but there are more peaks that are even higher. A simple maxfind on this would not suffice as it would be incorrect. These peaks are the result of two things, noise in the environment, or more importantly, harmonics. These harmonics occur at twice the frequency of the fundamental and again at twice the frequency of the first harmonic and etcetera. Our solution here was to implement a filter and limit the boundaries of the bins that we checked. The filter reduced any aliasing of higher harmonics while tossing out the bins we didn't care about ($\pm$ 15 Hz of the ideal fundamental) meant we could better hone in on our current fundamental. This however restricted the guitar as it could not be extremely out of tune.

With this, we proceeded to consider how our algorithm would work to tune a string and we came up with the following scheme:

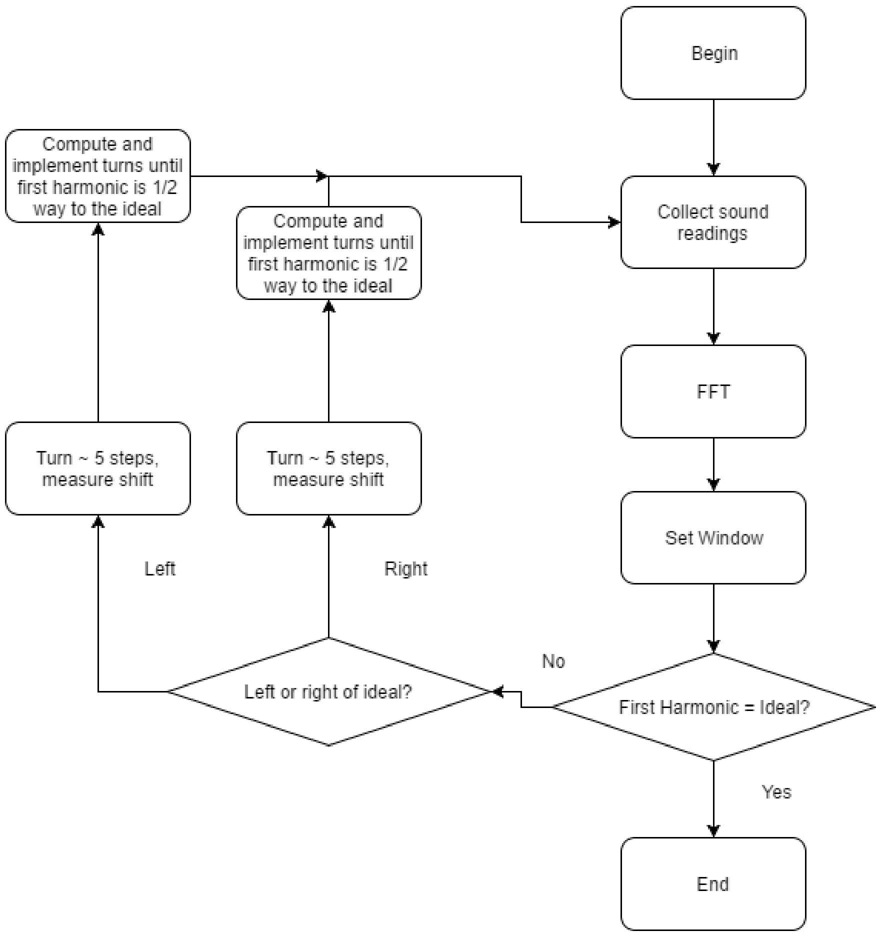
Begin
Collect sound readings
FFT
Set Window
First Harmonic = Ideal?
Yes
End
No
Left or right of ideal?
Left
Right
Turn ~ 5 steps, measure shift
Turn ~ 5 steps, measure shift
Compute and implement turns until first harmonic is 1/2 way to the ideal
Compute and implement turns until first harmonic is 1/2 way to the ideal

Figure 4: Algorithm Flowchart

In our prototype we only had the time to implement and test fully a method that turns based on a pre-measured constant and tunes to the harmonic, not half-way. However as we predicted this method produced oscillatory behavior and inaccuracy if the constant was not well suited to the string. Hence the presented algorithm is our recommendation for the design as tuning to the half-way point instead of directly to the ideal means that any overshoot doesn't actually overshoot the ideal by much if at all. Further, the method of turning a finite number of times and measuring the shift in frequency and using that as a constant means that it's more adaptable to a changing environment and a variety of guitars.

## Cost Analysis

Our design boils down to a short list of parts. These are the ULN2003 stepper motor driver, the BYJ48 stepper motor, the IRF640 transistor used as a switch for the motor, the Arduino IC used to control the stepper motor and finally the Aux port used to interface with the phone. The breakdown of the costs for these components is as follows:

- ULN2003 Driver: $542.5 per 2500 units
- BYJ48 Motor: $ 3960 per 1000 units
- IRF640(IRFR3910PBFTR-ND): $ 716.8 per 2000 units
- ATMEGA328P-AURTR-ND $ 2300 per 2000 units
- Aux Port (CP-2523SJTR-ND): $ 483.2 per 1000 units

Note that the stepper motor is sourced from a non-bulk supplier. From digikey, the same model stepper motor is priced at $ 4.95 per unit or 4950 per 1000. Hence given the above costs if we were to produce 2500 units (the bare minimum to acquire the driver at the listed price), we are looking at a per unit cost of $ 6.16. If we were to estimate additional minor parts (RC and PCB) as well as the cost of the manipulator and casing at a total of 3 dollars additional per unit, it could safely be placed at below 10 dollars per unit for the cost of production. The closest market competitor that currently exists is in the roadie guitar tuner that costs 99 dollars MSRP. If we were to reduce the cost further by finding a cheaper mechanical solution aside from the stepper motor, we could potentially eliminate the need for the ULN2003 driver and the IRF640.

## Bill of Materials

The materials used to construct our design are as follows:

- Arduino Uno (ATMEGA328P)
- BYJ48 Stepper Motor
- ULN2003 Driver

- Aux Port (No ref for our model, but in cost analysis we recommend one)
- IRF640
- Custom Printed Manipulator Piece

During the process of our design we had other solutions that either didn't work or were discarded, the above list covers the materials used in the final design implementation of our project.

## Hazards and Failure Analysis

With this simple design, there are few hazards to consider but they certainly exist. First, the stepper motor is a type of motor that when left idle will still draw current to the coils that maintain the current position. That leads to the motor heating up over use and as we are currently running the motor over the specified voltage, it means more current and more heat. Running a stepper motor over it's specified voltage is not a safety concern as it does not lead to harm to the motor, however the heat can be problematic. To address this we recommend a well ventilated casing surrounding the components and the IRF640's switching role should help to minimize heat build up.

Next, there is a concern that as the tuner is automatically manipulating strings that can be kept at a high tension that it could result in a snapping of strings when strung too taut. In a standard tuning configuration, this can easily be controlled in the algorithm by not allowing it to make any turns that would raise the fundamental frequency above a certain mark. But that is also not the best unless we assume the reading of the frequency to be entirely accurate. Another solution would be to find a ”sweet point” where the motor will output enough torque to reasonably tune but will not but out enough torque to break the string. Even more, we could measure the torque being output and limit it. We did not research this in great detail but it would be a potential concern for a final design.

## Conclusions

In conclusion we are proud to report the successful design and implementation of a cost effective, easy to use, and accurate automatic guitar tuner. We were able to tune an electric guitar to $\pm.5$Hz of a desired tuning with a system that has an estimated per unit cost of $\approx$\$9 Our design relies heavily on the functionality provided by an iPhone and its internal components. We use the iPhone as a user interface,microphone,signal processor,and motor controller. Combining these parts into an iOS app not only reduces cost but it also increases performance due to the quality of the parts within the iPhone.The physical components that make up the rest of our design all serve their respective purposes very well. The BYJ48 stepper motor is small enough to fit in an average hand yet provides plenty of torque to turn each pin on a given guitar. The IRF640 transistor we implement insures that we are not wasting any energy powering our system when it is not turning a pin.

The one component of our design that could be readily changed moving forward is the Arduino Uno (ATMEGA328P). Due to the processing power provided by our iOS app, the Arduino Uno becomes unnecessary. We use the arduino to tell our motor when and which direction to turn, internally this is accomplished by outputting an ordering of sequential pin values to send the motor.Do to the linear and simple nature of this computation, this functionality could be handled by a custom circuit with simple adder that can output each pin configuration as necessary.

Our design could be improved further with a more comprehensive and accurate pulse width modulation code on the iOS app. Currently our system is designed to send only 2 possible bits of information(A pulse on the left or right audio channel to signal a counterclockwise/-clockwise turn of the motor). With better pulse width modulation we would have more precise control of our motors which would provide a finer resolution in terms of frequency resolution on the guitar.

## References

[1] TUNE. *Guitar Tuning Image.* URL: http://www.guitar-book.com/index_htm_files/standard-guitar-tuning.png.

[2] Wikipedia. *Guitar tunings.* URL: https://en.wikipedia.org/wiki/Guitar_tunings.

[3] Mohanned Rawashdeh. *Instructables: BYJ48 Stepper Motor.* URL: http://www.instructables.com/id/BYJ48-Stepper-Motor/.

[4] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Third. New York: Pearson, 2009.

[5] Apple. *Accelerate Framework.* URL: https://developer.apple.com/library/tvos/documentation/Accelerate/Reference/AccelerateFWRef/index.html.

[6] EZAudio. *EZAudio Library.* URL: https://github.com/syedhali/EZAudio.

[7] WUSTL. *Wustl Logo.* URL: http://www.myfreecoursesonline.com/wp-content/uploads/2013/03/wustl-logo.jpg.

# Appendices

## Website

Our project website can be found athttp://wustlguitartuner.weebly.com/

## iOS Source Code

The source code for the iOS app is to large to include in this report. The code can be found at github.com/jawjay/Guitar_Tuning and is available for use under the MIT license.

## Arduino Source

```
/*
   BYJ48 Stepper motor code
   Connect :
   IN1 >> D8
   IN2 >> D9
   IN3 >> D10
   IN4 >> D11
   VCC ... 5V Prefer to use external 5V Source
   Gnd
  */
#define S1  6
#define S2 7
#define IN1  8
#define IN2  9
#define IN3  10
#define IN4  11
#define TS 6
int Steps = 0;
boolean Direction = true;// gre
unsigned long last_time;
unsigned long currentMillis ;
int steps_left = 4095;// steps per revolution, gear ration included
int steps_right = 100;
long time;
int prev_val_left;
int prev_val_right;
long count;
long countLeft;
long countRight;
int serBit;
void setup()
{
  Serial.begin(115200);
  //Serial.begin(9600);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(TS,OUTPUT);
  // delay(1000);
}
void loop()
{
  if (Serial.available()>0){
    serBit = int(Serial.read());
    if ( serBit>0 && serBit <=9){
    steps_left = int(pow(2,serBit));
      //digitalWrite(6,HIGH);
      while (steps_left > 0) {
      currentMillis = micros();
      if (currentMillis - last_time >= 1000) {
      stepper(1);
      time = time + micros() - last_time;
      last_time = micros();
      steps_left--;
        }
      }
  }
  if (serBit == 21){
    Direction = true;
    //clockwise
  }
  if(serBit == 22){
    Direction = false;
    //counterclockwise
  }
  }
}
void stepper(int xw) {
  for (int x = 0; x < xw; x++) {
    switch (Steps) {
      case 0:
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, HIGH);
        break;
      case 1:
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, HIGH);
        digitalWrite(IN4, HIGH);
        break;
      case 2:
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, HIGH);
        digitalWrite(IN4, LOW);
        break;
      case 3:
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, HIGH);
        digitalWrite(IN3, HIGH);
        digitalWrite(IN4, LOW);
        break;
      case 4:
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, HIGH);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, LOW);
        break;
      case 5:
        digitalWrite(IN1, HIGH);
        digitalWrite(IN2, HIGH);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, LOW);
        break;
      case 6:
        digitalWrite(IN1, HIGH);
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, LOW);
        break;
      case 7:
        digitalWrite(IN1, HIGH);
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, HIGH);
        break;
      default:
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, LOW);
        break;
    }
    SetDirection();
  }
}
void SetDirection() {
  if (Direction == 1) {
    Steps++;
  }
  if (Direction == 0) {
    Steps--;
  }
  if (Steps > 7) {
    Steps = 0;
  }
  if (Steps < 0) {
    Steps = 7;
  }
}
```

## Matlab Source Code

```
function [ stop ] = setGuitar( freq,s )
% Function to run until ideal frequency is heard through
   microphone.
% Send serial comands based on current frequencey

```

```
freqdes = freq;

rec = audiorecorder(44100,16,1);
lower = 21;%lower freq
higher  = 22; % higher freq
[b,a] = butter(3,[freqdes-20 freqdes+20]/(44100/2),'bandpass');
stop = 0;

while stop == 0

        [maxi,error] = getMaxF(rec,b,a,freqdes);

        if abs(error) < .5
           stop = 1;
            fprintf('DONE');
        elseif error> 0
                fwrite(s,lower);
                tmove = fliplr(dec2bin(80*error));
                top = size(tmove,2);

                for i = 1:1:top
                        if str2num(tmove(i))==1
                                fwrite(s,i);
                        end
                end

        else
                fwrite(s,higher);
                tmove = fliplr(dec2bin(-80*error));
                top = size(tmove,2);
                for i = top:-1:2
                        if str2num(tmove(i))==1
                                fwrite(s,i);
                        end
                end

        end

end

end
```

```
function [ maxi,error ] = getMaxF( rec,b,a,freqdes)
% Get current max frequency from mic input
% a,b define filter parameters
% freqdes is desired frequency

recordblocking(rec,5);
```

```
        y = getaudiodata(rec);
        ye = filter(b, a, y);


        Fse = 44100;
        Fs = Fse;             % Sampling frequency
        T = 1/Fs;              % Sampling period
        L = size(ye,1);              % Length of signal
        t = (0:L-1)*T;         % Time vector

        Y = fft(ye);

        P2 = abs(Y/L);
        P1 = P2(1:L/2+1);
        P1(2:end-1) = 2*P1(2:end-1);
        bfil = zeros(size(P1));
        bfil(round(freqdes*5)-15:round(freqdes*5)+15) = 1;
        Pn = P1.*bfil;
        plot(P1);
        f = Fs*(0:(L/2))/L;
        [MAX, ind] = max(Pn);
        maxi = f(ind)
        %+.2Hz each time
        error = f(ind) - freqdes;

end
```