Voice controlled guitar tuner

Igor Nikolaj Sok[1]

Faculty of Computer and Information science, Večna pot 113 Ljubljana, Slovenija
`is1946@student.uni-lj.si`

Abstract. This paper explores the pipeline for building a fully voice controlled guitar tuner. Many widely used tuners do their job well, but fall short in the scope of user interaction. Constantly having to manually tweak the tuner settings proves unintuitive and clunky while tuning an instrument. Additionally, support for alternative tunings is scarce. On the other hand, some tuners opt to accept no information about the tunings and just display the frequency that is being played, which can be challenging to understand for less experienced players. The tuner we developed aims to provide an intuitive tuning experience, without requiring any interactions with the keyboard or screen. Leveraging the unique characteristics of the guitar itself, everything is controlled by the users voice or certain frequencies produced by the instrument.

Keywords: Guitar · Tuning · Speech recognition · Fundamental frequency estimation (FFE)

1 Introduction

Tuning an instrument is often the first step a musician must take when interacting with his instrument. Whether in the context of practice, concert play, studio recordings, or playing in their free time. Tuning and therefore tuners are a fundamental part of the instrument playing experience. There are many tuners available on the market. In the scope of this project, we focused on guitar tuners, their shortcomings, and potential improvements that optimize the users interaction experience.

1.1 Guitar

The guitar is a string instrument that usually has 4 to 6 strings. We commonly place it in a subgroup called plucked string instruments. It consists of 3 distinct parts; the head, where the tuning screws are located, the neck with frets for playing different tones, and the body, where the strings are attached and the resonance lid and resonance opening are located. We produce tones by pressing on a fret with our left hand, while simultaneously plucking a string with our right. The tone produced is dependent on the frequency with which the string vibrates upon plucking it. We can affect this frequency through string choice, the topmost string having the lowest base frequency, while the bottommost has the highest, through fret presses, which shorten the string, making it vibrate with a higher frequency and the tension on the empty string. The tension on the empty strings is called the guitars tuning. Tunings have different names depending on the tone which the empty string would produce. The most common tuning is EADGBE. Many alternative tunings exist, such as DADGBE or CGDGBD. The tuning can be changed by tightening or loosening the screws located at the head of the guitar.

1.2 Tuners

Guitar tuners come in two main forms. The first are sound tuners, which through frequency analysis provide the user with some feedback on what tone is being played. The second are vibration tuners. These tuners are attached to the guitar, usually to the head or the body, and detect frequency through the vibration of the guitar, usually making them more robust and good in noisy environments. The first are much more widespread and popular. There are a few common issues present with sound tuners. The first is, that they often require user input to set, which string the user is trying to tune. This is often quite difficult for the user, as the guitar is placed in front of the user while playing and tuning, making it necessary for the player to shift its position around every time a new string is to be tuned. These tuners also seldom provide tuning support for any other tunings apart from the standard tuning (EADGBE). The other option are voice tuners, which don't accept any information about which string is being tuned and only display information about which frequencies are being detected. This can be very unintuitive and hard to understand for less experienced players and does not leverage any unique aspects of the guitar. The goal of this paper is to explore how a sound tuner can be made more intuitive, support any tunings, be controlled without requiring keyboard or screen inputs and present information to the user in a way, that does not require them looking at the screen.

2 Existing approaches

Two popular existing approaches are the mobile app GuitarTuna [4] and the web app Tuner-Online [8]. GuitarTuna offers tuning for predefined schemes, supporting the standard tuning and a few widely used alternatives, some of which are locked behind a paywall. When a string is played, a gauge indicates how close the tone is to the target pitch. However, this approach requires constant attention to the screen. Tuner-Online, on the other hand, does not restrict users to a single tuning scheme. Instead, it detects and displays the closest tone to the played note and indicates its deviation using a gauge. While it offers more flexibility, it also requires constant screen attention and may feel unintuitive for less experienced guitarists. The approach described in this article builds upon the strengths of these two methods while addressing their shortcomings. It aims to create a user-friendly guitar tuner that minimizes the need for screen interaction and provides flexibility by not limiting users to specific tuning schemes. A mind map of the project can be seen in figure4

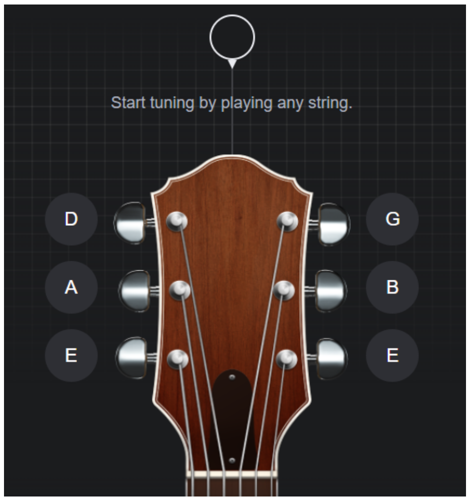


Fig. 1. GuitarTuna user interface

3 Methodology

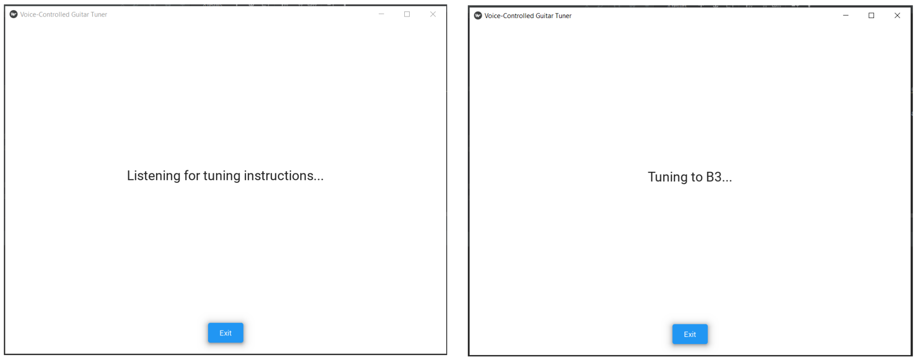
The tuner adheres to a few key design choices. The tuner should be intuitive to use, it should be hands free to use, it should leverage voice and guitar characteristics to navigate through different strings, it should support any tuning and it should be usable even if the user does not look at the screen. The app was built in accordance to these goals, using the Python programming language.

3.1 Speech recognition

Support for multiple tunings is implemented through speech recognition. Instead of limiting the user to some predefined tunings, the app allows the user to name any tuning, and the tuner adheres to the tuning named. The user can say a tone name, followed by its octave, which sets the tuner to listen for that specific tone. The speech recognition is done using the python SpeechRecognition[9] library. The library recognises voice using the Google Speech Recognition API.

3.2 Tone detection

Tone detection for instruments is an open issue in the field of Music Information Retrieval (MIR). The issue is not as simple as just running a Fourier transform and looking at the peak frequencies. That is because, different instruments also produce different harmonic frequencies that sound whenever a fundamental frequency is sounded. This transforms the problem from detecting frequencies to detecting fundamental frequencies of a given audio stream. Initial tone detections were done using Fast Fourier Transform (FFT), but higher and lower harmonics



Listening for tuning instructions...

Tuning to B3...

Fig. 2. The app in speech recognition mode

made this approach very noisy and unreliable. Later, i tired estimating fundamental frequencies by looking at the peaks of a Constant-q transform (CQT)[1]. This improved tone detection but proved too slow to be used in real-time. Finally i settled on a fundamental frequency estimation function called pyin from the librosa[7] library. The function implements the pYIN algorithm[6], a modification of the YIN algorithm[2]. In the first step of pYIN, F0 (fundamental frequency) candidates and their probabilities are computed using the YIN algorithm. In the second step, Viterbi decoding is used to estimate the most likely F0 sequence and voicing flags. The function returns all the fundamental frequency, the array indicating whether they are voiced or not and an array with the probability that they are voiced. I further filter these results, only accepting a fundamental frequency as voiced, if more than 10 of the nearby frequencies are voiced and if the probability that they are voiced exceeds 0.99. Tone estimation is performed on a running input stream every 0.05 seconds.

3.3 User interface

The user interface is build using the KivyMD library[5]. This library was chosen over simple python user interface libraries like tkinter, as it provides the option for mobile implementation and bundling the python program as an apk. The layout of the app is simple. It consists of a blank screen with some informative text in the middle, and a single button for exiting.

Tuning mode The initial state of the app involves the user saying a tone, they would like to tune into. Upon successful speech recognition, the app switches into tuning mode. Error handling when recognising tone names is done so, that if the wrong tone is selected, the user simply plays nothing for a few seconds, which makes the program enter voice recognition mode once again. In tuning mode, the app listens for fundamental frequencies and compares them to the target frequency.

Information presentation Following the design principle, that the user should have to focus as little as possible on the screen itself, the difference between the target frequency and the played frequency is presented in two ways. The first one is as a single colour, covering the entire app window, so that it is easily observed by the user, even at only a glance. Green colour represents the correct frequency. An incorrect frequency is presented in a different way, depending on whether the played frequency is lower or higher than the target frequency. If the played frequency is higher, it is presented as a shade of red, reaching its max if the played frequency is a half tone or more above the target frequency. Smaller differences are represented as a red-green gradient, corresponding to the difference. Lower played frequencies are represented in a similar way, the main difference being, that the colour used to indicate them is blue instead of red. The colours red and blue were chosen for higher and lower frequencies accordingly, due to their subconscious perceptions[3]. Blue is often associated with calm and slowness, making it a good candidate for a tone, which has a lower frequency than the target (the string vibrates slower). Red on the other hand, is associated with speed and action, making it a good candidate for a higher tone with a higher frequency (the string vibrates faster). The second way in which information is conveyed is through sound. Three distinct sounds are generated; a falling tone from E#4 to E4, a rising tone from Eb4 to E4 and a ka-ching like sound. All these sounds are created programmatically, using filters and the numpy library. The rising sound is played to the user, if the played tone is too low and he should tune up. The falling sound is played if the played tone is too high and the user should tune down. The ka-ching sound is played if the played tone is within an acceptable margin of the correct tone. The combination of these two information sources makes the tuning immersive, undisturbed and requiring minimal-to-none screen interaction. Additionally, the text in the middle of the window also updates with the corresponding information (tune up, tune down, in tune).

Navigation When the user decides that the string is tuned to their liking, they may elect to exit the tuner, or start tuning another string. To exit tuning mode and enter speech recognition mode, the characteristics of the guitar are leveraged. The neck of the guitar usually ends at the 12-th fret. If the 12-th fret is lightly pressed and the corresponding string is plucked, the tone that sounds is exactly an octave higher than the tone produced by the ope string. When the tuner hears this tone for the current string, it is assumed that the string is tuned correctly (as said tone could not be produced otherwise) and the program once again enters speech recognition mode. Here, the user can elect to start tuning a different string or say "exit" to close the program.

4 Results

The app works as intended and provides a smooth tuning experience. The information presentation as a combination of colour and sound feedback enables

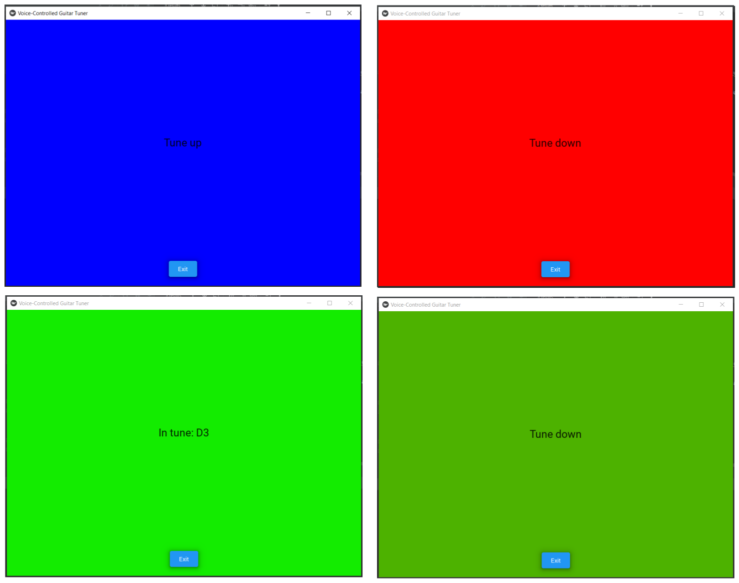


Fig. 3. Tuning mode when the detected tone is too low, to high, correct or slightly higher

the guitarist to look at the guitar neck and tuning screws instead of the screen, which enhances immersion and enables more efficient tuning. The tone detection is reliable and robust enough for accurate tuning. Voice recognition works as intended, but can take a few tries in a noisy environment. The app was tested with a classical guitar and a guitalele. The code and the demonstration video can be found here.

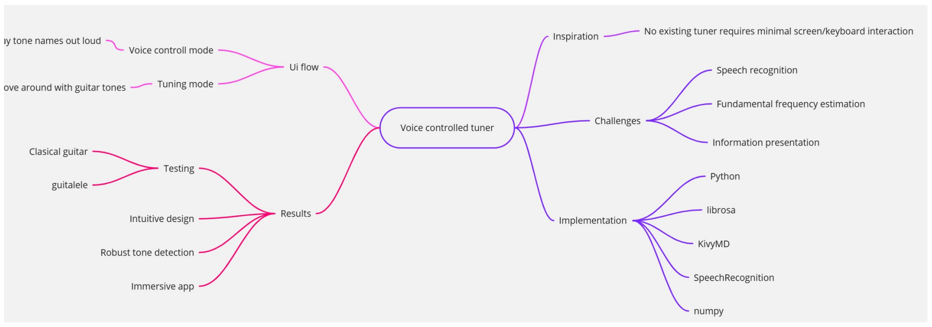


Fig. 4. A mind map detailing the main aspects of the project

4.1 Future improvements

The main improvement to the app would be porting it to a mobile platform. A mobile implementation would make it more convenient, as the user wouldn't have to rely for a nearby computer to tune his instrument. While the KivyMD library does provide android integration through its buildozer packaging tool, it is very buggy and unreliable when external libraries are used. The android build was attempted several times but the results were not satisfactory. A better approach would be to port the app to a native language, such as react native, which would east implementation and make any bugs simpler to resolve. Another improvement would be the use of a different speech recognition model. The current model requires an active internet connection to recognise speech correctly, which can be limiting in scenarios, where a stable connection is not available. Using an offline model, while putting more strain on the users device, would allow for more robustness and user freedom in such scenarios. Another extension of the app's functionality could be the addition of a simple song processing tool. This would enable a guitarist to use the app not only for tuning, but for a playing assistant, pointing out any missed tones for a selected song.

5 Discussion

Guitar tuners are often designed in a suboptimal manner, requiring excessive user interaction—such as selecting individual strings and inputting commands via a keyboard or screen between tuning each string. Many also lack support for non-standard tunings or fail to provide sufficient feedback on the tones being detected. The voice-controlled tuner recognizes tuning as a critical part of the playing experience and aims to streamline this process. By leveraging voice controls, color gradients, and sound feedback, the app creates a more immersive and efficient tuning experience. Its intuitive design requires no additional knowledge beyond specifying the desired tuning, making it accessible even to less experienced players. This ease of use fosters a positive attitude toward what is often perceived as a tedious aspect of playing guitar.

References

[1] Judith C Brown. "Calculation of a constant Q spectral transform". In: *The Journal of the Acoustical Society of America* 89.1 (1991), pp. 425–434.

[2] Alain De Cheveigné and Hideki Kawahara. "YIN, a fundamental frequency estimator for speech and music". In: *The Journal of the Acoustical Society of America* 111.4 (2002), pp. 1917–1930.

[3] Lu Geng, Xiaobin Hong, and Yulan Zhou. "Exploring the implicit link between red and aggressiveness as well as blue and agreeableness". In: *Frontiers in psychology* 11 (2021), p. 570534.

[4] *GuitarTuna*. URL: `https://yousician.com/guitartuna` (visited on 01/11/2025).

[5] *KivyMD documentation.* URL: `https://kivymd.readthedocs.io/en/latest/` (visited on 01/11/2025).

[6] Matthias Mauch and Simon Dixon. "pYIN: A fundamental frequency estimator using probabilistic threshold distributions". In: *2014 ieee international conference on acoustics, speech and signal processing (icassp)*. IEEE. 2014, pp. 659–663.

[7] Brian McFee et al. "librosa: Audio and music signal analysis in python". In: *Proceedings of the 14th python in science conference*. Vol. 8. 2015, pp. 18–25.

[8] *Tuner-online.* URL: `https://tuner-online.com/` (visited on 01/11/2025).

[9] Anthony Zhang. *SpeechRecognition.* `https://pypi.org/project/SpeechRecognition/`. 2025-11-1. 2024-29-12.