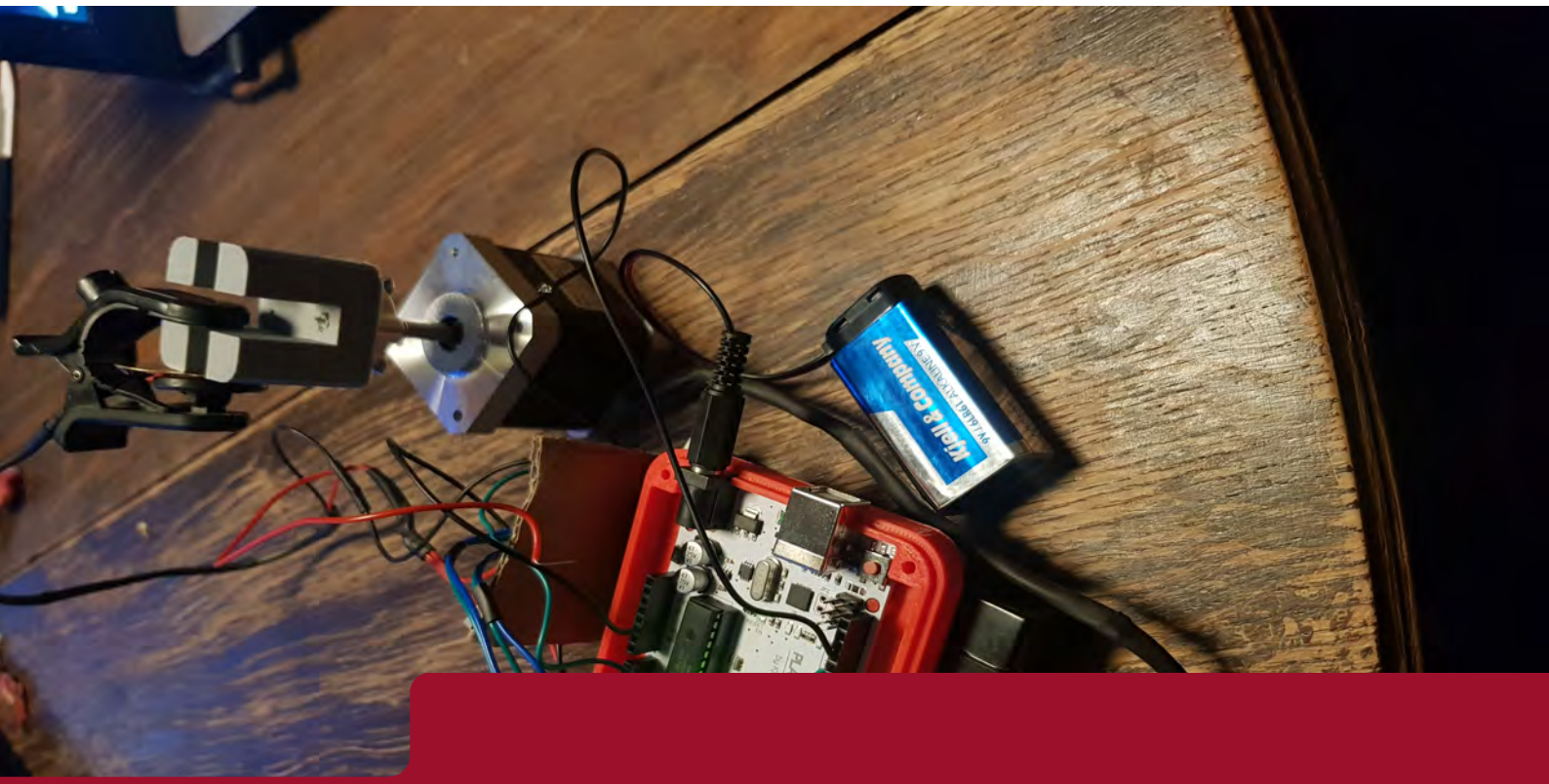
EXAMENSARBETE INOM MASKINTEKNIK,
GRUNDNIVÅ, 15 HP

STOCKHOLM, *SVERIGE 2021*

Piezoelectric Guitar Tuner

ALBIN BOESTAD

FABIAN RUDBERG



KTH
SKOLAN FÖR INDUSTRIELL TEKNIK OCH MANAGEMENT

Piezoelectric Guitar Tuner

Bachelor Thesis at ITM

ALBIN BOESTAD, FABIAN RUDBERG

Bachelor Thesis at ITM
Supervisor:Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2021:24

Abstract

This bachelor thesis in Mechatronics account for the process of constructing an automatic guitar tuner by means of a piezo-electric sensor, a stepper motor and Arduino-based control. The E_4 - string on an acoustic guitar was used as a proxy for tuning any other possible guitar string. The accuracy and tuning-speed of the construction was examined through physical experimentation. Accuracy was measured in terms of the average distance from a piezo-calibrated frequency value. The tuning-speed was appraised by counting the number of times a guitar string had to be plucked before the motor stopped within an acceptable tuning interval. The automatic guitar tuner were able to reliably get the E_4 - string in tune by plucking it once within an interval of ± 2 Hz and $+3.8$ cents and -5.1 cents from the theoretical value. The average error was -3.4 cents from the targeted value.

Keywords: Mechatronics Tuner Guitar Piezo Arduino

Referat

Piezoelektrisk Gitarrstämmare

I följande kandidatexamensarbete kontrueras en automatisk gitarrstämmare med hjälp av en piezosensor, en stegmotor och en Arduino-mikrokontroller. E_4-strängen på en akustisk gitarr användes som substitut för hur stämningsproceduren skulle kunna fungera för vilken annan gitarrsträng som helst. Noggrannheten samt stämningshastigheten undersöktes genom experiment. Genomsnittet av frekvensskillnaderna mellan de piezo-kalibrerade avläsningsvärdena och E_4-strängens värden definierade måttet på noggrannhet. Hastigheten på strängstämningen beräknades i form av hur många gånger en sträng behövdes slås an innan strängen var inom ett godkänt intervall. Den automatiska gitarrstämmaren visade sig pålitiligt kunna stämma E_4-strängen på ett försök inom ett noggrannhetsintervall på $\pm 2Hz$ från det teoretiska värdet. Stämmaren kunde stämma inom $+3.4$ cents och -5.1 cents samt var var i genomsnitt -3.4 cents i från det teoretiska värdet.

Nyckelord: Mekatronik Stämmare Guitar Piezo Arduino

Acknowledgements

We want to thank our supervisor Nihad Subasic for his unfaltering engagement during the course curriculum. Particularly for his immediate and straightforward critic during the various challenges that came up during the project. We would also like to thank course assistant Amir Avdic, who tirelessly and enthusiastically assisted us and other groups during the lab sessions.

Albin Boestad and Fabian Rudberg.
KTH Royal Institute of Technology, Stockholm, May 2021

Contents

List of Figures

List of Tables

List of Abbreviations

ADC - Analog-to-Digital Converter

CFT - Continuous Fourier Transform

DC - Direct Current

DFT - Discrete Fourier Transform

FFT - Fast Fourier Transform

FT - Fourier Transform

GND - Ground

I/O - Input/Output

KTH - Kungliga Tekniska Högskolan (Royal Institute of Technology)

MATLAB - Matrix Labratory

PID - Proportional Integral Derivative

PWM - Pulse Width Modulation

USB - Universal Serial Bus

Chapter 1

Introduction

In a busy life, all types of help to make everyday tasks easier are convenient. Therefore there is a growing market for automatizing monotone tasks. An acoustic guitar is usually tuned by the user manually turning the tuning keys on the guitar while comparing the sound it makes with a reference. The reference could be another instrument, an electric guitar tuner or a tuning fork. An automatic guitar tuner would only need the user to pluck the strings. This report presents how to create a device that will adjust, in this case, acoustic guitars tuning keys based on the vibrations created by the strings of the guitar.

1.1 Background

An acoustic guitar is a type of string instrument. The instrument produces sound from the vibrations of the strings. The most rudimentary way to tune a guitar is by comparing a string's sound with a fixed sound made by a tuning fork. The string and the tuning fork has the same frequency when they sound the same. Some tuners detect the string's frequency and let the user know if that tone is sharp or flat. The user can then tighten or loosen the tuning key to adjust the tone (see Appendix C). There are two standard procedures when using electrical tuners in transducing actual frequency values generated by the plucking of a string. The first method is to pick up the air-traveling sound waves with a microphone. The second method is to directly, from the guitar body itself, pick up the vibrations from the strings traveling through the guitar's body.

1.2 Purpose

The project aims to survey the use of a piezoelectric sensor and a controller to tune an acoustic guitar through an electric motor automatically. Furthermore, within a certain benchmark, the aim is to optimize the adaptation and the accuracy of the tuning process.

To conclude, the following three research questions are to be answered:

- How can a piezoelectric sensor and a controller be used together with an electric motor to tune an acoustic guitar automatically?
- What is the accuracy of the guitar tuner when tuning one-half step above or below the desired frequency value?
- How few times does a string need to be plucked before it is 'in tune'?

1.3 Scope

The project presented in this thesis covers a Bachelor thesis in Mechatronics at the Royal Institute of Technology (KTH). A similar project has previously been published as a Bachelor thesis at the Mechatronics department at KTH; they focused on tuning an electric guitar through a connected cable [1]. In this project, the tuner has been designed for tuning an acoustic guitar through a piezo-sensor. The project corresponds to 15 ECTS-points and is taking place over a time period of approximately four months. It has a budget of 1000 SEK as well as available materials in the Mechatronics lab at KTH.

1.4 Method

The first objective of the project was a literature study. Afterward, the first set of components was gathered. The first components were a stepper motor with a driver, a piezo element and an Arduino Uno board. With the initials test, we could see that the piezo could be used to get input signals. The motor was tested to rotate the tuning keys of the guitar. After these steps, a model for the tuning key grips was created using *Solid Edge 2019*. The model was 3D printed using *Cura Ultimaker*. The next step was starting to work on the signal analysis from the piezo element. To get the frequency, Fast Fourier Transforms was used. When this was successful, everything was connected to one circuit. Before regulating the steps for the motor, the readout of the piezo and FFT needed to be synchronized. The synchronization involved reading what the piezo got as the frequency when the string was in tune. The string was tuned to E4 and then ten readouts from the piezo were documented and an average frequency was set as the target frequency. The next step was to regulate the motor's rotation depending on how much in tune the string was. This was done by experiments where proper constants for the controller were tested. When the tuner could tune correctly on one attempt, tests of accuracy were conducted. Thirty measurements were made where the strings starting frequency was between $\pm$ a half step from E_4.

1.4.1 Calibration

The tuner was calibrated to improve the accuracy of the tuning. Values for the sampling frequency, the constant K_p, tuning interval and the motor's speed was chosen. The piezo element was synchronized with a reference tuner. Specifically the KORG Pitchclip 2 Clip-on tuner, a commercially available electronic tuner, see Appendix O for tuner specifications.

1.4.2 Sampling Frequency

The sampling frequency was chosen to a sample rate twice the maximum frequency of the highest signal according to Nyquist [2]. The highest signal was selected to be one-half step above the targeted frequency value. A guitar string is rarely by accident more out of tune than one-half step and a sampling frequency too far away from the targeted frequency affected the reading from the piezo.

1.4.3 Piezo Reading Calibration

The piezo output reading was synchronized with a commercial tuner, Pitchclip 2 Clip-on tuner that gets clip unto the guitar. According to the tuner manufacture, the tuner had an accuracy of ± 1 cent, Appendix O. One cent means that the tuner has an accuracy of one 100th of a half step. The E_4 string was manually tuned until the reference tuner said it was in tune. The string was plucked ten times, and the piezo reading was documented, see Appendix H.

1.4.4 Acceptable Tuning Interval

Finding a interval in which the string could be regarded as tuned was found by experimenting with different intervals in tandem using the reference tuner (Appendix O) as a guide for accuracy.

1.4.5 K_p Calibration

In the Theory Section 2.5, a detailed description about the proportional constant K_p is introduced. To calibrate the value of the factor K_P a series of experiments were conducted. To start with a low value to work from, K_P was preliminary set to $K_P = 3$. After that, the E_4 string was tuned to be a half step below E_4, meaning it was tuned in D#, see Section 2.1.1 and Table 2.1. The motor was then attached to the tuning key of the E_4 string and after plucking the string, the motor rotated a certain amount of steps which were documented. Then the string was plucked again and the motor rotated again. This procedure was repeated until the string was in tune. Then the value of K_p was adjusted and the same experiment was conducted. The same experiment was also done by tuning the string one half-step above E_4, i.e. F_4.

1.4.6 Accuracy

To measure accuracy a series of tests was conducted where the tuner was tuned ten times from D#, ten times from F_4 and ten times from a range of frequencies between D# and F_4.

1.4.7 Speed

The speed of the guitar tuner is defined as the number of times the string needs to be plucked before it is in tune. If the number of trials in an attempt is two, the first pluck results in the motor to rotate and the second pluck of the string checks that the string is in tune.

Chapter 2

Theory

The following chapter consists of the relevant theory to the thesis.

2.1 Musical intervals

All music is built on intervals between different notes and these notes are measured in frequencies. The important music theory is the musical alphabet, octaves and harmonics.

2.1.1 The musical alphabet

There are multiple ways of describing music. A musical interval is simply the distance between two frequencies. Specifically related to this thesis, frequency is the measurement of the number of repeating cycles per second that a guitar string makes when plucked. The first seven letters of the Latin alphabet A, B, C, D, E, F and G, represent the names of the characteristic sounds that the human ear perceives from these frequencies. [3]

The smallest interval within western music is called a half-step. Between all notes (A, B, C, D, E, F, G) except in the middle of B, C and E, F respectively, going one half-step down in pitch is denoted by adding the flat symbol b; and going one half-step up in pitch is denoted by adding the ”sharp”-symbol, #. These symbols are called 'accidentals'. For example, ”Bb”, pronounced 'b flat', refers to the note with the frequency corresponding to one half-step below the note B. ”B#”, pronounced 'b sharp', refers to the note with the frequency corresponding to one half-step above B [4]. These ”frequency-interval-building-blocks” of western music theory can be more directly communicated by using the familiar image of a piano keyboard, see figure 2.1. Observe that”C#” is the same as ”Db”, and ”D#” is the same as ”Eb”; and that there are no accidentals between E,F and B, C.

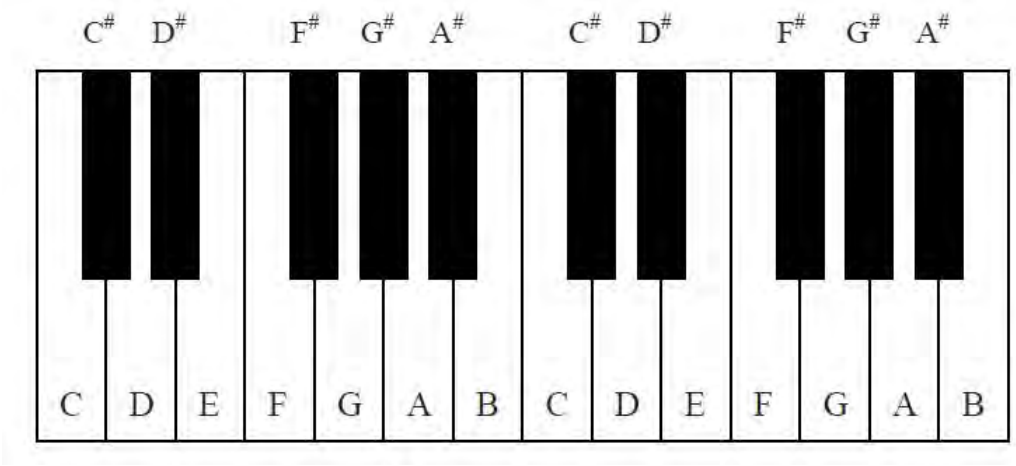


Figure 2.1. Piano Keyboard with note names [3].

2.1.2 Octaves

The twelve musical notes are periodic, meaning that they are repeating themselves. The note twelve half-steps above the note C is also called C. This interval is called an octave. Going up one octave in pitch doubles the frequency of the lower octave. A number is added to the note's letter to distinguish between these C's (or any other note). A lower number means a lower octave. C_1 is half the frequency of C_2 and C_2 is half the frequency of C_3. Furthermore, the notation C_1, C_2, C_3, etc is called "Scientific pitch notation" [5].

Another common type of unit describing pitch intervals is called a cent. One cent is one hundredth of a half-tone interval. This unit is often used in electronic guitar tuners and by musicians. [3]

For this thesis we are only concerned with the open strings in standard tuning, they are from the lowest string: E_2, A_2, D_3, G_3, B_3, E_4. These are marked red in Table 2.1. For reference, the human hearing range spans roughly from E_0 to E_{10}, the lowest note on a piano is A_0 and the highest note on a piano is C_8 [3], see Table 2.1. Also, notice how one octave up doubles the frequency from the previous octave.

Table 2.1. Frequency of guitar strings within the human hearing range (Hz). The open strings of a guitar in open tuning are marked in red. The empty cells are referring to sharps/flats but are omitted to make the table more tolerable to look at [3].

E_0	20.602	E_3	164.81	E_6	1318.5	E_9	10548
F_0	21.827	F_3	174.61	F_6	1396.9	F_9	11175
	23.125		185.00		1480.0		11840
G_0	24.500	G_3	196.00	G_6	1568.0	G_9	12544
	25.957		207.65		1661.2		13290
A_0	27.500	A_3	220.00	A_6	1760.0	A_9	14080
	29.135		233.08		1864.7		14917
B_0	30.868	B_3	246.94	B_6	1975.5	B_9	15804
C_1	32.703	C_4	261.63	C_7	2093.0	C_{10}	16744
	34.648		277.18		2217.5		17740
D_1	36.708	D_4	239.66	D_7	2349.3	D_{10}	18795
	38.891		311.13		2489.0		19912
E_1	41.203	E_4	329.63	E_7	2637.0	E_{10}	21096
F_1	43.654	F_4	349.23	F_7	2793.8		
	46.249		369.99		2960.0		
G_1	48.999	G_4	392.00	G_7	3136.0		
	51.913		415.30		3322.4		
A_1	55.000	A_4	440.00	A_7	3520.0		
	58.270		466.16		3729.3		
B_1	61.735	B_4	493.88	B_7	3951.1		
C_2	65.406	C_5	523.25	C_8	4186.0		
	69.296		554.37		4434.9		
D_2	73.416	D_5	587.33	D_8	4698.6		
	77.782		622.25		4978.0		
E_2	82.407	E_5	659.26	E_8	5274.0		
F_2	87.307	F_5	698.46	F_8	5587.7		
	92.499		739.99		5919.9		
G_2	97.999	G_5	783.99	G_8	6271.9		
	103.83		830.61		6644.9		
A_2	110.00	A_5	880.00	A_8	7040.0		
	116.54		932.33		7458.6		
B_2	123.47	B_5	987.77	B_8	7902.1		
C_3	130.81	C_6	1046.5	C_9	8372.0		
	138.59		1108.7		8869.8		
D_3	146.83	D_6	1174.7	D_9	9397.3		
	155.56		1244.5		9956.0		

2.1.3 Harmonics

The sound produced from a vibrating guitar string does not only consist of one unique frequency element. What is perceived by the human ear as sound is a cluster of sinusoidal waves. The group made up of these waves is called the natural frequencies and the lowest frequency wave among those is called the fundamental frequency. The natural frequencies of a string can be calculated according to equation (2.1) where n is an integer (multiple of the fundamental frequency), L is length, T is tension and d is the density of the string. The fundamental frequency is the number of wave cycles per second that humans experience as the loudest sound. The other waves are called harmonics and those frequencies consist of integer-multiples of the fundamental frequency [3].

$$f(n) = \frac{n\pi}{L}\sqrt{\frac{T}{d}} \tag{2.1}$$

Consequently, tuning a guitar is partly a matter of identifying the fundamental frequency among the natural frequencies. Harmonics resulting from the vibrating guitar string can therefore be viewed as undesirable, i.e. noise, with regard to the aim of accurately tuning a guitar string.

2.1.4 Tuning accuracy

Cents is a measurement of the accuracy of a tune. A cents corresponds to 1 hundredth of a semitones. Meaning that between A_2 and $A_2\#$ there is 100 cents. Human perception can identifier a differences of ± 5 cents.

2.2 The Fast Fourier Transform

The Fast Fourier Transform is an effective algorithm of the discrete Fourier transform. To elucidate about the workings of the FFT, definitions of the underlying concepts are appropriately delineated.

The natural frequencies, see Section 2.1.3, can be mathematically expressed with the Fourier transform (FT). The concept behind The FT is that any periodic wave can be described as a summation of sinusoidal functions [6]. Using the FT, one can transform these sinusoidal functions from the time domain to the frequency domain. The continuous Fourier transform (CFT) is used for signals expressed as functions of continuous time variables. To take advantage of the CFT the function of the signal itself must be known so as to be able to integrate over it.

Often in practice, the function of the signal is unknown, but experimental data samples at discrete times have been recorded and are available for analysis. In that case, the discrete Fourier transform (DFT) can be utilized. When calculating the FT with the DFT the finite collection of data samples has a computational complexity of O(n^2). The FFT is an algorithm that reduces this complexity to O(NlogN) [7].

2.3 Piezoelectric effect

The word piezo is derived from the Greek "piezein", which means to squeeze [8]. In the late 1800's Pierre and Jacques Curie discovered that some naturally occurring crystals, like quartz, can generate an electrical charge when pressure is applied to them. The term piezoelectricity then means "pressure-driven electricity". When a mechanical force is applied to a piezoelectric substance, the electrical charges within the molecule of that substance reorient. This reorientation causes a differential in the surface charge-distribution, which results in a voltage [9], see figure 2.2.

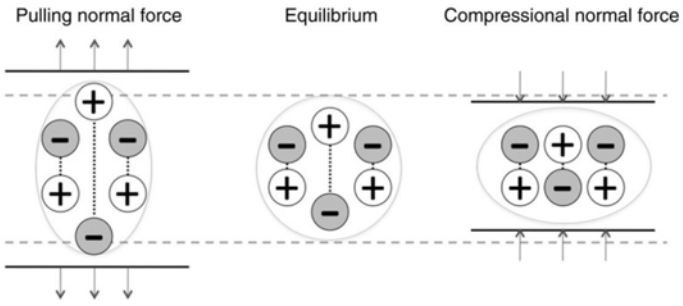


Figure 2.2. Illustration of the piezoelectricity effect [9].

2.4 Arduino Uno

Arduino Uno is a microcontroller developed by the open-sourced company Arduino. Arduino Uno is based on the single-chip microcontroller ATmega328. The controller consist of a USB port, power jack, a reset button, 14 input/output pins, 6 analog pins. The USB port can be used to power the Arduino as well as uploading code to the Arduino. The power jack can power the Arduino by a battery or from an AC-to-DC adapter. The input/output pins are labeled 0 to 13 on the Arduino board. Out of those 14 pins, six of them can be used as PWM. They are labeled with a tilde sign ~ next to their number. The analog pins are labeled A0 - A5 [10]. See figure 2.3 for an overview of the pins. PWM stands for Puls-Width Modulation. The PWM modulates the duty cycle of a square wave to imitate an analog signal level [11].

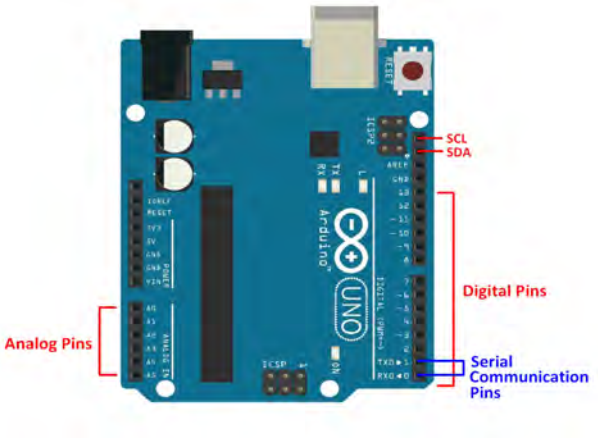


Figure 2.3. Overview of a Arduino UNO pins positioning [12].

2.5 P-Controller

To control a system a controller is needed. A system is something changes depending on the input to the system. One type of controller is a P-controller. It stands for a proportional controller. The proportional controller takes an input signal, multiply the input with a factor and sends the new signal as input to the system. The output of the controller is u(t). The input to the controller is the error, e(t). e(t) is calculated according to (2.2).

$$e(t) = r(t) - y(t) \tag{2.2}$$

r(t) is the reference state, i.e., the state we want the system to be. y(t) represents where the system currently is. If y(t) were equal to r(t), the error would be zero. Thereby would the input to the system also be zero and the system would not change. The following function calculates the signal u(t), i.e the signal that goes to the system, for a P-controller:

$$u(t) = K_p e(t) \tag{2.3}$$

K_p is a factor that is multiplied with the error. The higher K_p is, the faster the system response. However, this can lead to instability. Experiments can decide the value of K_P [2].

2.6 Stepper Motor

A stepper motor is a type of DC motor. The motor divides a complete revolution into steps and can therefore make precise movements in the form of steps. The motor consists mainly of two parts: a stator and a rotor. The stator consists of wounded coils that are paired. The coils are distributed evenly around the rotor. Each pair of coils are facing each other with the rotor between them. Figure 2.4 is an overview of a stepper motor. The rotor has a permanent magnet in it, and the rotor is magnetized in the axial direction. In a stepper motor, only one pair of coils is active at a time. When one pair of coils is active, they induce a magnetic pull that moves the rotor a step. After one step, the following pair of coils are activated and the rotor rotates another step [13].

There are two types of stepper motor, Unipolar and Bipolar. The unipolar operates phase with a winding and a center tap, see figure 2.5. The center tap allows a Unipolar stepper motor to reverse without the need to change the current. A Bipolar stepper motor has a winding per phase without a center tap, see figure 2.6. A Bipolar needs a H-bridge to be able to reverse [15].

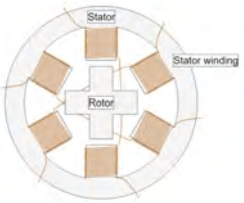


Figure 2.4. Overview of a stepper motor [14].

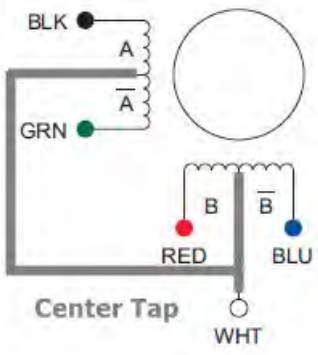


Figure 2.5. Wiring setup for a Unipolar stepper motor [16]

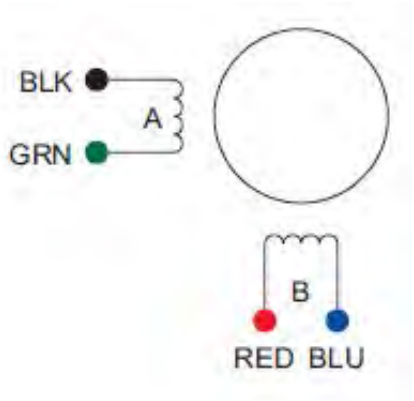


Figure 2.6. Wiring setup for Bipolar stepper motor [16]

2.6.1 Stepper Motor Driver

A stepper motor driver is a circuit that controls a stepper motor. The driver controls that the motor takes the right amount of steps. The driver also controls which way the motor steps [13].

2.6.2 H-bridge

An H-bridge is a type of integrated drive-circuit module. The H-bridge arrangement makes it possible to change the polarity of the voltage applied. This allows for running a DC stepper motor both forward and backward [17].

Chapter 3

Demonstration

In the following sections, the construction and its constituent parts are outlined.

3.1 Problem Formulation

The guitar tuner consists of two sections: the frequency detection part and the control of the motor to adjust the tuning keys on the guitar. From the piezo element, a signal is transported into the Arduino. That signal is in the time domain. To get the frequency of that signal the signal is transformed using Fast Fourier Transform. The transformation turns the signal from the time domain to the frequency domain.

3.2 Hardware and Electronics

The following sections cover all the hardware and electronics used in the project.

3.2.1 Piezoelement

To pick up the vibrations from the guitar, a piezo element is connected to the headstock of the guitar, see Appendix C. The piezo used in the project is a 7BB-20-6 6.3 KHz Piezo-element by Murata. The piezo is connect by soldering two wires to the element, see figure 3.1.

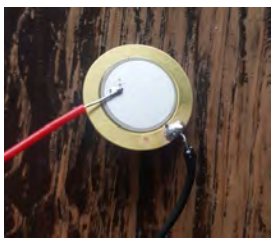


Figure 3.1. Piezo-element soldered with two wires. Picture taken by authors.

3.2.2 Arduino Uno

To control the system a micro controller is needed. The controller used will be an Arduino Uno (Playknoelogy Uno Rev. 3 model). For datasheet see reference [10].

3.2.3 Stepper Motor NEMA 17

To move the tuning key a motor is needed. A stepper motor is a dc motor that rotates a certain amount of steps, see subsection 2.6. The motor chosen in this project is a Nema 17 bipolar stepper motor. The datasheet can be found in the Apendix D. A bipolar stepper motor needs a H-bridge to be able rotate in both directions. It runs on 12 Volt. To turn the tuning keys a grip was designed in *Solid Edge* and 3D printed using an *Ultimaker 2* 3D print machine. The blueprint can be found in Appendix E. The grip connects to the shaft of the motor by an axle hub, see figure 3.2. The motor can be seen in figure 3.3 where it is assembled with the grip and the axle hub.

Figure 3.2. Actobotics axle hub [18].

Figure 3.3. Nema17 stepper motor. Picture taken by authors.

3.2.4 Stepper motor driver Adafruit

To control the stepper motor a stepper motor driver is used. The driver chosen in this project is a Adafruit Stepper motor, see figure 3.4. The driver can power a 12 Volt stepper motor, and has a built in H-bridge. For datasheet and assembly description see Appendix K.

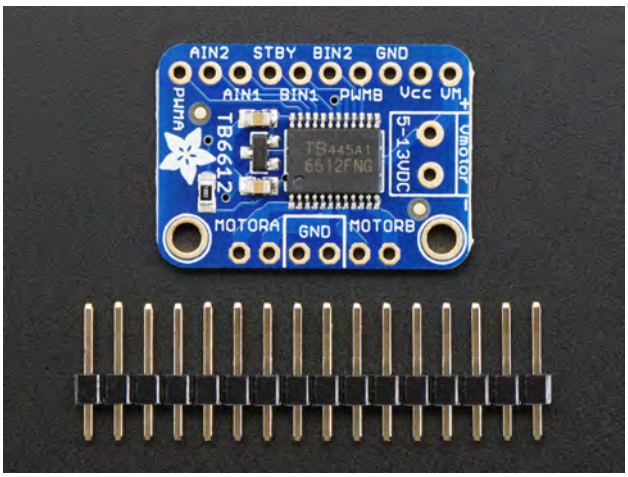


Figure 3.4. Picture of Adafruit Stepper Motor Driver [19].

3.2.5 Connection

The piezo element is attached to a plastic clamp to hold the piezo onto the guitar head stock, see Appendix C. This clamp is the disassembled clip-on part from an old guitar KORG AW2G clip-on chromatic guitar tuner, see Appendix N. The piezo has two wires soldered unto it. One of them goes to the Arduino Unos 5 volt output pin. The other splits to one that goes through a 10M Ohm resistor and then ground on the Arduino. The other goes to the Arduino Uno's analog pin A0. From the motor driver pins Vcc, PWMA and PWMB connect to the Arduino 5 volt output pin. GND on the driver connects to one of the Arduinos ground pins. Pins AIN1 and AIN2 connects to pins 8 and 9 on the Arduino. Pins BIN1 and BIN2 connect to pins 10 and 11. A 12 volt battery connects to the driver boards VMOTOR pins with a 100 μF capacitor between the positive and negative poles of the battery. The capacitor protects the driver board from potential Volt spikes. The battery powers only the motor. A brief overview of the connection can be seen in figure 3.5 below.

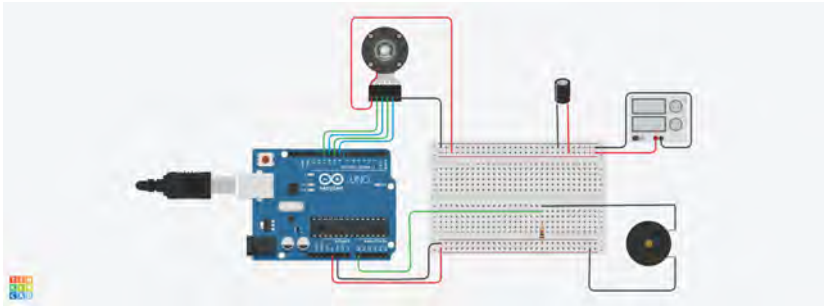


Figure 3.5. This is an overview of the connections, it was made using *Tinkercad*.

3.3 Software

The Arduino was programmed in Arduinos own software, Arduino IDE 1.8.13. The code can be found in Appendix F. The code is built around two sections. The signal analysed with Fast Fourier Transform and the control of the stepper motor. The signal input and Fast Fourier Transform part of the code is based on the code Audio Frequency Detector by Clyde A. Lettsome [20]. The Audio Frequency Detector uses arduinoFFT-library [21]. The control of the stepper motor is based on Arduinos built-in stepper library [22]. The control is a P-regulator. The larger the error is, the higher the number of steps the stepper motor takes. A flow-chart of the program is shown in figure 3.6.

3.4 Frequency detection

To know what tone the guitar string was currently tuned at, a KORG Pitchclip 2, clip-on guitar tuner was used, see Appendix O. The clip-on Guitar tuner has as an accuracy of +-1 cent. Cent is an audio accuracy measurement. 1 cent approximates one hundreth of a halfstep, as described in section 2.1.2.

When a string is plucked, the vibrations gets detected by the piezo element which is clamped on to the guitar head stock. The Arduino takes the signal as an input from Analog A0. Samples of the analog input are stored as elements in an array as a function of time. The FFT-function transforms the array from the time-domain to the frequency domain. The constituent parts of the signal as a function of time are thereby approximated and re-represented as magnitude and phase. The sampling frequency is a measure of the number of samples per second (Hz) used in the FFT. Depending on the sampling frequency, the approximation of the frequencies will vary and thereby influence how accurate the tuning becomes. The sampling frequency is set to double the highest frequency that will be detected. For a guitar string tuned to its base setting, a sampling frequency set to double of one half step above is sufficient for accurate detection of the frequency.

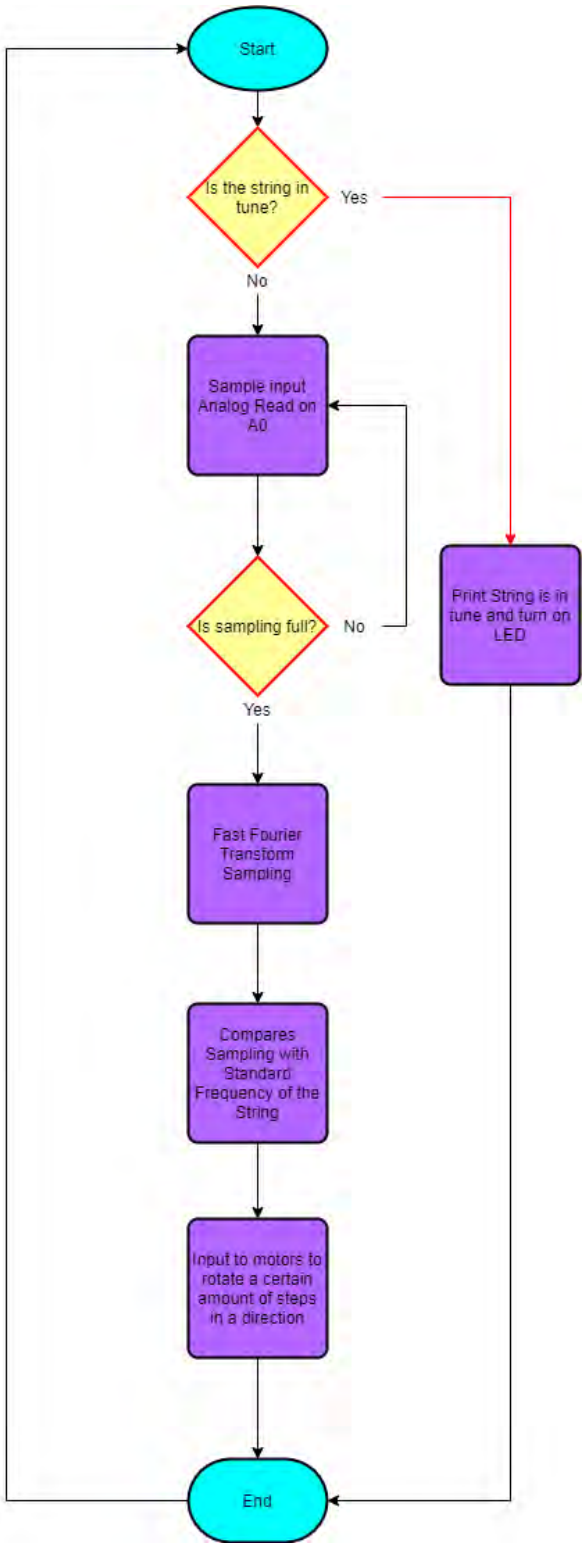


Figure 3.6. Flow-chart of the program. The figure was created with *draw.io*.

Chapter 4

Results

The following chapter covers the results from the calibration of the proportional controller, the piezo frequency readings and the experiments that determine how reliable the tuner is. The results and the calibration in this chapter are based on tuning the E_4 string of a steel-string acoustic guitar.

4.1 Calibration

Results from the calibrations described in Section 1.4 are presented in the following sections.

4.1.1 Sampling Frequency

Experiments confirmed that having the sampling frequency to be double the frequncy of a half step above E_4 was working when tuning from different frequencies within $\pm$ a semitone around E_4.

4.1.2 Piezo Reading Calibration

The result can be seen in figure 4.1. The average output of the piezo for the E_4 tuning was 333.54 Hz.

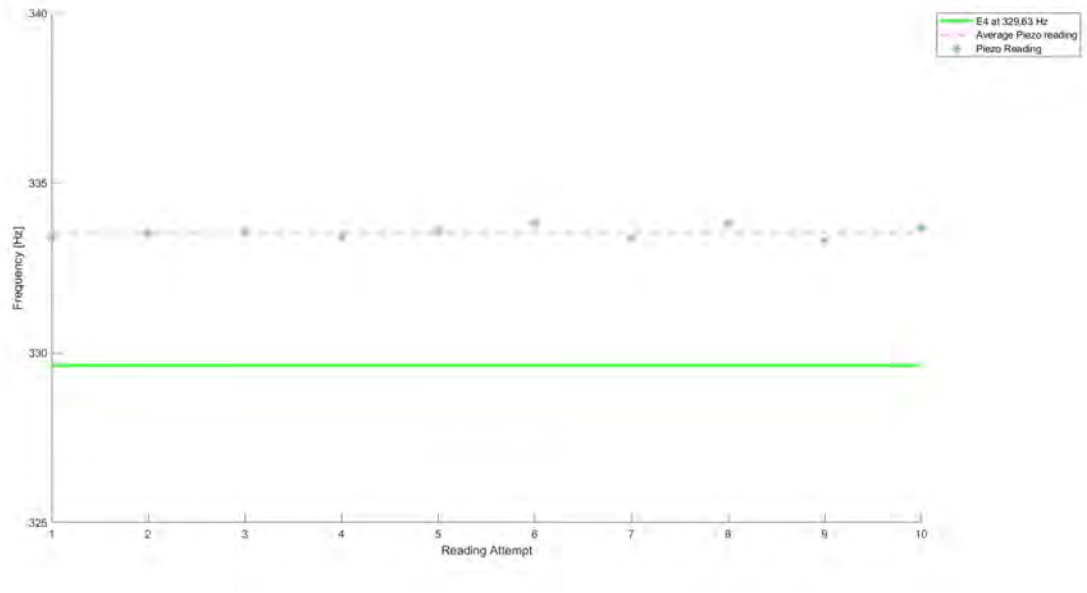


Figure 4.1. Readings of the Piezo when a string is tuned to E_4, graf made using *MATHWORKS* MATLAB 2020b.

4.1.3 Acceptable Tuning Interval

Within a specific interval of the calibrated frequency value of 333.54 Hz, the string was determined to be in tune. The size of this interval affected the trial numbers in our experiments, see Appendix J. A huge interval approved values which we deemed to be too inaccurate. A tiny interval missed acceptable values resulting in an unreasonable amount of trials and equivalently a much longer time to get the guitar string in tune. Experimenting with different intervals in tandem using the reference tuner (Appendix O) as a guide for correct tuning, an acceptable tuning interval of ± 2 Hz was established as good enough.

4.1.4 K_p Calibration

The results of the experiments can be found in Appendix I. These experiments concluded that the following two values of K_p were to be used. $K_p = 4.8$ was used when the frequencies were below E_4 and $K_p = 6.8$ if the frequencies were above E_4.

4.2 Research findings

Using the calibrated values presented above, the following experiment was conducted to answer the research questions in this paper.

4.2.1 Accuracy

From figure 4.2, 4.3 and 4.4 the tuning process can be observed. In figure 4.5 the error from the 30 attempts are shown in Hz and in figure 4.6 the error is in cents. The average error was 0.6 Hz from the target frequency. The accuracy of the tuner was within the interval -5.14 cents and +3.76 cents. The negative value means below the frequency of E_4 and the positive value means above. Notice that the reference tuner used for calibration is specified to have an accuracy of ± 1 cent. One cent is one hundredth of a semitone.

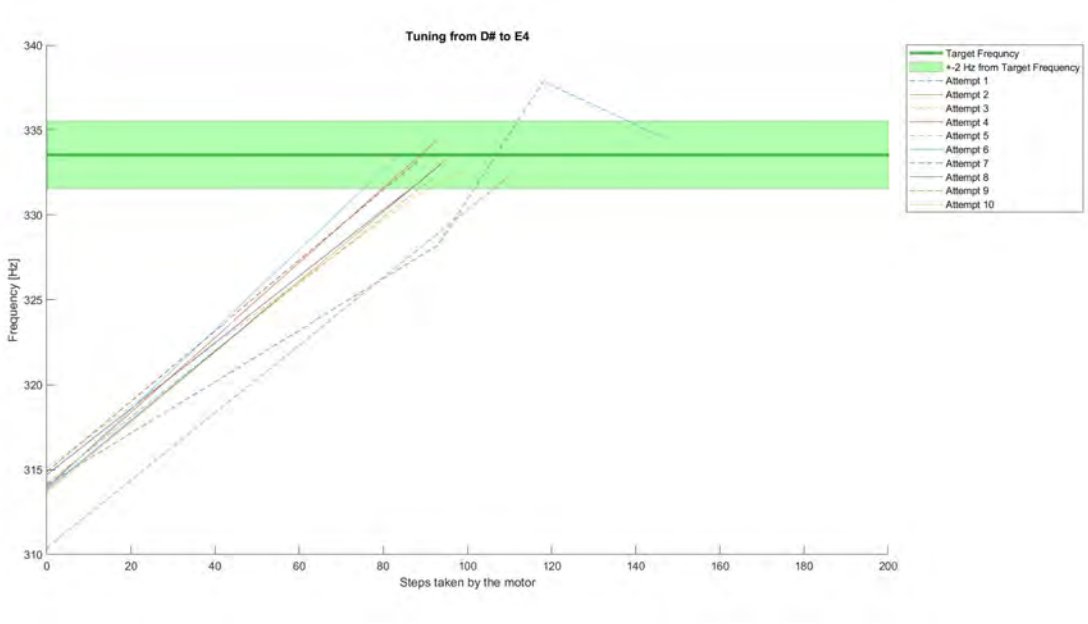


Figure 4.2. Tuning from D# to E_4, graf made using *MATHWORKS* MATLAB R2020b.

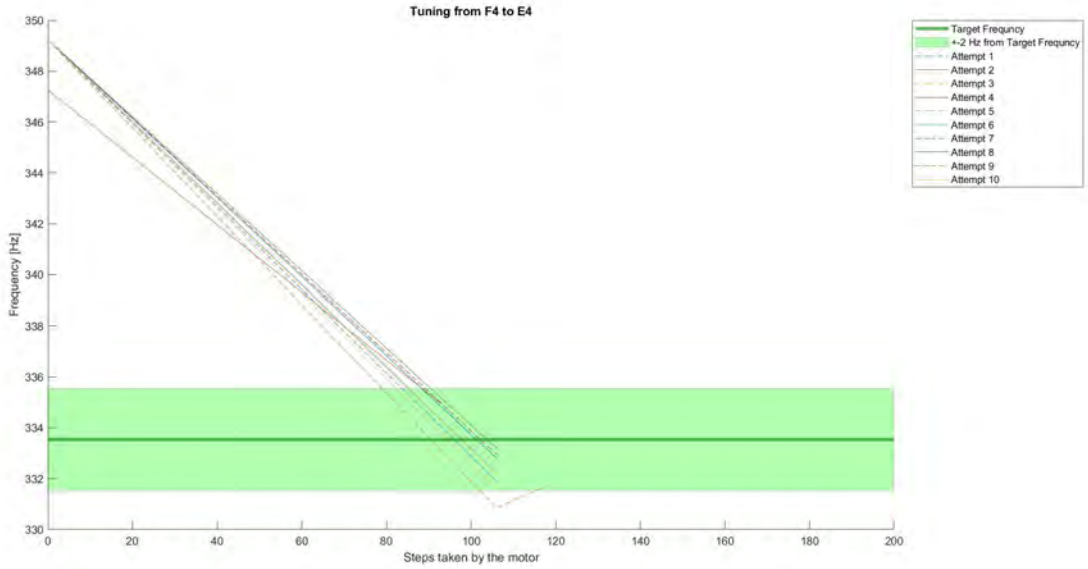


Figure 4.3. Tuning from F_4 to E_4, graf made using *MATHWORKS* MATLAB R2020b.

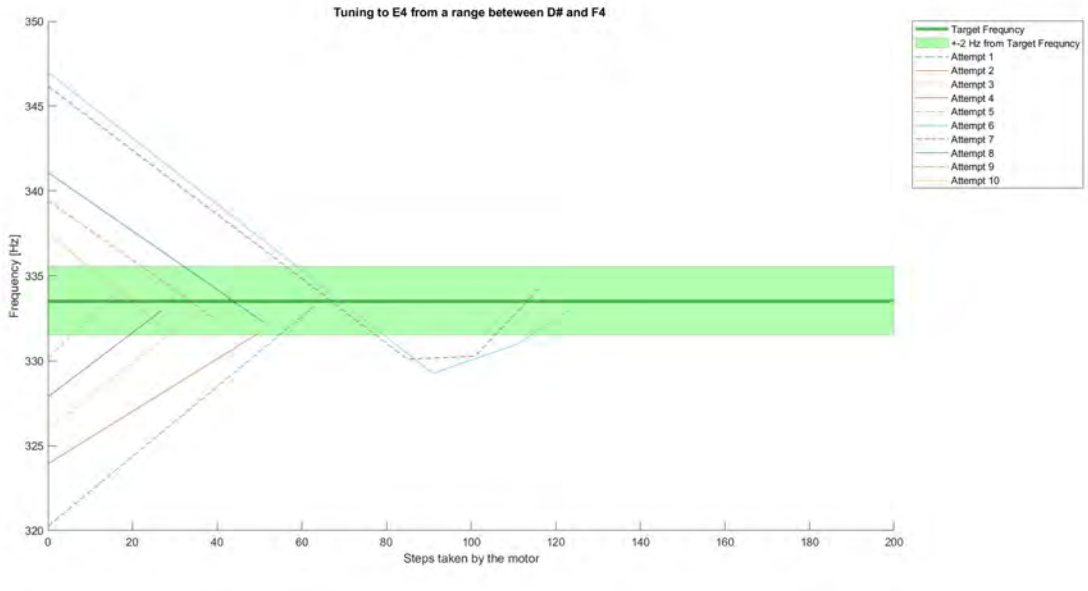


Figure 4.4. Tuning to E_4 from a range of starting point betwene D# and F_4, graf made using *MATHWORKS* MATLAB R2020b.

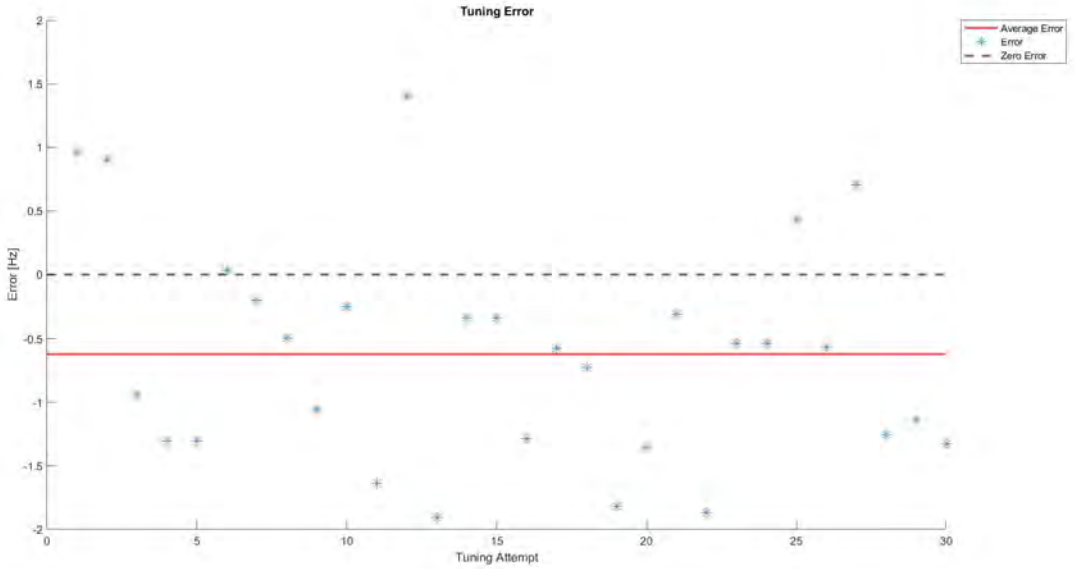


Figure 4.5. Graf over the error of 30 tuning attempts as well as the avrage error, graf made using *MATHWORKS* MATLAB R2020b.

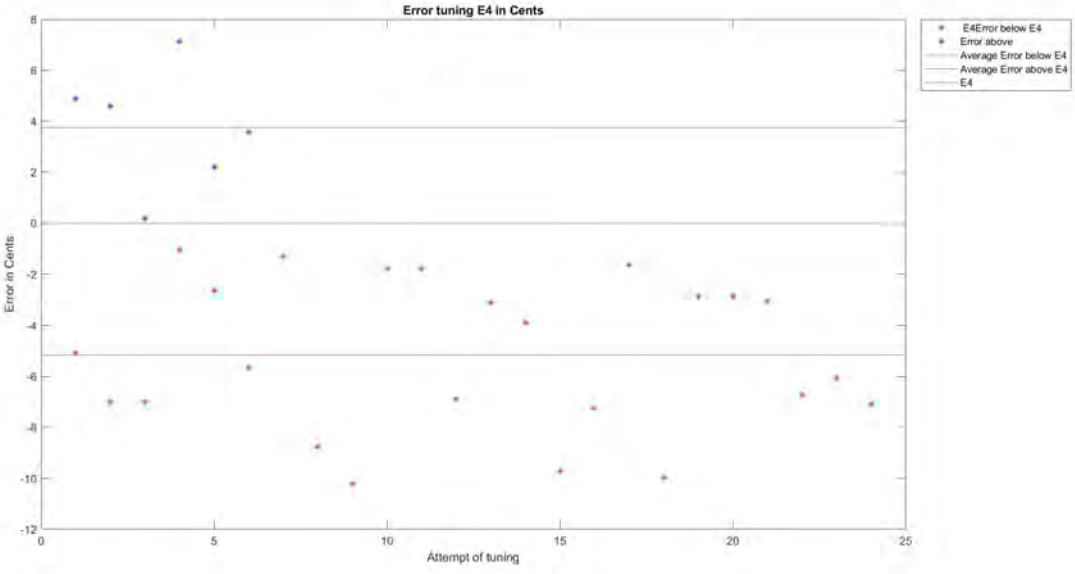


Figure 4.6. Error for the tune represented in cents, graf made using *MATHWORKS* MATLAB R2020b.

4.2.2 Speed

Figure 4.7 shows the result of how many times the string needed to be plucked in 30 tuning attempts.

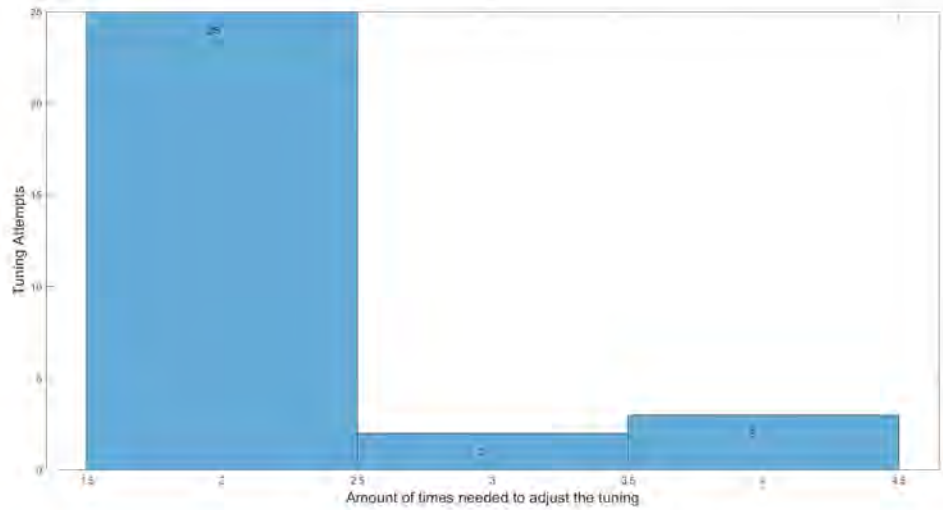


Figure 4.7. The number of times the string was plucked before it is was in tune. In 25 attempts out of 30, the string was in tune after plucking the string twice. That means that the string was turned to the right frequency after just one iteration of the motor rotating. After plucking the string again, the computer program recognized the frequency to be within the acceptable tuning interval, graf made using *MATHWORKS* MATLAB 2020b.

Chapter 5

Discussion and Conclusion

5.1 Discussion

During the initial tests of the piezoelectric guitar tuner, the end result of the tuning always ended up at a frequency value below that given by the clip-on reference tuner.

To compensate, the tuner was calibrated so that the frequency detected by the tuner corresponded to the clip-on tuners response of the frequency. The result became more accurate towards being actually in tune after the calibration.

One significant part of the construction was the choice of motor. At first, a 5 Volt stepper was used. The motor could turn the tuning keys of the guitar however when the motor had to take few steps, the motor could not manage to create enough torque to turn the guitar keys. Therefore a high torque stepper motor, Nema17, was used instead. This motor performed significantly better. But even this motor can have difficulty turning the guitar key if it only takes small steps. We believe this occurs due tot the fact that it is a stepper motor. With a regular DC motor there would be a constant torque when active. With a stepper motor the amount of steps are of importance, and the motor probably executed the given amount of steps before the torque could turn the key. This results in lowered accuracy of the tuner.

For the project the E_4 was used as the proxy for the guitar strings. The E_4 tuning key had the least resistance when turning with the guitar in this project. It should be noted that the guitar used in this project had quite the stiff tuning keys overall. Changing the string to a lighter type of string did have an unrecognizable effect on how stiff the tuning key became. However, when restringing the guitar, turning the tuning key so that it was as loose as possible before restringing the guitar did affect the stiffness of the key.

The sampling frequency for the FFT proved to play a major role in detecting the

frequency of the guitar string. If the Sampling frequency is set too low, the FFT will not comprehend frequency above half the sampling frequency resulting in a minimal readout. If it is set too high, the FFT will have difficulty reading out a frequency far below half the sampling frequency. In the experiments when the tuner was tuning from F_4 to E_4, see Appendix I.2, the readout of the tuner nearly always was 349.23 as the starting frequency. In these tests, the sampling frequency was set to double the frequency of F_4. When the string was tuned to be close to F_4, the tuner detected 349.23 Hz no matter what. One way to eliminate this is to increase the sampling frequency. This results in a better readout for when a string is half a step above the tune. Yet this would also worsen the readout if the guitar string is tuned a half or a whole step below the supposed tune. A guitar string is rarely more than one half-step from its original tuning unless the artist explicitly wants it to be. Therefore, it was decided that the sampling frequency will be set to double the frequency of one half-step above the tuning of the string.

The target frequency interval was decided as ± 2 Hz. This was done after tests were conducted where the interval was set to ± 3 Hz. In those experiments, the tuner never ended near the edge of the interval, see Apendix J.5 . The interval was consequently shrunk to ± 2 Hz. When tried with a closer interval, the tuner had a hard time to finish the tune. This was due to it never ultimately reaching the target frequency and kept going back and forth. But also that the tuner sometimes picks up the wrong frequency and sabotages the tuning. When the motor takes a few steps the motor tended to take the steps but the tuning key would not turn any significant amount to change the frequency and then the tuner would be stuck.

Sometimes the frequency detected something utterly different than what the string was tuned to. At first, we believed that the piezo element somehow got affected by the noise in the lab-room. That hypothesis quickly got rejected after some testing and what seems to be happening was that other vibrations got generated from the table of the lab environment or the picking hand touching the guitar body when plucking the string. It could also have been other strings vibrating as a result of plucking the E_4 string. This is a phenomenon called sympathetic resonance. Sympathetic resonance takes place when one string on any multiple-stringed instrument is plucked. The other strings that are tuned so that they align with the harmonic series of the frequency of the plucked string also start vibrating [23]. This could also be read from the feedback display on the commercially available reference tuner. To mediate disturbances, a $\pm$ 50 Hz of E_4 threshold was set up in the software code. If the detected frequency were not within the threshold, the motor would take any steps. After a string was a plucked all strings were lightly damped by the palm of the hand.

When our motor stopped and the program judged the string to be in tune, the reference tuner also showed the string to be in tune. So even though the error from our data was significantly larger than that of the reference tuner, the practical

difference was unrecognizable, see Appendix O. Most times, the tuner only needed to rotate the motor one time to be in tune. Note that this one time rotation is recorded as two attempts in the data. The second attempt is to verify for the Arduino program that the guitar string is in tune. An interesting comparison is that the human ear can notice a difference in pitch of about five to six cents [24], and the average error for our tuner was -3.4 cents.

5.2 Conclusion

A piezoelectric sensor can be used to tune a guitar. The piezo needs to work with a FFT to be able to understand the noted frequencies. The readout of the sensor is strongly connected with the setup. Mainly what sampling frequency used by the FFT affected the readout from the piezo. With proper calibration of the controller, the tuner performed well. The accuracy of the tune was easily on average within -5 cent to +4 cents or $\pm 2Hz$ of the target frequency. Compared to the commercial tuner used to compare, it showed us to be perfectly in tune. On average, the tuning was -3 cents of the target frequency. The human ear can notice differences of 5-6 cents. With the correct values of K_p , the constant for the p-controller, the tuner could most times rotate the tuning key so the string was in tune on its first try.

Chapter 6

Future work

A development of this project could be to create a device that can tune all strings without specifying which tuning key the device is mounted on. This could be done just by categorizing the piezo readings into different intervals belonging to each string. In order to make this work a stronger motor is needed to handle the inertia of the stiffer strings.

An integrated design is appropriate in order to make the device more user friendly. This would albeit be a nice expansion of this device for a project more oriented towards the design aspects of construction. Everything needs to be able to fit inside something that can be handheld.

The tuner in this project uses a P-controller to control the amount steps taken by the stepper motor. This can be developed to be a PID-controller which would be more accurate. However, for our tests with the E_4 string a P-controller are more than sufficient to get a decent result.

Bibliography

[1] Gylling, M and Svensson, *DEGREE PROJECT IN TECHNOLOGY, FIRST CYCLE, 15 CREDITS KTH STOCKHOLM, SWEDEN 2017 Robotic Electric Guitar Tuner* ,KTH, 2017. Available [Onl ine]: http://www.diva-portal.org/smash/get/diva2:1200615/FULLTEXT01.pdf

[2] Glad,T Ljung, L. *Reglerteknik: Grundläggande Teori* Studentlitteratur AB. 4:16 th edition. October 2006. [Book] ISBN 978-91-44-02275-8

[3] French, R. *Engineering the Guitar Theory and Practice* New York, NY : Springer US : Imprint: Springer [Book] Available [Online]: https://kth-primo.hosted.exlibrisgroup.com/permalink/f/qra184/46KTH_ALMA_DS51174906940002456

[4] *En Liten Bok Om Musikteori* Wise publications,2001. Översättare Ola Eriksson. [Book]

[5] Sauveur, J *Principes d'acoustique et de musique, ou Système général des intervalles des sons et de son application à tous les systèmes et à tous les instruments de musique.* Inserted in the "Mémoires" of 1701 of the Royal Academy of Sciences, by M. Sauveur. Date access: 2021-02-15. Available [Online]: https://kth-primo.hosted.exlibrisgroup.com/permalink/f/1vh8pa3/TN_cdi_bnf_primary_oai_bnf_fr_gallica_ark_12148_bpt6k1510877z

[6] Fourier, J-B.J. *Théorie analytique de la chaleur* Paris, Gallica Ebooks Available [Online]: https://kth-primo.hosted.exlibrisgroup.com/permalink/f/1vh8pa3/TN_cdi_bnf_primary_oai_bnf_fr_gallica_ark_12148_bpt6k1045508v

[7] Vretblad, A. *Fourier Analysis and its Applications* 2003, New York, NY: Springer, Available [Online]: https://kth-primo.hosted.exlibrisgroup.com/permalink/f/1vh8pa3/TN_cdi_springer_books_10_1007_b97452

[8] Pring, J. (1965). *The Oxford dictionary of modern Greek* Oxford: Clarendon.[Book]

[9] Manbachi, A, Cobbold, R. S. C. *Development and Application of Piezoelectric Materials for Ultrasound Generation and Detection.* Ultrasound, 19(4), 187-196. Date access: 2021-02-15 Available [Online]:
https://kth-primo.hosted.exlibrisgroup.com/permalink/f/1vh8pa3/TN_cdi_crossref_primary_10_1258_ult_2011_011027

[10] *Arduino Uno* Date access: 2021-02-05 [Online] Available:
https://store.arduino.cc/arduino-uno-rev3:

[11] Barr, M. *Pulse Width Modulation* Embedded Systems Programming, September 2001, pp.103-104. Date access: 05-02-2021. Available [Online] :
https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation

[12] Overview of the Arduino pin Overview of the Arduino pins. Date access. 2021-02-16 Available [Online]:
https://www.electronicwings.com/arduino/arduino-uno-r3-board

[13] *Basics of stepper motor* Date access: 2021-02-15. Available [Online]:
https://www.orientalmotor.com/stepper-motors/technology/stepper-motor-basics.html

[14] Stepper Motor Date access: 2021-02-24 Available [Online]:
https://www.monolithicpower.com/stepper-motors-basics-types-uses

[15] The difference between unipolar and bipolar stepper motors Date Access: 2021-04-10. Available [Online]:
https://techexplorations.com/blog/arduino/blog-the-difference-between-unipolar-and-bipolar-stepper-motors/

[16] Unipolar and Bipolar stepper motor Date Access: 2021-04-10. Available [Online]
https://blog.orientalmotor.com/wiring-basics-unipolar-vs-bipolar

[17] Johansson, H.B. Elektroteknik 2013, KTH, Department of Machine Design [Book]

[18] Actobatic-axle-hub Date Access: 2021-04-26. Available [Online]
https://www.electrokit.com/produkt/actobotics-axelnav-med-fastskruv-5mm/

[19] *Adafruit Stepper Motor driver* Date access: 2021-04-26. Available [Online]:
https://www.adafruit.com/product/2448l

[20] *AudioFrequncyDetector with FFT* Date Access: 2021-03-20. Available [Online]: https://clydelettsome.com/blog/2019/12/18/my-weekend-project-audio-frequency-detector-using-an-arduino/

[21] *ArduinoFFT* Date Access: 2021-03-15. Available [Online]: https://www.arduino.cc/reference/en/libraries/arduinofft/

[22] Stepper motor software Date Access 2021-03-10. Available [Online]: https://lastminuteengineers.com/28byj48-stepper-motor-arduino-tutorial/

[23] Rossing, T.D. *The Science of String Instruments* 2010 Springer [Book] Available [Online]: https://kth-primo.hosted.exlibrisgroup.com/permalink/f/1vh8pa3/TN_cdi_askewsholts_vlebooks_9781441971104

[24] Loeffler, D.B. *Instrument Timbres and Pitch Estimation in Polyphonic Music* ,Georgia Institute of Technology, 2006. Date access: 2021-05-05. Available [Online]: https://web.archive.org/web/20060914233900/http://etd.gatech.edu/theses/available/etd-04102006-142310/unrestricted/loeffler_dominik_b_200605_mast.pdf

[25] *14 relevant parts of an acoustic guitar.* Date Access: 2021-03-15. Available [Online]: https://bestbeginnerguitartoday.com/parts-of-an-acoustic-guitar/

[26] Korg AW2G Clip-On Chromatic Guitar Tuner Date Access: 2021-04-27. Available [Online]: https://www.amazon.com/Korg-AW2G-Clip-Chromatic-Guitar/dp/B001XJBWXG?th=1

[27] Pitchclip 2 Clip-on Tuner Date Access: 2021-04-30. Available [Online]: https://www.korg.com/se/products/tuners/pitchclip2/specifications.php

Appendix A

Component list

- Set screw hub (5mm). Appendix M.
- Arduino Uno Microcontroller Board (Playknowlogy Uno Rev. 3. model). For datasheet see reference [10].
- Battery (12V) from Yuasa.
- Battery snap (for 9V battery).
- Battery (9V lithium) *****.
- Capacitor (100μF),
- Clip-on tuner (Korg Pitchclip 2) used for calibration. Appendix O.
- Clip-part de-assembled from a 'Clip-On Guitar Tuner' (Korg AW2G) Appendix N.
- Grip for Tuning Key, 3D printed according to blueprint in Appendix E.
- Jumper wires of appropriate length *****.
- Heat sink 20x20mm *****.
- Piezo element (9 kHz, Murata)
- Resistor (10MΩ).
- Stepper Motor (Bipolar, Nema 17), Appendix D.

APPENDIX A. COMPONENT LIST

* No further specification required in order to replicate the construction described above. These parts are elementary and any corresponding version with similar measurements can be used equivalently.

Appendix B

Glossary

fretting	pressing a finger somewhere on the fretboard to get a certain note
open string	playing a guitar string without fretting

Appendix C

Parts of an acoustic guitar

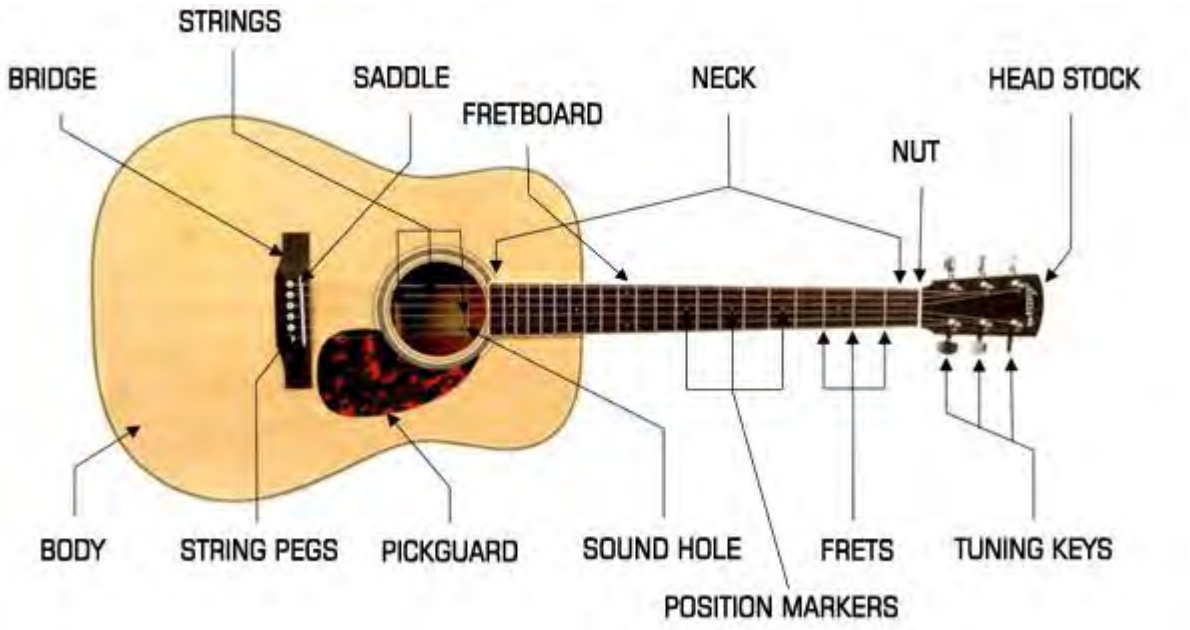


Figure C.1. 14 relevant parts of an acoustic guitar [25].

Appendix D

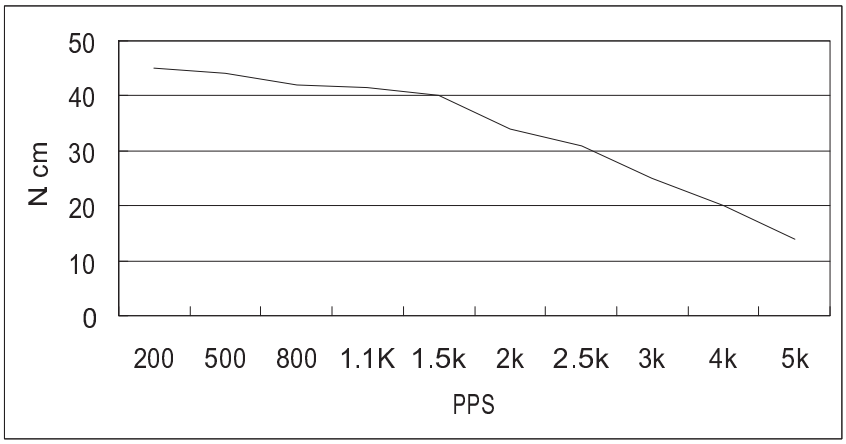
Stepper Motor Datasheet: Nema 17

HIGH TORQUE HYBRID STEPPING MOTOR SPECIFICATIONS

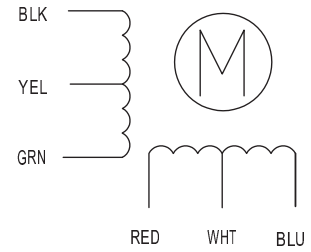
General specifications		Electrical specifications	
Step Angle (°)	1.8	Rated Voltage (V)	4
Temperature Rise (°C)	80 Max (rated current.2 phase on)	Rated Current (A)	1.2
Ambient temperature (°C)	-20～+50	Resistance Per Phase (±10%Ω)	3.3 (25°C)
Number of Phase	2	Inductance Per Phase (±20%mH)	2.8
Insulation Resistance	100MΩ, Min (500VDC)	Holding Torque (Kg.cm)	3.17
Insulation Class	Class B	Detent Torque (g.cm)	200
Max.radial force (N)	28 (20mm from the flange)	Rotor Inertia (g. cm²)	68
Max.axial force (N)	10	Weight (Kg)	0.365

● Pull out torque curve:

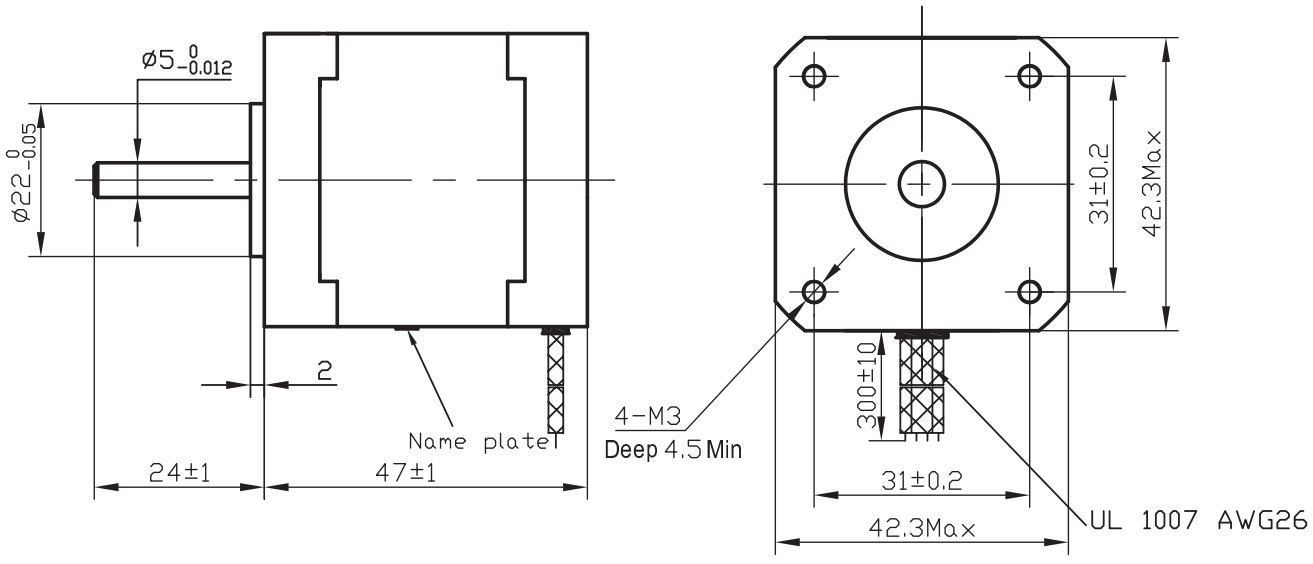
VOLTAGE: 24VDC, CONSTANT CURRENT: 1.2A, HALF STEP



● Wiring Diagram:



● Dimensions:
(unit=mm)



REV	REVISIONS	DESCRIPTION	BY	DATE
2				
1				
DRAW	任飞飞 2010.06.29			
CHECK				
APPROVE				

SY42STH47-1206A

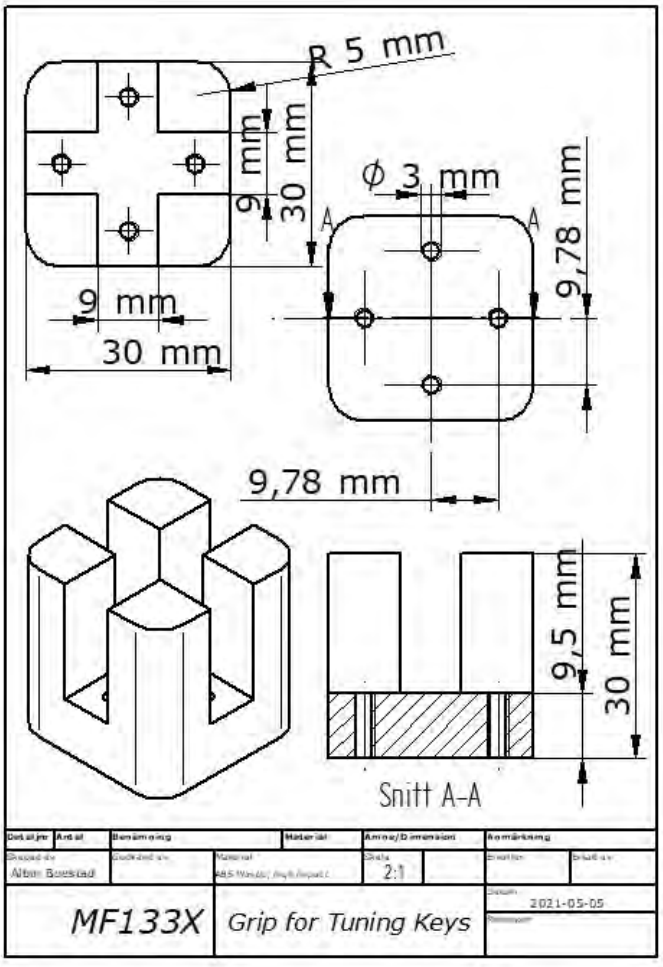
TECHNICAL CONDITIONS

CHANGZHOU SONGYANG MACHINERY & ELECTRONICS NEW TECHNIC INSTITUTE

060047000

Appendix E

Blueprint Tuning Key Gripper



Appendix F

Arduino Code

```
/*
 File/Sketch Name: Guitar Tuner
 School: KTH Royal Institute of Technology
 Made by: Albin Boestad  & Fabian Rudberg
 Created: 2021-04-09  Version 1.0:
 Trita:-ITM-EX 2021:24

 Description: This code is based on the AudioFrequencyDetection code written by Clyda.A Lettersome
 and can be found: https://clydelettsome.com/blog/2019/12/18/my-weekend-project-audio-frequency-detector-using-an-arduino/
 This code detects the frequency detetctet from a piezo sensor on a guitar. The frequncy is compared
 to see if the string is in tune. Then the Arduino controls the stepper motorto adjust the tuning peg
 so the guitar is in tune.

 The piezo elemment sends a analog adio signal to A0 on the Arduino Uno and then gets sampled.
 A Fast Fourier Tranfrom(FFT) is used on the data using the library arduinoFFT.h. The FFT transform
 the signal from the time doma  to the frequency domain. The maximum frequency detetcted is determined.
 The frequencys is displays using the Arduino Serial Monitor and is compared to the frequency of a in
 tune string. Then a stepper motor is rotated based on how wrong the frequency is.
*/
```

```
20 //Includes the arduinoFFT library as well as the Stepper
     library.
21 #include <arduinoFFT.h>
22 #include <Stepper.h>
23
24  //float freqarray[] = {82.41, 110, 146.83,196.00, 246.94,
      329.63}; //329.63Standrad frequency for a guitar in
      following order [E2, A2, D3, G3, B3, E4]
25  float piezoarray[] = {333.54}; //This frequncy is the
      avrage readout for the piezo when the string is tuned to
       E4
26
27 #define SAMPLES 128          //Sets the amount of samples
     that will be taken for the FFT. Must be a base 2 number.
     Max 128 for Arduino Uno.
28 #define SAMPLING_FREQUENCY 698.46//Based on Nyquist, must be
      2 times the highest expected frequency. For the guitar
     E4 is the highest frequency.
29
30
31
32 unsigned int samplingPeriod; //Creates an int for the
     sampling period
33 unsigned long microSeconds; //creats a long for the
     microSeconds
34
35 double vReal[SAMPLES]; //create vector of size SAMPLES to
     hold real values
36 double vImag[SAMPLES]; //create vector of size SAMPLES to
     hold imaginary values
37
38 // Number of steps per internal motor revolution
39 const float STEPS_PER_REV = 200;
40
41
42 // Setups the steppermotor, creaets an Stepper class named
     steppermotor.
43 // Specifies the pins that going to the stepper driver. The
     pins that are
44 // used are 8,9,10,11 that connects to In1, In2, IN3, In4 on
      the stepper
45 // motor driver board ULN2003. The pins need to be entered
     in the sequence
46 // 1-3-2-4 to be able to make the proper steps sequencing
```

```
47 Stepper steppermotor(STEPS_PER_REV, 7,8,10,13);
48 //Creates a instance of arduinoFFT class named FFT.
49 arduinoFFT FFT = arduinoFFT();
50
51 // Define Variables
52
53 int StepsRequired;//StepsRequired is the amount steps that
      are needed to turn.
54 float Difference;
55 int var = 1; // Var is variable that changes depending on
      which case the code is in
56 int t = 0; //t is a variabel that changes which string we
      are tuning. As of this version only E4 is tuned and
      therefore is t set to 0.
57 float factor ; //Factor for the P-regulator. Facktor = 4.8
      is good when below E4 and factor = 6.8 is good when above
       E4.
58
59 void setup()
60 {
61     Serial.begin(115200); //Baud rate for the Serial Monitor
62     samplingPeriod = round(1000000*(1.0/SAMPLING_FREQUENCY))
          ; //Period in microseconds
63
64     steppermotor.setSpeed(250);
65
66 }
67
68 void loop()
69 {
70
71   if(var ==1)
72   {
73   int var = 1; // Sets varible to 1 so that case 1 is
         running.
74   }
75
76   if(var !=1)
77   {
78   int var = var; //Sets the varible so that i does not
         change. If it is in case 2 it stays in cse 2. Case 2 is
          when the tuning is finished.
79   }
80
```

```
81      switch(var){
82
83        case 1: //Case 1  Case 1 is where it does sampling and
                turns the stepper motor
84
85        Serial.print("\n");
86        Serial.print("New reading");
87        Serial.print("n");
88        delay(1000);
89
90
91
92      /*Sample SAMPLES times*/
93      for(int i=0; i<SAMPLES; i++)
94      {
95          microSeconds = micros();    //Returns the number of
              microseconds since the Arduino board began
              running the current script.
96
97          vReal[i] = analogRead(A0); //Reads the value from
              analog pin 0 (A0), quantize it and save it as a
              real term.
98          vImag[i] = 0;              //Makes imaginary term
              always 0
99
100         /*remaining wait time between samples if necessary*/
101         while(micros() < (microSeconds + samplingPeriod))
102         {
103           //do nothing
104         }
105
106     }
107
108
109
110     /*Perform FFT on samples*/
111     FFT.Windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING,
          FFT_FORWARD);
112     FFT.Compute(vReal, vImag, SAMPLES, FFT_FORWARD);
113     FFT.ComplexToMagnitude(vReal, vImag, SAMPLES);
114
115     /*Find peak frequency and print peak*/
116     double peak = FFT.MajorPeak(vReal, SAMPLES,
          SAMPLING_FREQUENCY);
```

```
117
118    /*Print out on Serial Monitor*/
119     Serial.print("\n");
120     Serial.print("Peak frequency:");
121     Serial.print("\n");
122     Serial.println(peak);      //Print out the most dominant
           frequency.
123
124     /*Depending on which string and what frequencys was
           detetcted the following happens*/
125  if (peak<=(piezoarray[t]+50) && (piezoarray[t]-50)<=peak)
126  // The motor will only rotate if peak frequency is not to
       far away.
127  //This is a measurment incase the sampling went wrong and
       the detetcted peak frequencys is wrong
128    {
129     if((piezoarray[t]-2)<= peak && peak <= (piezoarray[t]+2)
           ) //If the sampled frequencys is withing
130      //an interval of +-2 Hz, we say it is in tune
131        {       /*Writes on seriel monitorn the following*/
132                Serial.print("String is in tune");
133                Serial.print("\n");
134                var = 2;         //Switches var so the case
                      will switch to case 2
135                delay(3000);     //Waits 3 seconds, usefull to
                       able to see what happens
136                                 //on Serial Monitor as well
                                       as the motors rotation.
137                break;           //breaks the case and returns
                       to the loop
138         }
139
140                Difference = (piezoarray[t]-peak); //
                      Calculates the differences of the peak and
                       intune frequncy.
141                /*Prints out the following in serial monitor
                      */
142                Serial.print("Diffrences:");
143                Serial.print(piezoarray[t]-peak);
144                Serial.print("\n");
145                Serial.print("Amount of steps ");
146
147
148
```

```
149                 if (Difference<=0) //If we are above E4 teh
                       differences will be neagtive.
150                 {
151                  faktor = 6.8;     //Facktor set to 6.8 will
                        be used if the detected frequncy is above
                         E4
152                  StepsRequired = Difference*faktor; //Steps
                        the motor needs to take
153                 }
154                  else
155                  {
156                   faktor = 4.8;     //If the differences is
                        positive the detetcted frequncy is below
                         E4, and
157                                       //the factor = 4.8 will
                                            be used
158                   StepsRequired = Difference*faktor; //Steps
                        the motor needs to take
159                  }
160                   Serial.print(StepsRequired);
161                   delay(1000);
162                   Serial.print("\n");
163                   steppermotor.step(StepsRequired); //
                        Rotatates the motor StepsRequired step.
164                   delay(2000);
165    }
166    memset(vReal, 0, sizeof(vReal)); //Makes sure that all
          values in vReal is set to zero before a new sampling
167    break; //G  r  ut case 1
168
169
170    case 2: //When the string is in tune a led is lit up by
          seting pin 2 to HIGH.
171
172   //Do nothing when the string is in tune
173
174   ;
175     }
176
177
178 }
```

Appendix G

Acumen Simluation

```
//Simulation Piezoelectric guitar tuner
// SChool: KTH Royal Institute of Technology
//Made by: Albin BOestad & Fabian Rudberg
// Created: 2021-03-23
// Trita: ITM-EX 2021:24

//Description: Simulation of the piezoelectri guitartuner.
//The tuner will rotate a grip that is attached to the
// guitar tunning key.
//The tuner will be able to rotate the the tuning key in
//both direction depending if the string needs to be
//tightened or loosened.



model box (x,y,z) = //Creates a box which has its center
    // in koordinates (x,y,z).
initially
_3D = ()


always

         _3D =
           (Box     //Creates a box the represents all
    //the electronics for the tuner.
           center=(x,y, z)
           size=(10, 7, 12) //The size of the box is given
            // on the form (x, y, z)
```

```
           color = white)

 model cyl_ (x,y,z,a) =  //Creates a cylinder. In data is
 // the center of the cylinder and
 //the angle,"a", that the cylinder
 // should rotate
 initially
 _3D = ()
 always

          _3D =  (Cylinder //Creates model of the griper that
   //will attach to the tuning key.
            center = (x, y,z)

            size = (2, 2)  //Size if the cylinder is given
   // as (hight, radius).
          color = yellow
          rotation = (0,a,0) //Tells the model to rotate around
     //the y-axle with the angle a.
          )
model peg (x,y,z,a) =   //Model of the tuning key

initially
   _3D = ()
 always
        _3D =
           (Box
           center=(x,y, z)
           size=(0.5, 1, 5)
           color = blue
           rotation = (0,a,0)) //Tells the model to rotate around
       // the y-axle with angle a. The same
       //rotation for both the tning key
      //model as the grip model


model Main(simulator) =
initially //THe following code is run only once

b = create box(0,0,-4),  //Creates the obejct box
c = create cyl_(0, -4.5, -1,0), //Creates the object cylinder
tp = create peg(0, -5.5, -1,0), //Creates the obejct tuning key

t = 0, t'=0,  //Time starts at Zero and does not accelrate
```

```
a = 0, a' = 0 //Angle "a" is zero in the beginning and the angular
      //velocity, a', is zero.


always //Following code is run constantly.n.
t'=1, //Decides the speed of the simluation. With t' = 1 the time
      //will move with 1 second/second
          // Sets the angular velocity to 1 when t<3, corresponds
  //to the Arduino tells the motor to rotate
          // in one direction a certain amount of steps.
          if t<= 3 then

             a' = 1

            //Here the motor has rotate in one direction but rotate
      //to far and therefore rotates back
            else if  3<t && t<= 6 then
            a' = -1 //The minus corresponds to rotate in the
    //other direction

           //After thiscorrectio we are still to far of so the
           //tuner makes one last rotation
           else if  6<t && t<=7 then
             a'=1


            //Now the tuner says the guitar is in tune
             else
             a' = 0, // When the angular velocity is set to zero the model
     //stops rotating
 angle a
  c.a = a, //Send to the cylinder model that it shall rotate with
   //the the angle 'a'. By sending this comand constantly
   //the model will make a rotating motion
  tp.a =a //Send to the tuning key model that it shall rotate with
  // the the angle 'a'. By sending this comand constantly
  //the model will make a rotating motion
```

Appendix H

Piezo Calibration Data

Table H.1. Tuning the E_4 string to the E_4-frequency of 329.63Hz according to the tuner and noting the equivalent frequency as detected by the Piezo-element (and approximated as frequency through the FFT-algorithm). For Pitchclip 2 specs, see Appendix O and for Piezo-element specification, see Appendix **??**

Pitchclip 2 - Frequency	***Piezo - Frequency**
329.63	333.40
329.63	333.49
329.63	333.56
329.63	333.40
. 329.63	333.58
329.63	333.81
329.63	333.35
329.63	333.79
329.63	333.31
329.63	333.67

***** The average piezo-frequency value for E_4 became **333.52 Hz**.

Table H.2. Tuning the E_4 string to the $D\#$-frequency of 311.13Hz according to the tuner and noting the equivalent frequency as detected by the Piezo-element (and approximated as frequency through the FFT-algorithm). For Pitchclip 2 specs, see Appendix O and for Piezo-element specification, see Appendix **??**

Pitchclip 2 - Frequency	***Piezo - Frequency**
311.13	314.13
311.13	314.00
311.13	315.11
311.13	314.71
311.13	314.18
311.13	313.88
311.13	314.89
311.13	313.82
311.13	310.41
311.13	313.70

***** The average piezo-frequency value for $D\#$ became **314.18 Hz**.

Table H.3. Tuning the E_4 string to the F-frequency of 349.23Hz according to the tuner and noting the equivalent frequency as detected by the Piezo-element (and approximated as frequency through the FFT-algorithm). For Pitchclip 2 specs, see Appendix O and for Piezo-element specification, see Appendix **??**

Pitchclip 2 - Frequency	***Piezo - Frequency**
349.23	349.23
349.23	347.26
349.23	349.23
349.23	349.23
. 349.23	349.23
349.23	349.23
349.23	349.23
349.23	349.23
349.23	349.23
349.23	349.23

***** The average piezo-frequency value for F became **349.03 Hz**.

Appendix I

Calibration data for the constant of proportional control, K_p

I.1 Tuning from $D\#_4$ to E_4.

Table I.1. $K_p = 3.0$. Test 1.

Iteration	Frequency	Steps
1	314.59	56
2	352.42	24
3	331.87	5
4	332.61	0

Table I.2. $K_p = 5.0$. Test 1.

Iteration	Frequency	Steps
1	314.19	96
2	334.27	0

APPENDIX I. CALIBRATION DATA FOR THE CONSTANT OF PROPORTIONAL CONTROL, K_p

Table I.3. $K_p = 5.0$. Test 2.

Iteration	Frequency	Steps
1	314.35	95
2	334.78	-6
3	334.77	-6
4	334.55	0

Table I.4. $K_p = 5.0$. Test 3.

Iteration	Frequency	Steps
1	314.66	94
2	334.19	0

Table I.5. $K_p = 5.0$. Test 4.

Iteration	Frequency	Steps
1	314.25	96
2	335.40	-9
3	334.74	-5
4	333.77	0

Table I.6. $K_p = 5.0$. Test 5.

Iteration	Frequency	Steps
1	314.34	96
2	335.35	-9
3	334.04	0

APPENDIX I. CALIBRATION DATA FOR THE CONSTANT OF PROPORTIONAL CONTROL, K_p

Table I.7. $K_p = 4.5$. Test 1.

Iteration	Frequency	Steps
1	313.84	88
2	333.94	0

Table I.8. $K_p = 4.5$. Test 2.

Iteration	Frequency	Steps
1	314.63	85
2	331.30	10
3	333.99	0

Table I.9. $K_p = 4.5$. Test 3.

Iteration	Frequency	Steps
1	314.44	85
2	331.47	9
3	333.99	0

Table I.10. $K_p = 4.8$. Test 1.

Iteration	Frequency	Steps
1	314.58	91
2	334.34	0

Table I.11. $K_p = 4.8$. Test 2.

Iteration	Frequency	Steps
1	313.20	97
2	333.95	0

APPENDIX I. CALIBRATION DATA FOR THE CONSTANT OF PROPORTIONAL CONTROL, K_p

Table I.12. $K_p = 4.8$. Test 3.

Iteration	Frequency	Steps
1	314.56	91
2	331.29	10
3	334.85	-6
4	334.99	-6
5	334.55	-5
6	334.11	0

Table I.13. $K_p = 4.8$. Test 4.

Iteration	Frequency	Steps
1	314.58	91
2	334.00	10

Iteration	Frequency	Steps
1	314.57	91
2	333.35	0

Table I.14. $K_p = 4.8$. Test 5.

I.2 Tuning from F_4 to E_4

Iteration	Frequency	Steps
1	349.23	-75
2	339.34	-27
3	334.20	0

Table I.15. $K_p = 4.8$. Test 1.

.

Iteration	Frequency	Steps
1	349.23	-75
2	339.39	-28
3	333.83	0

Table I.16. $K_p = 4.8$. Test 2.

.

Iteration	Frequency	Steps
1	349.23	-94
2	335.40	-11
3	332.72	0

Table I.17. $K_p = 6.0$. Test 1.

.

Table I.18. $K_p = 6.0$. Test 2.

Iteration	Frequency	Steps
1	349.23	-94
2	336.02	-14
3	332.96	0

.

APPENDIX I. CALIBRATION DATA FOR THE CONSTANT OF PROPORTIONAL CONTROL, K_p

Table I.19. $K_p = 7.0$. Test 1.

Iteration	Frequency	Steps
1	349.23	-109
2	331.62	13
3	333.93	0

Table I.20. $K_p = 6.5$. Test 1.

Iteration	Frequency	Steps
1	349.23	-101
2	334.22	0

Table I.21. $K_p = 6.5$. Test 2.

Iteration	Frequency	Steps
1	349.23	-101
2	334.61	-6
3	333.48	0

Table I.22. $K_p = 6.5$. Test 3.

Iteration	Frequency	Steps
1	349.23	-101
2	333.78	0

Table I.23. $K_p = 6.8$. Test 1.

Iteration	Frequency	Steps
1	349.23	-106
2	333.63	0

APPENDIX I. CALIBRATION DATA FOR THE CONSTANT OF PROPORTIONAL CONTROL, K_p

Table I.24. $K_p = 6.8$. Test 2.

Iteration	Frequency	Steps
1	349.23	-106
2	332.65	0

Table I.25. $K_p = 6.8$. Test 3.

Iteration	Frequency	Steps
1	349.23	-106
2	333.83	0

Appendix J

Results Appendix

J.1 Data Tables. One half-step below and above E_4. Acceptable tuning interval ± 3. K = 5. Non-calibrated reference frequency for Piezo.

Table J.1. Automatically tuning from $D\#_4$ to E_4 with proportional constant, K=5, and acceptable tuning interval equal to ±3 Hz

Attempt*	Tuner Freq. [Hz]	Piezo Freq. [Hz]	Steps**	Trials***
1	311.13	313.86	78	
		329.83		2
2	311.13	314.39	76	
		330.05		2
3	311.13	314.47	75	
		329.74		2
4	311.13	313.63	80	
		330.17		2
5	311.13	313.85	78	
		330.28		2
6	311.13	313.82	79	
		330.37		2
7	311.13	314.85	73	
		329.72		2
8	311.13	314.57	75	
		349.23	-98	
		311.22	92	
		329.05		4
9	311.13	314.07	77	
		329.48		2
10	311.13	349.23	-98	
		292.95	183	
		330.45		3

***** The E4 - string count as 'in tune' when within the interval of 329.63 +/- 3 Hz. This is for uncalibrated piezo-frequency and with a wider larger interval than later was shown to be more effective.

****** Negative sign (-) before recorded steps means clockwise rotation

******* One trial means hitting the string once and getting a response-measurement

Table J.2. Automatically tuning from F_4 to E_4 with proportional constant, K=5, and acceptable tuning interval equal to ±3 Hz

Attempt	Tuner Freq. [Hz]	Piezo Freq. [Hz]	Steps	Trials
1	349.23	349.23	-98	
		335.18	-27	
		329.70		3
2	349.23	349.23	-98	
		335.17	-27	
		329.67		3
3	349.23	349.23	-98	
		334.75	-25	
		330.68		3
4	349.23	349.23	-98	
		334.70	-25	
		330.16		3
5	349.23	349.23	-98	
		334.35	-23	
		329.68		3
6	349.23	349.23	-98	
		333.81	-20	
		329.25		3
7	349.23	349.23	-98	
		334.96	-26	
		330.03		3
8	349.23	349.23	-98	
		334.92	-26	
		329.77		3
9	349.23	349.23	-98	
		349.23	-98	
		313.27	81	
		329.47		4
10	349.23	348.36	-93	
		339.03	-47	
		329.70		3

J.2 Data Tables. One half-step below and above E_4. Acceptable tuning interval ± 2. K = 4.8. Calibrated Piezo-frequency.

Table J.3. Automatically tuning from $D\#_4$ to E_4 with proportional constant, K=4.8, and acceptable tuning interval equal to ± 2 Hz

Attempt*	Tuner Freq. [Hz]	Piezo Freq. [Hz]	Steps**	Trials***
1	311.13	314.13	93	
		328.21	25	
		337.87	-29	
		334.50	0	4
2	311.23	314.00	93	
		334.44	0	2
3	311.23	315.11	88	
		331.16	11	
		332.59	0	3
4	311.23	314.71	90	
		332.23	0	2
5	311.23	314.18	92	
		332.57	0	2
6	311.23	313.88	84	
		333.57	0	2
7	311.23	314.89	89	
		333.34	0	2
8	311.23	313.82	94	
		333.04	0	2
9	311.23	310.41	111	
		332.48	0	2
10	311.23	313.70	95	
		333.29	0	2

* The E4 - string count as 'in tune' when within the interval of 333.52 +/- 2 Hz. The value of 333.52 piezo-calibrated according to Appendix H

** Negative sign (-) before recorded steps means clockwise rotation

*** One trial means hitting the string once and getting a response-measurement

Table J.4. Automatically tuning from F to E_4 with proportional constant, K=4.8, and acceptable tuning interval equal to ±2 Hz

Attempt	Tuner Freq. [Hz]	Piezo Freq. [Hz]	Steps	Trials
1	349.23	349.23	-106	
		331.90	0	2
2	349.23	347.26	-93	
		334.94	0	2
3	349.23	349.23	-106	
		331.63	0	2
4	349.23	349.23	-106	
		333.20	0	2
5	349.23	349.23	-106	
		333.20	0	2
6	349.23	349.23	-106	
		332.25	0	2
7	349.23	349.23	-106	
		332.96	0	2
8	349.23	349.23	-106	
		332.81	0	2
9	349.23	349.23	-106	
		330.85	12	
		331.72	0	3
10	349.23	349.23	-106	
		332.18	0	2

J.3 Data Tables. Five points between $D\#_4$ and E_4. Acceptable tuning interval ± 2. K = 4.8. Calibrated Piezo-frequency.

Attempt	Piezo freq.	Steps	Trials
1	320.25	63	
	333.23	0	2
2	323.93	50	
	331.67	0	2
3	325.94	36	
	333.00	0	2
4	327.88	27	
	333.00	0	2
5	330.18	16	
	333.97	0	2

J.4 Data Tables. Five points between F and E_4. Acceptable tuning interval ± 2. K = 6.8. Calibrated Piezo-frequency.

Attempt	Piezo freq.	Steps	Trials
1	347.00	-91	
	329.27	20	
	331.00	12	
	332.97	0	4
2	346.17	-85	
	330.10	16	
	330.28	15	
	334.24	0	4
3	341.08	-51	
	332.28	0	2
4	339.43	-40	
	332.40	0	2
5	337.48	-26	
	332.23	0	2

J.5 Graphs for acceptable tuning interval ± 3

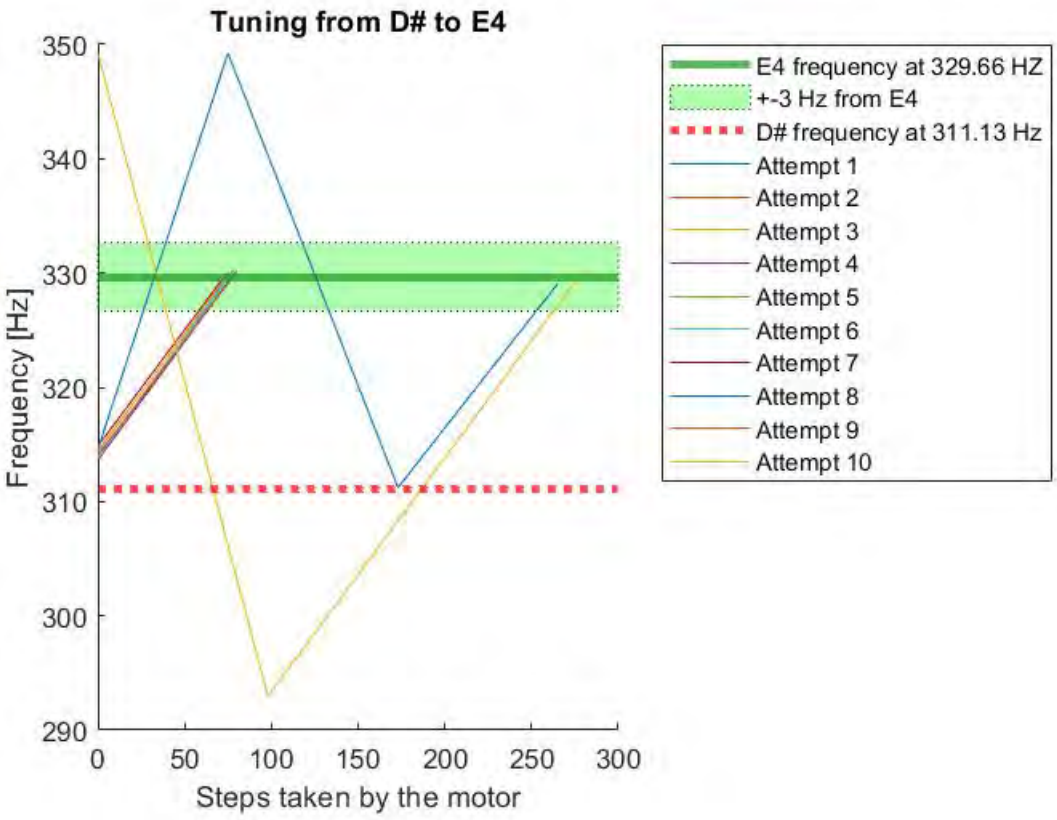


Figure J.1. Graf of tuning from D# to E_4 with a +-3Hz acceptable tuning interval, graf was made using emphMATHWORKS MATLAB R2020b

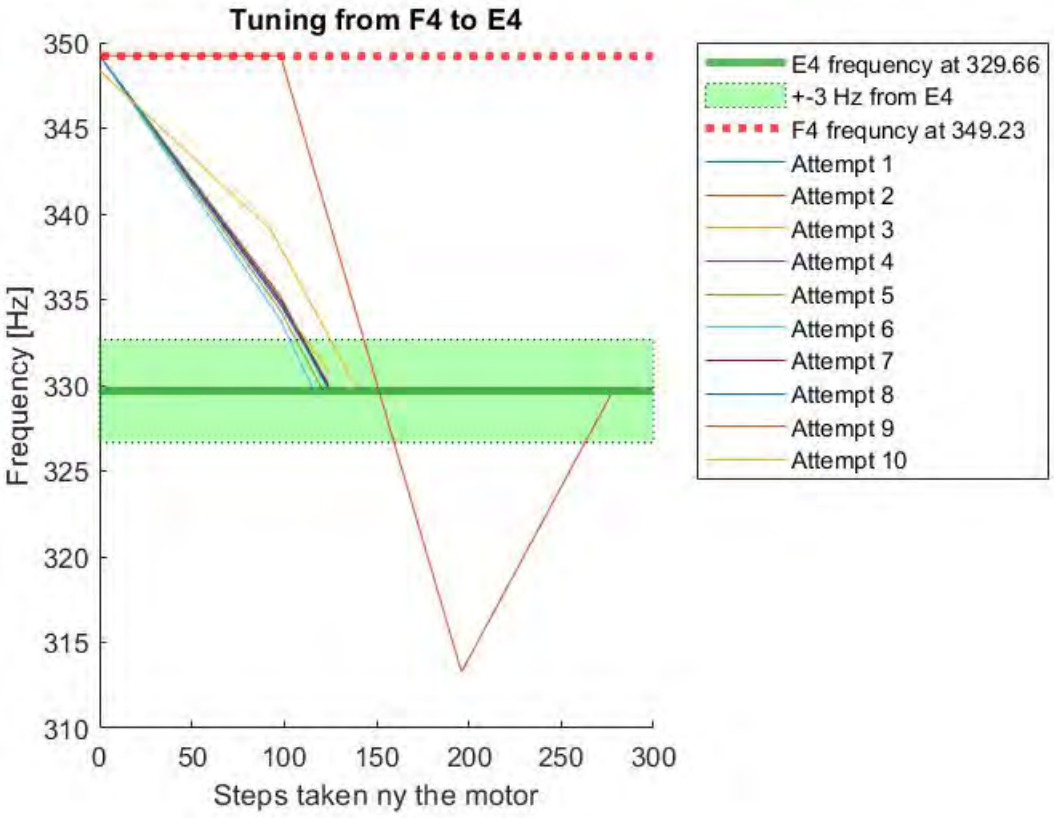


Figure J.2. Graf of tuning from F_4 to E_4 with a +-3 Hz acceptable tuning interval, graf was made using emphMATHWORKS MATLAB R2020b

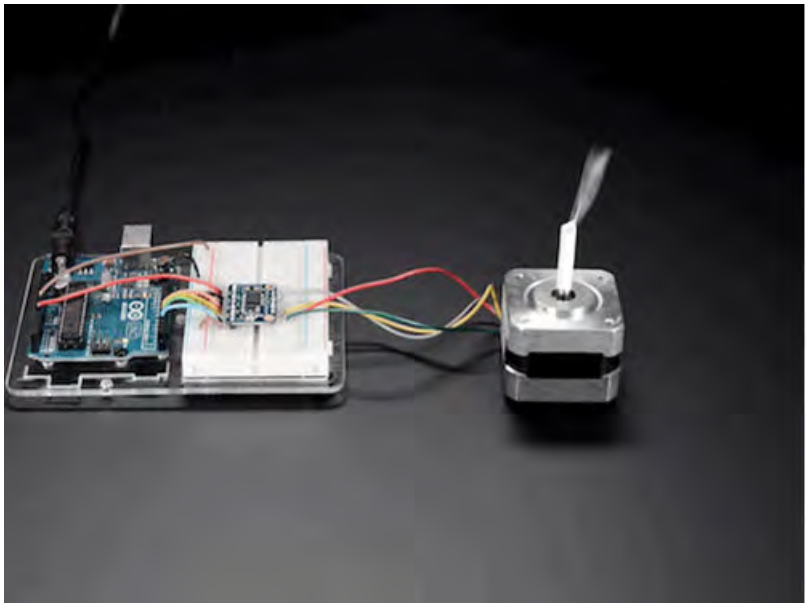
Appendix K

Adafruit Stepper Motor driver

adafruit learning system

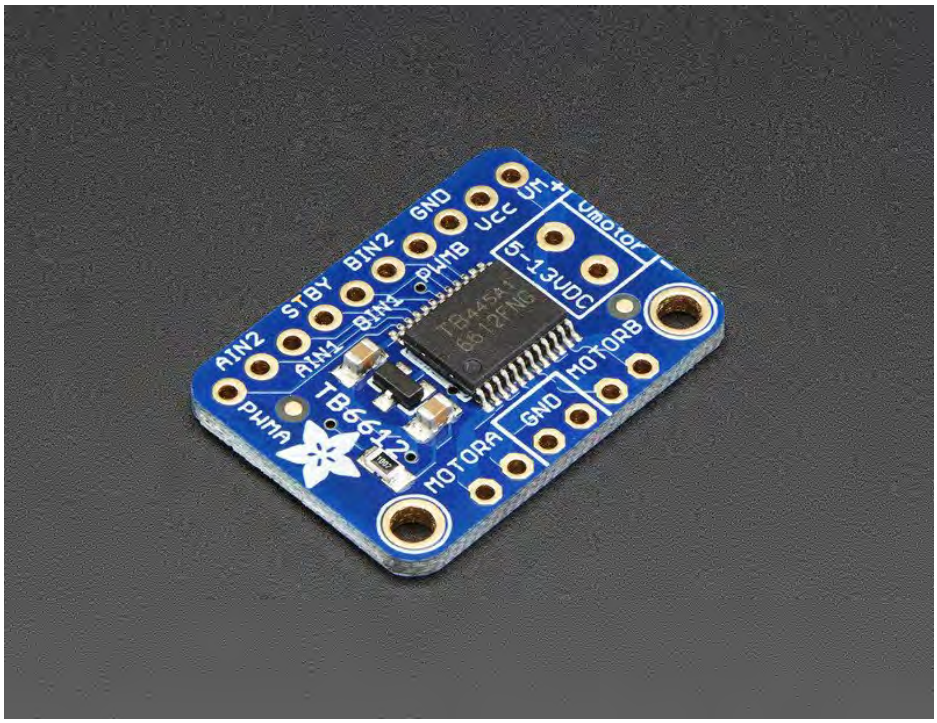
Adafruit TB6612 1.2A DC/Stepper Motor Driver Breakout Board

Created by lady ada

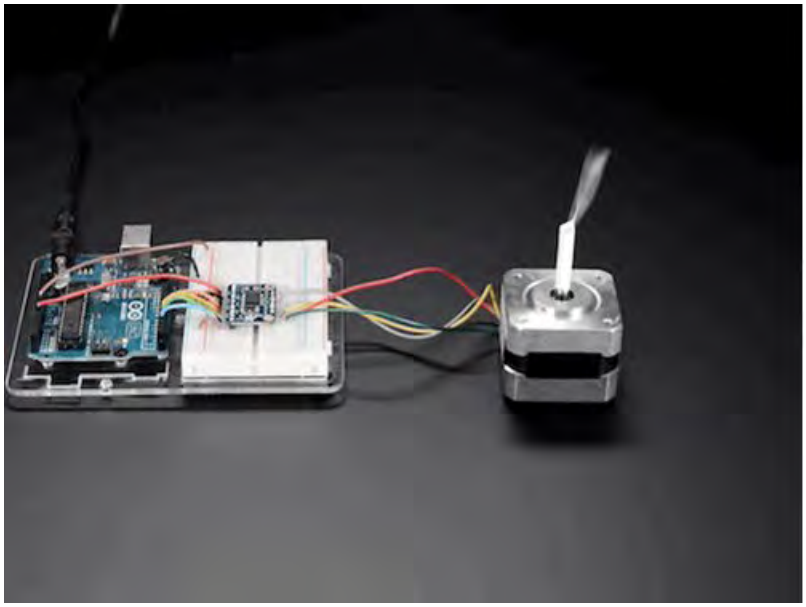


Last updated on 2020-11-20 05:59:31 PM EST

Overview

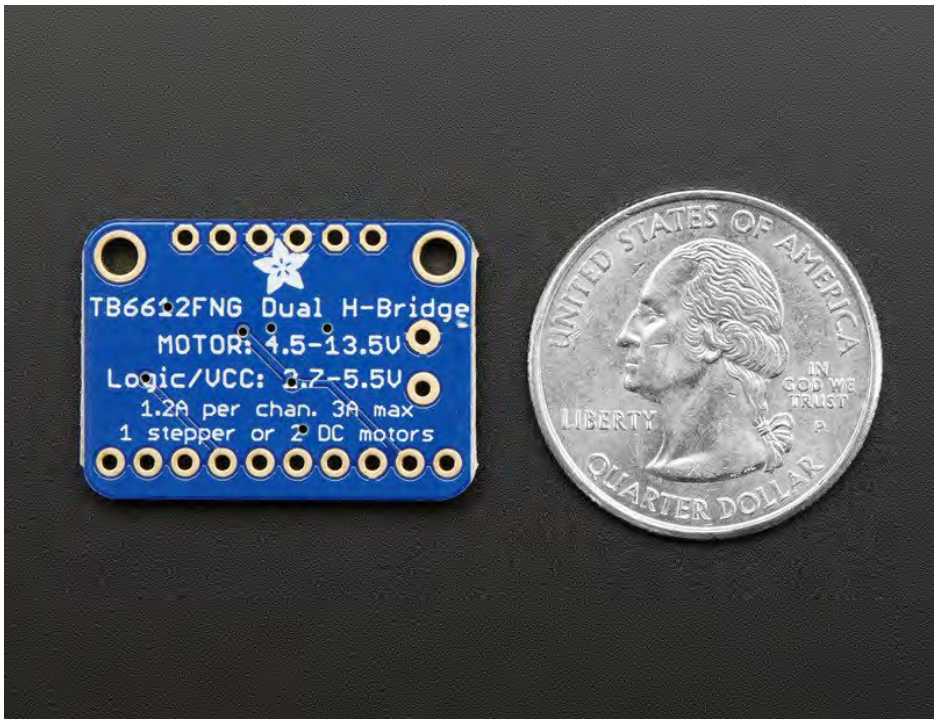


Fire four solenoids, spin two DC motors or step one bi-polar or uni-polar stepper with 1.2A per channel using the TB6612. These are perhaps better known as "the drivers in our assembled Adafruit Motorshield (http://adafru.it/1438) or Motor HAT (https://adafru.it/eRq)" We really like these dual H-bridges, so if you want to control motors without a shield or HAT these are easy to include on any solderless breadboard or perma-proto.

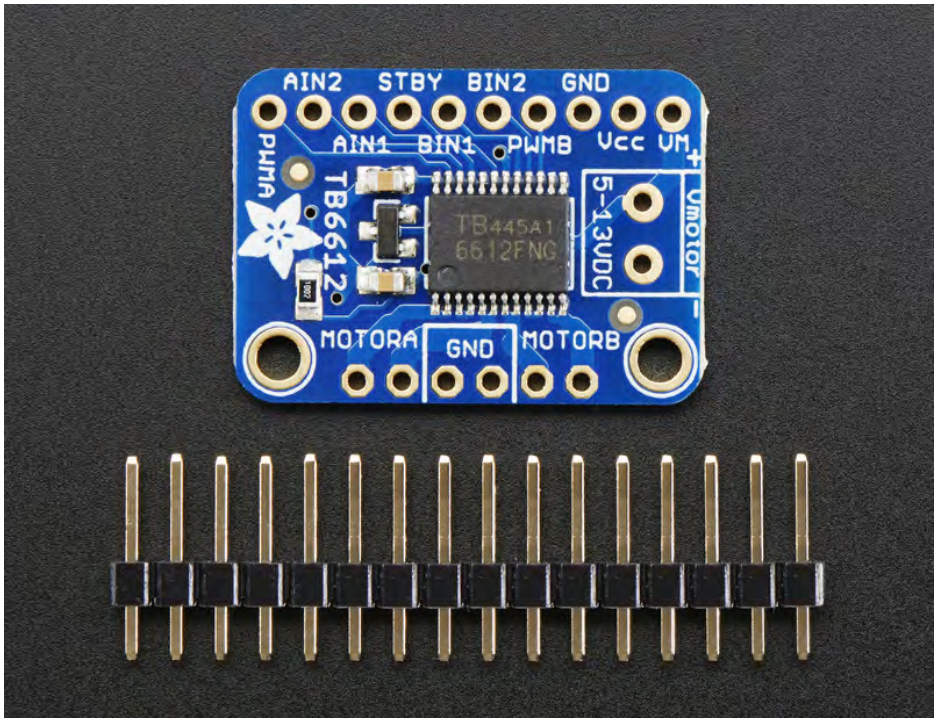


We solder on TB6612 onto a breakout board for you here, with a polarity protection FET on the motor voltage input and a pullup on the "standby" enable pin. Each breakout chip contains two full H-bridges (four half H-bridges). That means you can drive four solenoids, two DC motors bi-directionally, or one stepper motor. Just make sure they're good for 1.2 Amp or less of current, since that's the limit of this chip.

They do handle a peak of 3A but that's just for a short amount of time, about 20 milliseconds. What we like most about this particular driver is that it comes with built in kick-back diodes internally so you dont have to worry about the inductive kick damaging your project or driver!

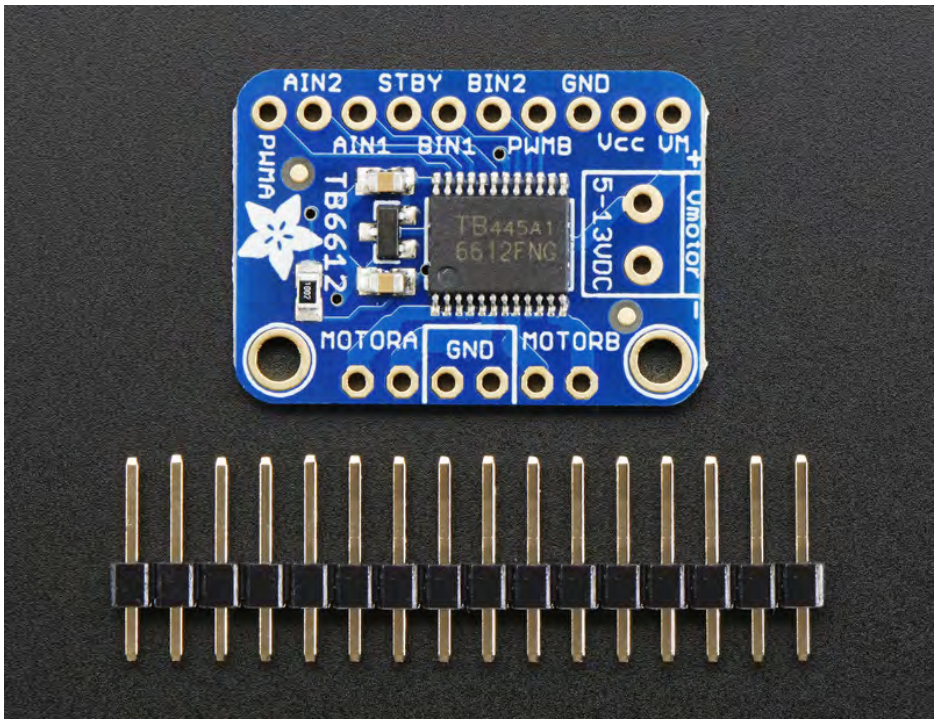


There's two digital inputs per H-bridge (one for each half of the bridge) as well as a PWM input per driver so you can control motor speed. Runs at 2.7V-5V logic. The motor voltage is separate from the logic voltage. Good for motor voltages from 4.5V up to 13.5V! This wont work well for 3V motors.



Comes as one assembled and tested breakout plus a small strip of header. You'll need to do some light soldering to attach the header onto the breakout PCB. Arduino, motors, and power supply not included.

Pinouts



This motor driver is a fairly simple breakout of the TB6612 motor chip, so do check out the datasheet for the TB6612 for any details you need about pin voltage limits, capacitance, etc! (https://adafru.it/emK)

Power Pins

- **Vmotor** - This is the voltage for the motors, not for the logic level. Keep this voltage between 4.5V and 13.5V. This power supply will get noisy so if you have a system with analog readings or RF other noise-sensitive parts, you may need to keep the power supplies seperate (or filtered!)
- **Vcc** - this is the voltage for the logic levels. Set to the voltage logic you'll be using on your microcontroller. E.g. for Arduinos, 5V is probably what you want. Can be 2.7V to 5.5V so good for 3V or 5V logic
- **GND** - This is the shared logic and motor ground. All grounds are connected

Signal in Pins

These are all 'Vcc logic level' inputs

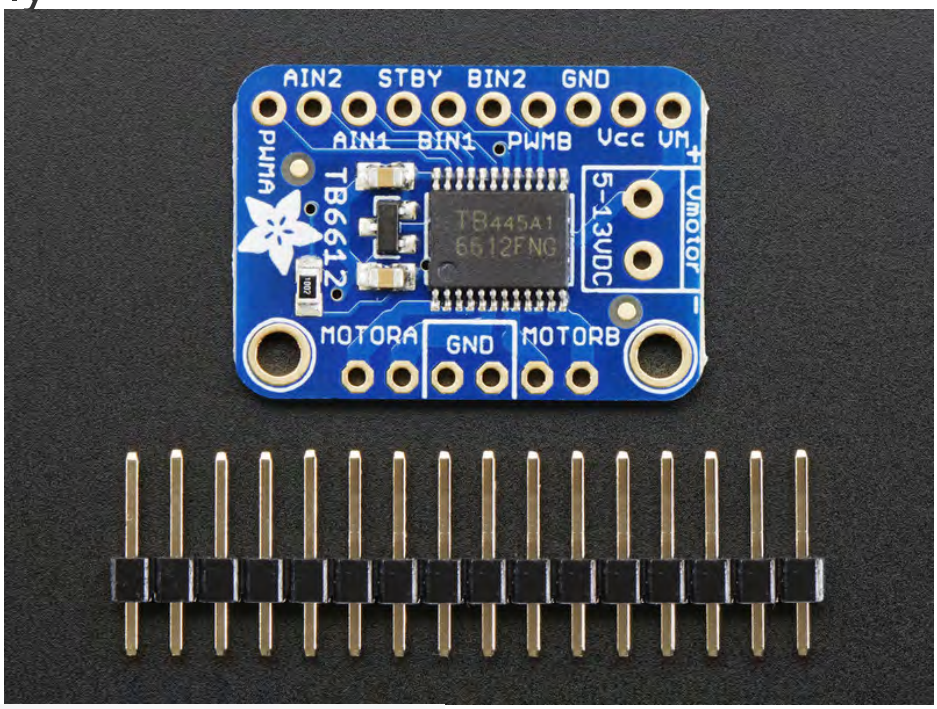
- **INA1, INA2** - these are the two inputs to the Motor A H-bridges
- **PWMA** - this is the PWM input for the Motor A H-bridges, if you dont need PWM control, connect this to logic high.
- **INB1, INB2** - these are the two inputs to the Motor B H-bridges
- **PWMB** - this is the PWM input for the Motor B H-bridges, if you dont need PWM control, connect this to logic high.
- **STBY** - this is the standby pin for quickly disabling both motors, pulled up to Vcc thru a 10K resistor. Connect to ground to disable.

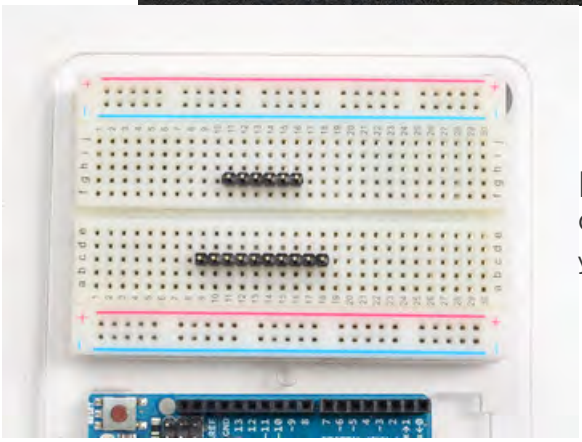
Motor Out Pins

These are 'Vmotor level' power outputs

- **Motor A** - these are the two outputs for motor A, controlled by INA1, INA2 and PWMA
- **Motor B** - these are the two outputs for motor B, controlled by INB1, INB2 and PWMB

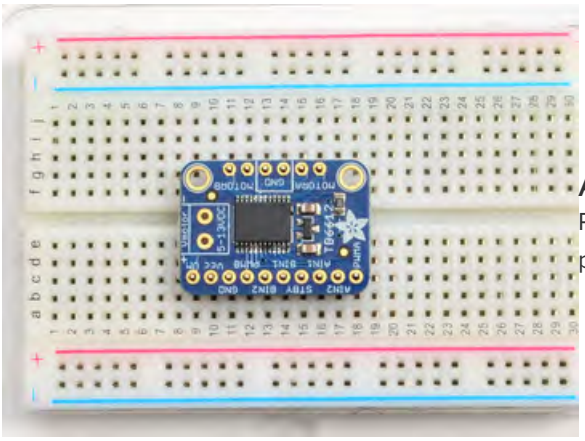
Assembly



- 

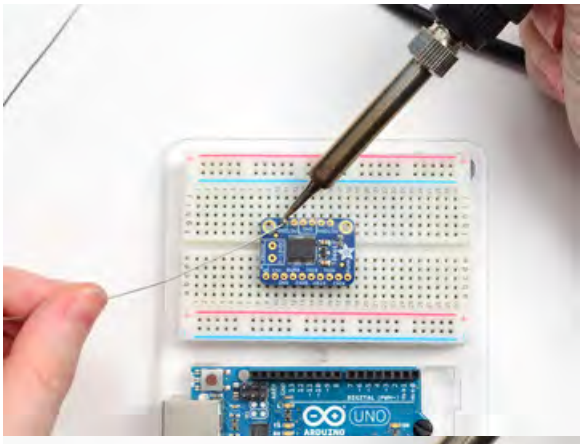
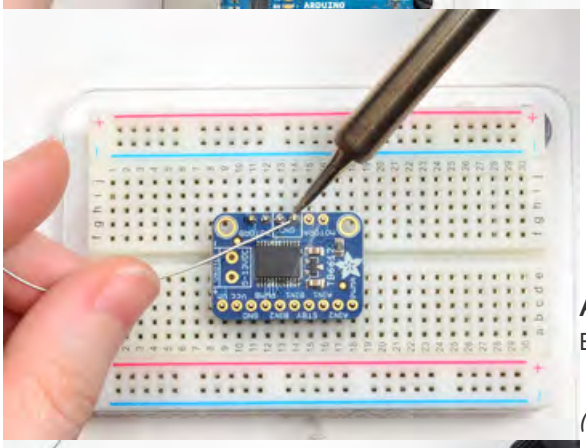
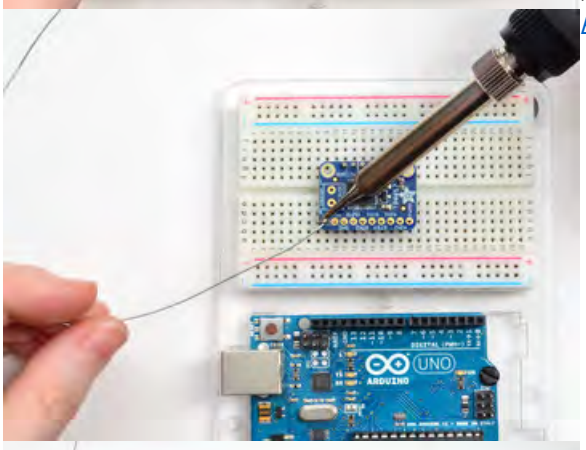
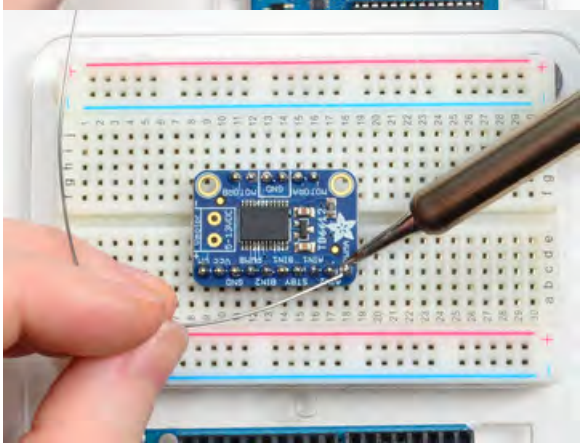
Prepare the header strip:

Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

- 

Add the breakout board:

Place the breakout board over the pins so that the short pins poke through the breakout pads

- 
- 
- 
- 

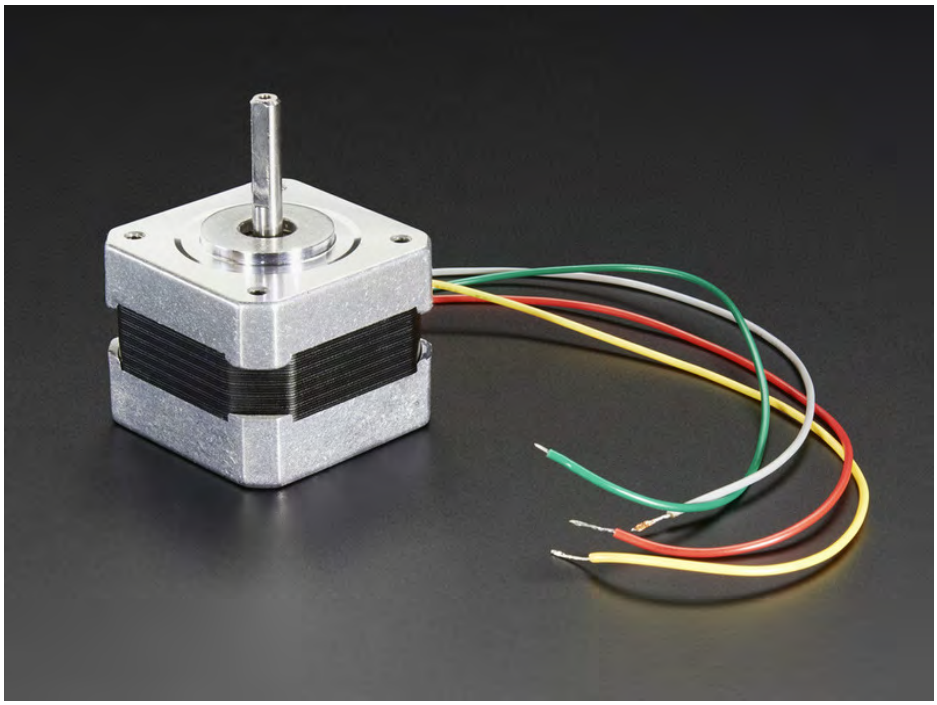
And Solder!

Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk).

Arduino Use: Stepper Motors

In this example we'll wire up and use a bi-polar stepper motor with recommended 12V motor voltage, and 200 steps per rotation.



Wiring

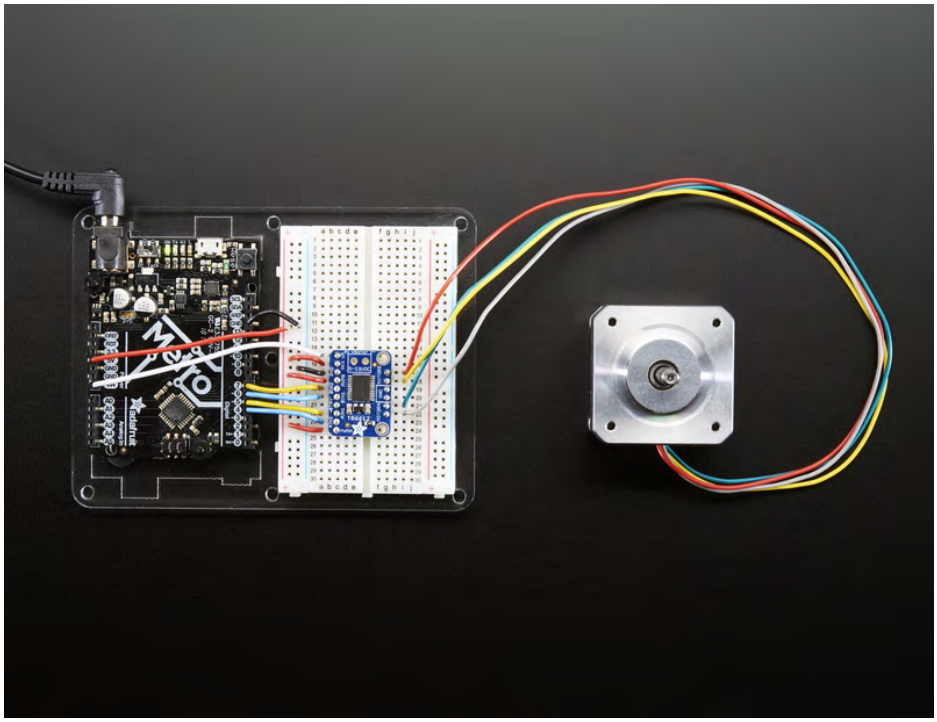
We'll wire it to a Metro, but you can use any microcontroller you like!

Connect:

- **Vmotor** to 12V (red wire)
- **Vcc** to 5V (orange wire)
- **GND** to ground
- **AIN2** to Digital 4
- **AIN1** to Digital 5
- **BIN1** to Digital 6
- **BIN2** to Digital 7
- **PWMA** and **PWMB** to Vcc (orange wire)

Then hook one stepper motor coil to **Motor A** (red and yellow) and the second coil to **Motor B** (green and gray/brown). If you have another motor, you'll need to experiment a little to figure out which wires are which coil. Check any documentation you have! You can use a multimeter to measure between wires, the ones with a small resistance between them are a pair to a coil, for example. If the motor is vibrating but not spinning, check all wires are connected and try flipping around a pair or rechecking the wire pairs.

If you have a unipolar motor, there will be a 5th or 6th wire that is the 'common' wire. Connect these wires to the GND pins in between the Motor A and B outputs on the breakout.



Software

We'll use the built-in Arduino Stepper library (https://adafru.it/eRw), but you can manually toggle the AIN1/AIN2/BIN1/BIN2 pins with your own favorite microcontroller setup

```
#include <Stepper.h>

// change this to the number of steps on your motor
#define STEPS 200

// create an instance of the stepper class, specifying
// the number of steps of the motor and the pins it's
// attached to
Stepper stepper(STEPS, 4, 5, 6, 7);


void setup()
{
  Serial.begin(9600);
  Serial.println("Stepper test!");
  // set the speed of the motor to 30 RPMs
  stepper.setSpeed(60);
}

void loop()
{
  Serial.println("Forward");
  stepper.step(STEPS);
  Serial.println("Backward");
  stepper.step(-STEPS);
}
```

Basically after you make the **Stepper** object with the 4 control pins, you can set the rotational speed (in RPM) with **setSpeed(*rpm*)** and then step forward or backwards with **.step(*steps*)** where *steps* is positive for 'forward' and negative for 'backward'

Appendix L

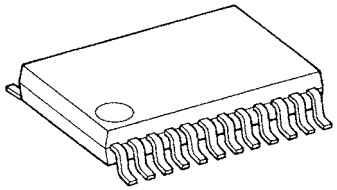
TB6612 driver

Toshiba Bi-CD Integrated Circuit Silicon Monolithic

TB6612FNG

Driver IC for Dual DC motor

TB6612FNG is a driver IC for DC motor with output transistor in LD MOS structure with low ON-resistor. Two input signals, IN1 and IN2, can choose one of four modes such as CW, CCW, short brake, and stop mode.



SSOP24-P-300-0.65A

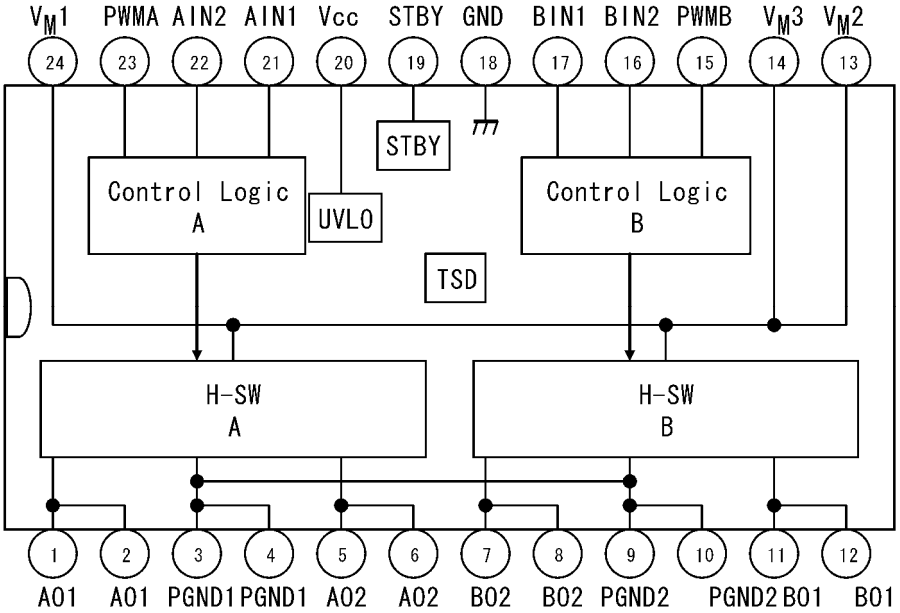
Weight: 0.14 g (typ.)

Features

- Power supply voltage: VM = 15 V(Max)
- Output current: I_{OUT} = 1.2 A(ave)/3.2 A (peak)
- Output low ON resistor: 0.5Ω (upper+lower Typ. @ VM ≥ 5 V)
- Standby (Power save) system
- CW/CCW/short brake/stop function modes
- Built-in thermal shutdown circuit and low voltage detecting circuit
- Small faced package(SSOP24: 0.65 mm Lead pitch)

* This product has a MOS structure and is sensitive to electrostatic discharge. When handling this product, ensure that the environment is protected against electrostatic discharge by using an earth strap, a conductive mat and an ionizer. Ensure also that the ambient temperature and relative humidity are maintained at reasonable levels.

Block Diagram



Pin Functions

No.	Pin Name	I/O	Function
1	AO1	O	ch A output 1
2	AO1		
3	PGND1	—	Power GND 1
4	PGND1		
5	AO2	O	ch A output 2
6	AO2		
7	BO2	O	ch B output 2
8	BO2		
9	PGND2	—	Power GND 2
10	PGND2		
11	BO1	O	ch B output 1
12	BO1		
13	VM2	—	Motor supply
14	VM3		
15	PWMB	I	ch B PWM input/200 kΩ pull-down at internal
16	BIN2	I	ch B input 2/200 kΩ pull-down at internal
17	BIN1	I	ch B input 1/200 kΩ pull-down at internal
18	GND	—	Small signal GND
19	STBY	I	“L” = standby/200 kΩ pull-down at internal
20	Vcc	—	Small signal supply
21	AIN1	I	ch A input 1/200 kΩ pull-down at internal
22	AIN2	I	ch A input 2/200 kΩ pull-down at internal
23	PWMA	I	ch A PWM input/200 kΩ pull-down at internal
24	VM1	—	Motor supply

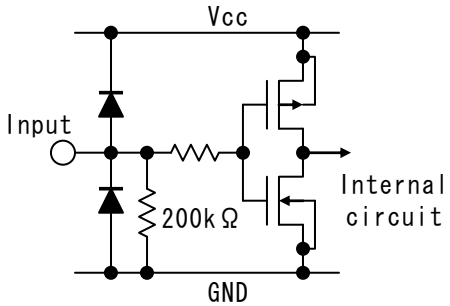
Absolute Maximum Ratings (Ta = 25°C)

Characteristics	Symbol	Rating	Unit	Remarks
Supply voltage	VM	15	V	
	V_{CC}	6		
Input voltage	V_{IN}	-0.2 to 6	V	IN1,IN2,STBY,PWM pins
Output voltage	V_{OUT}	15	V	O1,O2 pins
Output current	I_{OUT}	1.2	A	Per 1 ch
	I_{OUT} (peak)	2		tw = 20 ms Continuous pulse, Duty ≤ 20%
		3.2		tw = 10 ms Single pulse
Power dissipation	P_D	0.78	W	IC only
		0.89		50 mm × 50 mm t = 1.6 mm Cu ≥ 40% in PCB mounting
		1.36		76.2 mm × 114.3 mm t = 1.6 mm Cu ≥ 30% in PCB monting
Operating temperature	T_{opr}	-20 to 85	°C	
Storage temperature	T_{stg}	-55 to 150	°C	

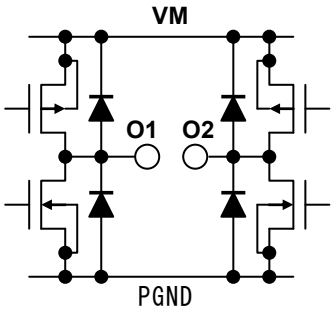
Operating Range (Ta = -20 to 85°C)

Characteristics	Symbol	Min	Typ.	Max	Unit	Remarks
Supply voltage	V_{CC}	2.7	3	5.5	V	
	VM	2.5	5	13.5	V	
Output current (H-SW)	I_{OUT}	—	—	1.0	A	VM ≥ 4.5 V
		—	—	0.4		4.5 V > VM ≥ 2.5 V Without PWM Operation
Switching frequency	f_{PWM}	—	—	100	kHz	

Input pin: IN1, IN2, PWM, STBY



Output pin: O1, O2

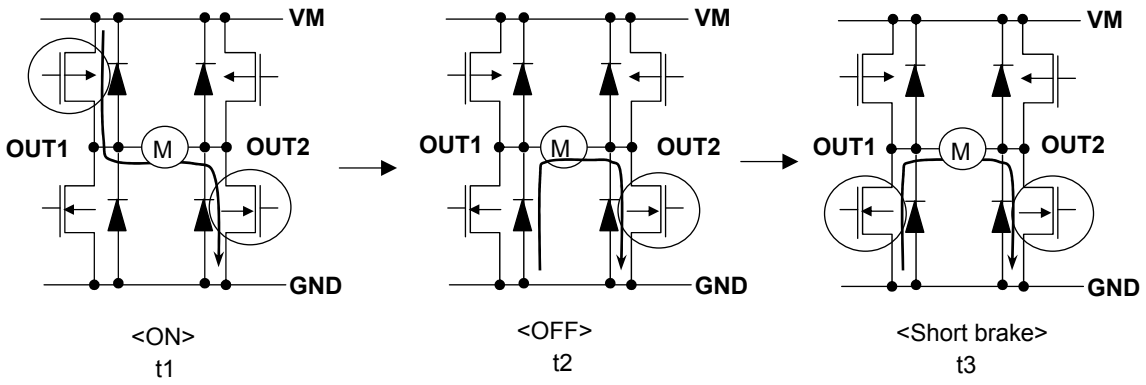


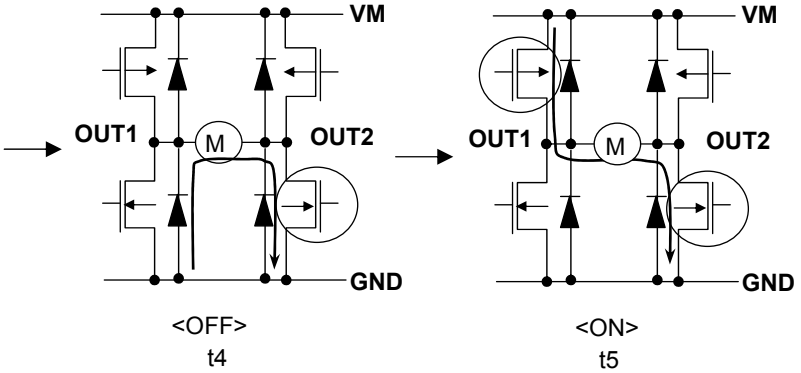
H-SW Control Function

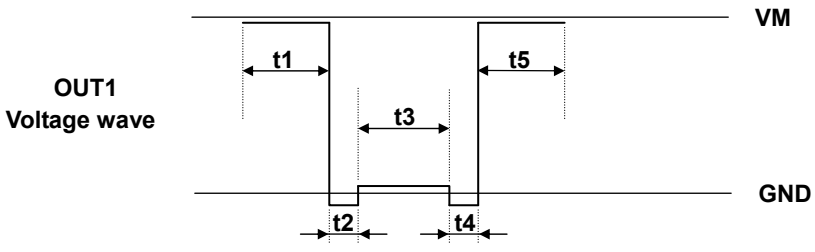
Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

H-SW Operating Description

・To prevent penetrating current, dead time t2 and t4 is provided in switching to each mode in the IC.



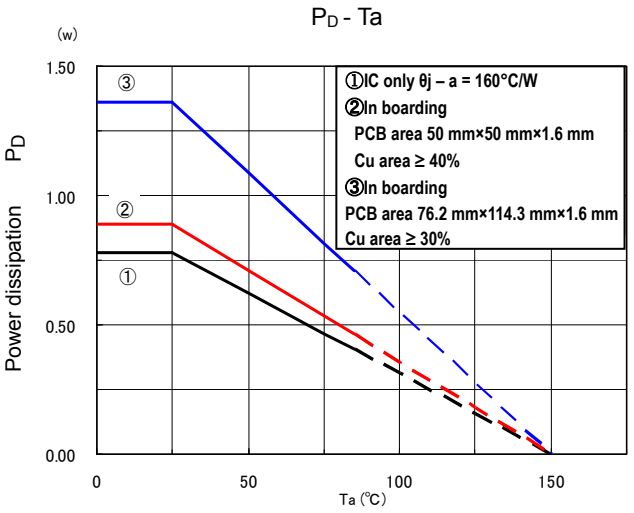


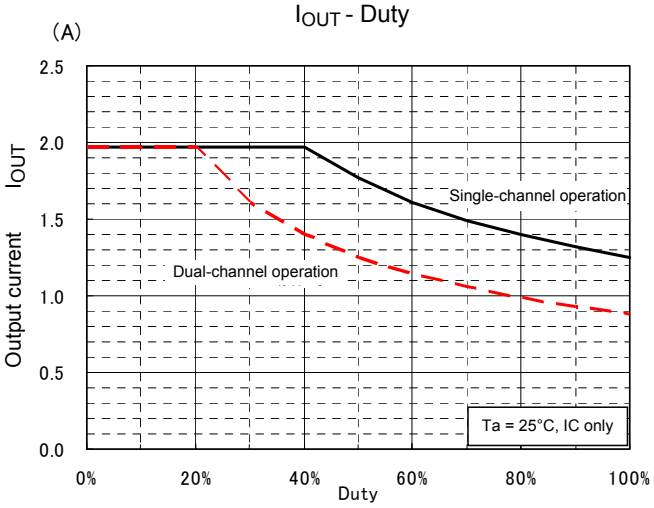


Electrical Characteristics (unless otherwise specified, Ta = 25°C, Vcc = 3 V, VM = 5 V)

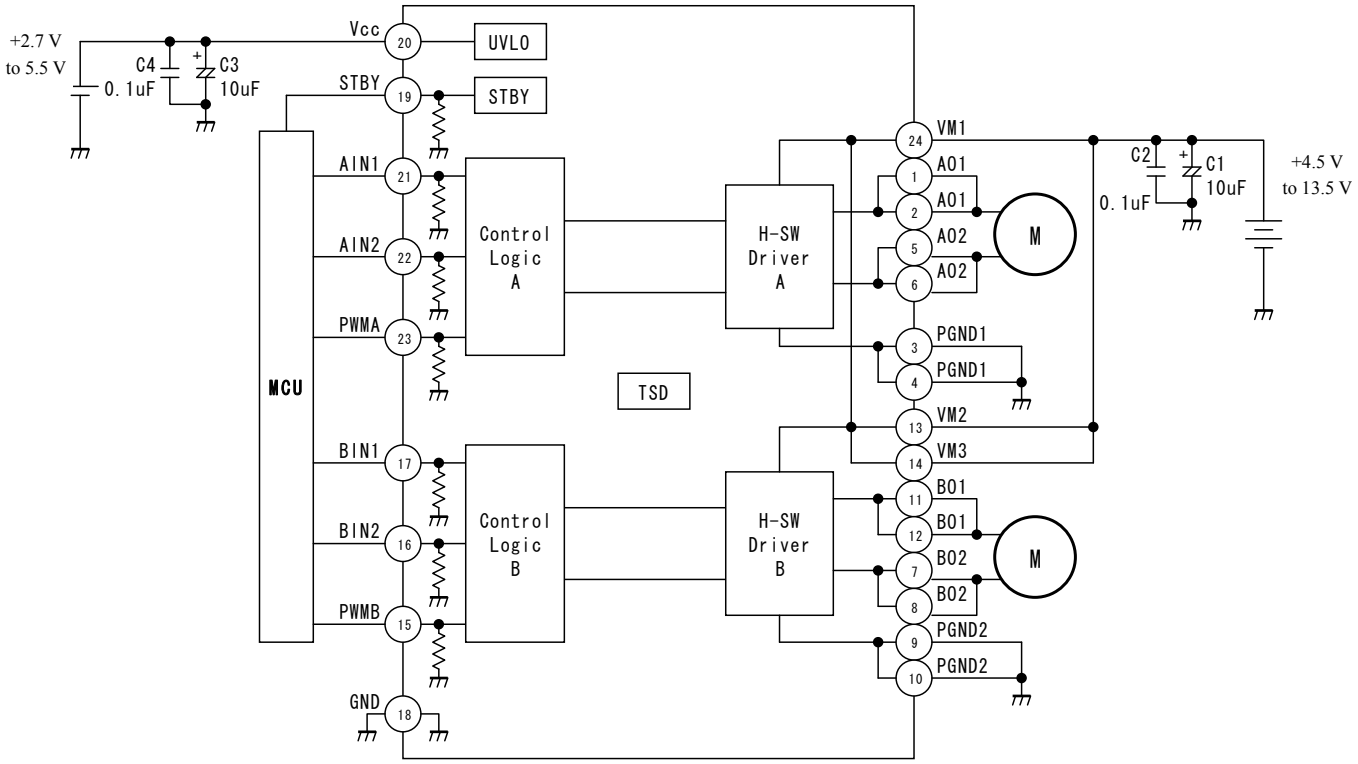
Characteristics	Symbol	Test Condition	Min	Typ.	Max	Unit
Supply current	I_{CC}(3 V)	STBY = Vcc = 3 V, VM = 5 V	—	1.1	1.8	mA
	I_{CC}(5.5 V)	STBY = Vcc = 5.5 V, VM = 5 V	—	1.5	2.2	
	$I_{CC(STB)}$	STBY = 0 V	—	—	1	μA
	$I_{M(STB)}$		—	—	1	
Control input voltage	V_{IH}	—	Vcc×0.7	—	Vcc+0.2	V
	V_{IL}		-0.2	—	Vcc×0.3	
Control input current	I_{IH}	V_{IN} = 3 V	5	15	25	μA
	I_{IL}	V_{IN} = 0 V	—	—	1	
Standby input voltage	$V_{IH(STB)}$	—	Vcc×0.7	—	Vcc+0.2	V
	$V_{IL(STB)}$		-0.2	—	Vcc×0.3	
Standby input current	$I_{IH(STB)}$	V_{IN} = 3 V	5	15	25	μA
	$I_{IL(STB)}$	V_{IN} = 0 V	—	—	1	
Output saturating voltage	$V_{sat(U+L)1}$	I_O = 1 A, Vcc = VM = 5 V	—	0.5	0.7	V
	$V_{sat(U+L)2}$	I_O = 0.3 A, Vcc = VM = 5 V	—	0.15	0.21	
Output leakage current	$I_{L(U)}$	VM = V_{OUT} = 15 V	—	—	1	μA
	$I_{L(L)}$	VM = 15 V, V_{OUT} = 0 V	-1	—	—	
Regenerative diode VF	$V_{F(U)}$	I_F = 1A	—	1	1.1	V
	$V_{F(L)}$		—	1	1.1	
Low voltage detecting voltage	UVLD	(Design target only)	—	1.9	—	V
Recovering voltage	UVLC		—	2.2	—	
Response speed	t_r	(Design target only)	—	24	—	ns
	t_f		—	41	—	
	Dead time H to L	Penetration protect time	—	50	—	
	Dead time L to H	(Design target only)	—	230	—	
Thermal shutdown circuit operating temperature	TSD	(Design target only)	—	175	—	°C
Thermal shutdown hysteresis	ΔTSD		—	20	—	

Target characteristics





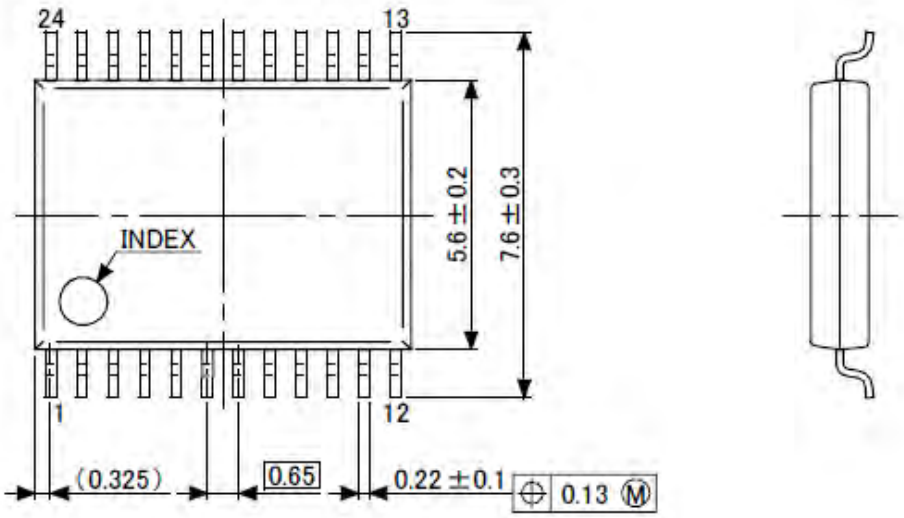
Typical Application Diagram

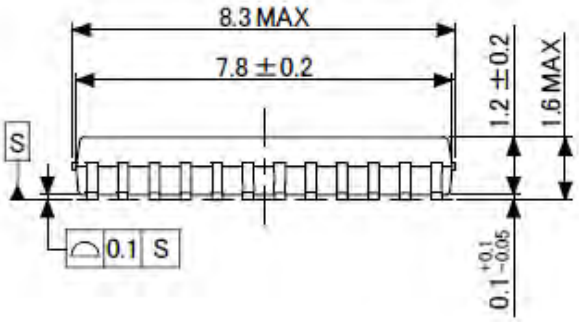


Note: Condensers for noise absorption (C1, C2, C3, and C4) should be connected as close as possible to the IC.

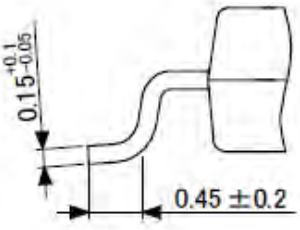
Package Dimennsions

SSOP24-P-300-0.65A "Unit : mm"





Detail of a terminal



Weght: 0.14 g (typ)

Notes on Contents

1. Block Diagrams

Some of the functional blocks, circuits, or constants in the block diagram may be omitted or simplified for explanatory purposes.

2. Equivalent Circuits

The equivalent circuit diagrams may be simplified or some parts of them may be omitted for explanatory purposes.

3. Timing Charts

Timing charts may be simplified for explanatory purposes.

4. Application Circuits

The application circuits shown in this document are provided for reference purposes only. Thorough evaluation is required, especially at the mass production design stage.
Toshiba does not grant any license to any industrial property rights by providing these examples of application circuits.

5. Test Circuits

Components in the test circuits are used only to obtain and confirm the device characteristics. These components and circuits are not guaranteed to prevent malfunction or failure from occurring in the application equipment.

IC Usage Considerations

Notes on handling of ICs

[1] The absolute maximum ratings of a semiconductor device are a set of ratings that must not be exceeded, even for a moment. Do not exceed any of these ratings.
Exceeding the rating(s) may cause the device breakdown, damage or deterioration, and may result injury by explosion or combustion.

[2] Use an appropriate power supply fuse to ensure that a large current does not continuously flow in case of over current and/or IC failure. The IC will fully break down when used under conditions that exceed its absolute maximum ratings, when the wiring is routed improperly or when an abnormal pulse noise occurs from the wiring or load, causing a large current to continuously flow and the breakdown can lead smoke or ignition. To minimize the effects of the flow of a large current in case of breakdown, appropriate settings, such as fuse capacity, fusing time and insertion circuit location, are required.

[3] If your design includes an inductive load such as a motor coil, incorporate a protection circuit into the design to prevent device malfunction or breakdown caused by the current resulting from the inrush current at power ON or the negative current resulting from the back electromotive force at power OFF. IC breakdown may cause injury, smoke or ignition.
Use a stable power supply with ICs with built-in protection functions. If the power supply is unstable, the protection function may not operate, causing IC breakdown. IC breakdown may cause injury, smoke or ignition.

[4] Do not insert devices in the wrong orientation or incorrectly.
Make sure that the positive and negative terminals of power supplies are connected properly.
Otherwise, the current or power consumption may exceed the absolute maximum rating, and exceeding the rating(s) may cause the device breakdown, damage or deterioration, and may result injury by explosion or combustion.
In addition, do not use any device that is applied the current with inserting in the wrong orientation or incorrectly even just one time.

Points to remember on handling of ICs

(1) Thermal Shutdown Circuit

Thermal shutdown circuits do not necessarily protect ICs under all circumstances. If the thermal shutdown circuits operate against the over temperature, clear the heat generation status immediately.

Depending on the method of use and usage conditions, such as exceeding absolute maximum ratings can cause the thermal shutdown circuit to not operate properly or IC breakdown before operation.

(2) Heat Radiation Design

In using an IC with large current flow such as power amp, regulator or driver, please design the device so that heat is appropriately radiated, not to exceed the specified junction temperature (Tj) at any time and condition. These ICs generate heat even during normal use. An inadequate IC heat radiation design can lead to decrease in IC life, deterioration of IC characteristics or IC breakdown. In addition, please design the device taking into considerate the effect of IC heat radiation with peripheral components.

(3) Back-EMF

When a motor rotates in the reverse direction, stops or slows down abruptly, a current flow back to the motor's power supply due to the effect of back-EMF. If the current sink capability of the power supply is small, the device's motor power supply and output pins might be exposed to conditions beyond absolute maximum ratings. To avoid this problem, take the effect of back-EMF into consideration in system design.

RESTRICTIONS ON PRODUCT USE

- Toshiba Corporation, and its subsidiaries and affiliates (collectively "TOSHIBA"), reserve the right to make changes to the information in this document, and related hardware, software and systems (collectively "Product") without notice.
- This document and any information herein may not be reproduced without prior written permission from TOSHIBA. Even with TOSHIBA's written permission, reproduction is permissible only if reproduction is without alteration/omission.
- Though TOSHIBA works continually to improve Product's quality and reliability, Product can malfunction or fail. Customers are responsible for complying with safety standards and for providing adequate designs and safeguards for their hardware, software and systems which minimize risk and avoid situations in which a malfunction or failure of Product could cause loss of human life, bodily injury or damage to property, including data loss or corruption. Before customers use the Product, create designs including the Product, or incorporate the Product into their own applications, customers must also refer to and comply with (a) the latest versions of all relevant TOSHIBA information, including without limitation, this document, the specifications, the data sheets and application notes for Product and the precautions and conditions set forth in the "TOSHIBA Semiconductor Reliability Handbook" and (b) the instructions for the application with which the Product will be used with or for. Customers are solely responsible for all aspects of their own product design or applications, including but not limited to (a) determining the appropriateness of the use of this Product in such design or applications; (b) evaluating and determining the applicability of any information contained in this document, or in charts, diagrams, programs, algorithms, sample application circuits, or any other referenced documents; and (c) validating all operating parameters for such designs and applications. **TOSHIBA ASSUMES NO LIABILITY FOR CUSTOMERS' PRODUCT DESIGN OR APPLICATIONS.**
- **PRODUCT IS NEITHER INTENDED NOR WARRANTED FOR USE IN EQUIPMENTS OR SYSTEMS THAT REQUIRE EXTRAORDINARILY HIGH LEVELS OF QUALITY AND/OR RELIABILITY, AND/OR A MALFUNCTION OR FAILURE OF WHICH MAY CAUSE LOSS OF HUMAN LIFE, BODILY INJURY, SERIOUS PROPERTY DAMAGE AND/OR SERIOUS PUBLIC IMPACT ("UNINTENDED USE").** Except for specific applications as expressly stated in this document, Unintended Use includes, without limitation, equipment used in nuclear facilities, equipment used in the aerospace industry, medical equipment, equipment used for automobiles, trains, ships and other transportation, traffic signaling equipment, equipment used to control combustions or explosions, safety devices, elevators and escalators, devices related to electric power, and equipment used in finance-related fields. **IF YOU USE PRODUCT FOR UNINTENDED USE, TOSHIBA ASSUMES NO LIABILITY FOR PRODUCT.** For details, please contact your TOSHIBA sales representative.
- Do not disassemble, analyze, reverse-engineer, alter, modify, translate or copy Product, whether in whole or in part.
- Product shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable laws or regulations.
- The information contained herein is presented only as guidance for Product use. No responsibility is assumed by TOSHIBA for any infringement of patents or any other intellectual property rights of third parties that may result from the use of Product. No license to any intellectual property right is granted by this document, whether express or implied, by estoppel or otherwise.
- **ABSENT A WRITTEN SIGNED AGREEMENT, EXCEPT AS PROVIDED IN THE RELEVANT TERMS AND CONDITIONS OF SALE FOR PRODUCT, AND TO THE MAXIMUM EXTENT ALLOWABLE BY LAW, TOSHIBA (1) ASSUMES NO LIABILITY WHATSOEVER, INCLUDING WITHOUT LIMITATION, INDIRECT, CONSEQUENTIAL, SPECIAL, OR INCIDENTAL DAMAGES OR LOSS, INCLUDING WITHOUT LIMITATION, LOSS OF PROFITS, LOSS OF OPPORTUNITIES, BUSINESS INTERRUPTION AND LOSS OF DATA, AND (2) DISCLAIMS ANY AND ALL EXPRESS OR IMPLIED WARRANTIES AND CONDITIONS RELATED TO SALE, USE OF PRODUCT, OR INFORMATION, INCLUDING WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, ACCURACY OF INFORMATION, OR NONINFRINGEMENT.**
- Do not use or otherwise make available Product or related software or technology for any military purposes, including without limitation, for the design, development, use, stockpiling or manufacturing of nuclear, chemical, or biological weapons or missile technology products (mass destruction weapons). Product and related software and technology may be controlled under the applicable export laws and regulations including, without limitation, the Japanese Foreign Exchange and Foreign Trade Law and the U.S. Export Administration Regulations. Export and re-export of Product or related software or technology are strictly prohibited except in compliance with all applicable export laws and regulations.
- Please contact your TOSHIBA sales representative for details as to environmental matters such as the RoHS compatibility of Product. Please use Product in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. **TOSHIBA ASSUMES NO LIABILITY FOR DAMAGES OR LOSSES OCCURRING AS A RESULT OF NONCOMPLIANCE WITH APPLICABLE LAWS AND REGULATIONS.**

Appendix M

Set screw hub - 5mm Bore

Figure M.1. Set screw hub datasheet.

Appendix N

Clip-part deassembled from Korg AW2G tuner

Figure N.1. The lower part of this tuner connected with the ball joint has been deassembled. For reference see [26]. The lower part has been used to push the piezo-element, onto the head stock of the guitar, see Appendix C.

Appendix O

Pitchclip 2 Clip-on tuner

Scale	12-note equal temperament
Detection range (sine wave)	AO (27.5 Hz) - C8 (4186 Hz)
Detection Accuracy	+/- 1 cent
Power Supply	CR2032 lithium battery x 1
Battery Life	Approximately 24 hours (A4 input)
Dimensions (W x D x H)	52 x 24 x 34 mm
Weight	17g (including batteries)

Figure O.1. This clip-on tuner model was used to calibrate our the piezo element's frequency detection [27] .

TRITA TRITA-ITM-EX 2021:24