Solving the Floating Bridge Guitar Problem

Christopher Lim

May 2025

1 Introduction

The goal of this project is to develop an intelligent tuning system for guitars, specifically those with complex floating bridges like Floyd Rose systems, without requiring detailed manual measurements from users. The Floyd Rose guitar is difficult to tune because of the coupling effect between strings. Tuning one string will affect the tuning of all other strings in a non-linear fashion. By leveraging physically-based modeling and optimization techniques, we aim to build a solution that adapts to different guitar setups and tunings with limited prior data.

2 Code

The code consists of Python and JavaScript. Several libraries are needed for audio processing and numerical optimization. This includes scipy and CREPE. The program is segmented into three stages (simulation, calibration, and tuning) as described in more detail later. These files are named ”simulation.py,” ”calibration_data.py,” and ”tune_guitar.py” respectively. Flask is required to launch the tuning server. Ideal parameters are saved in comma separated files to reduce repeated calculations.

3 Context

In a non-floating bridge (normal) guitar, the strings are a fixed length and are attached to the body of the guitar directly. A Floyd Rose or “floating bridge” guitar is a special type of electric guitar with a tremolo system. The strings attach to a “floating” metal block which can move, allowing you to change the length of the strings and change the notes. That is, applying force to a metal bar will allow you to change the pitch of the note. This is used to create vibrato among other methods similar to instruments like the violin. For instance, pulling up will increase tension on the strings and pitch, and pushing down will decrease the tension and therefore the pitch. This creates complications when tuning the guitar, as tuning one string will change the tuning of all other strings. If we increase the tension of one string, the bridge will rotate forward. This decreases the length of the strings and decreases the tension of the strings.

4 Motivation

There are several factors that motivate the development of this program. Currently, the method to tune the guitar is using brute force. The strings are tuned to the correct pitch individually, and this process is repeated until the guitar is in tune. In other words, we converge on a solution naturally with the naive method, and each iteration has less of a difference than the past. There are several audiences that will benefit from an easier tuning method. Namely, beginners will find it easier to learn how to use this particular type of guitar. Natural factors like humidity and temperature affect the size of the wood and therefore the length of the strings, requiring frequent retuning. Oftentimes different pieces of music require different tunings, and changing the tuning of the guitar is equivalent to tuning the guitar from scratch in most cases. One such example of a common tuning is "drop-D," where the lowest string is de-tuned a whole tone from an E to a D. In particular, many artists bring multiple guitars in order to not change tunings, which is not feasible for most individuals. Oftentimes it is more convenient to use the fine tuners instead of the large tuners on the head of the guitar because they do not require any screws or tools. The fine tuners have limited range of motion, and the naive solution does not provide the most efficient solution. Therefore some guitar states are impossible to tune with only fine tuners. An improved solution would involve less rotations and find the most efficient way to obtain the correct tuning.

5 State of the Art

Currently there are no software solutions to this problem. There are several hardware solutions that restrict the movement of the system and therefore allow easier tuning of the guitar. However, these devices defeat the purpose of the Floyd Rose bridge and have several additional downsides. The Tremol-No does not support alternative tunings, and the EVH D-Tuna does not function with floating bridge guitars (bridges which can move both up and down). Both solutions are cost prohibitive and have an invasive installation.

6 Neural Network Solution

Currently it is possible to tune the guitar using the naive method, requiring multiple passes. We want to tune the guitar in one pass, meaning only tuning each tuning peg one time per string. This means we have to tune each string to some offset "wrong" pitch, and only after we tune the last string will all the strings suddenly be in tune. In particular, given some initial points we want to tune the guitar in "one-pass." Given an initial state of 6 tunings and n context points, we return a sequence of 6 points representing the tuning needed for each string at each intermediate step. We have two solutions in mind. In the first solution, we find the parameters from the context points using a neural network. Then we run the simulation with those specific parameters until we find the solution. Alternatively we run many simulations with various different parameters. Then we use the data to map the initial state along with the n context points to the solution with a neural network. This solution has several downsides. In particular, this solution is not generalizable to guitars with varying number of strings. Furthermore, the neural network is approximating a very complex function and a large portion of artificial data will need to be generated. We do not know ahead of time whether the data will be suitable for a given guitar.

7 General Solution

We propose a general solution which uses no neural network or data-based methods. The method has three components which combine to tune the guitar in one pass. First, we need a guitar simulation which calculates the offset and bridge displacement for a given set of frequencies or a change in one frequency. Then we need an optimization algorithm which finds the relevant parameters (string diameter, bridge length, spring tension) from minimal data points. Finally, we need an algorithm which finds the intermediate frequencies based on the offsets. This method generalizes to any guitar without training a neural network, although the optimization is not guaranteed to converge.

8 Physics

The bridge pivots around the two screws, where the knife edge creates hinge like joint. This is the circle in the figure above. On the top of the guitar the tension of the six strings pull the bridge forward. On the back of the guitar the springs pull the bridge backward to keep the bridge parallel at equilibrium. The length of the springs and the strings are variable, and therefore the tension of the springs and strings are also variable. Therefore, changing the tuning one of string, or increasing the tension of that string, will affect every other variable in the system, creating a coupled system. Figure one illustrates the forces at equilibrium.

8.1 String Equations

The following equations relate the frequency f of a tensioned string to its tension T. In particular, we can see that the square of the frequency is proportional to its tension, and the ratio is determined by the linear mass density mu and the length of the string L. We can also see that the force of a string is related to the displacement, or how much the string has been stretched. We can see that the force F and the ratio of the displacement over the original length $\frac{\Delta L}{L}$ is

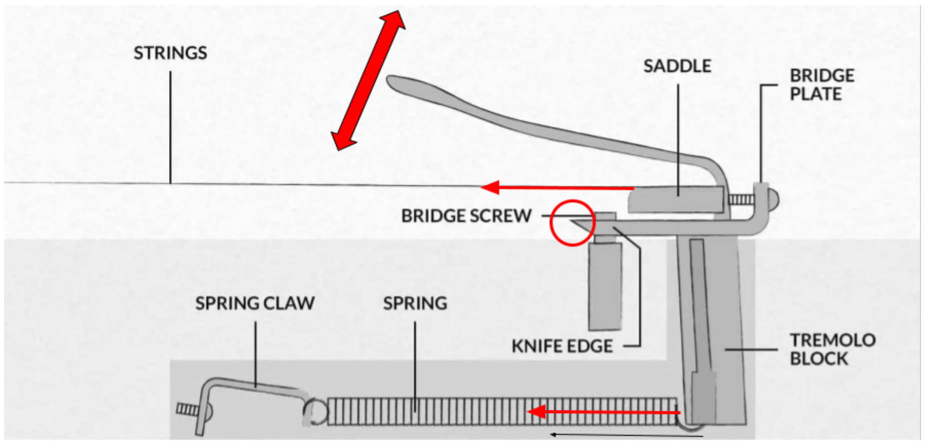

Figure 1: Diagram of the floating-bridge system

proportional with a ratio of Young's modulus Y times the cross-sectional area of the strings A. We can of course determine the area from the diameter d by $A = \pi(\frac{d}{2})^2$.

$$f = \frac{1}{2L}\sqrt{\frac{T}{\mu}}$$

$$F = \frac{YA\Delta L}{L}$$

8.2 Bridge Equilibrium Equations

The following equations relate the forces of the strings and the forces of the strings at equilibrium. In particular, we introduce two new variables. First, x represents the lengthening of the strings due to the bridge rotating backwards. Notice that we use x in the spring equation as well using the small-angle approximation. We also have the o_i parameters, which represents the offsets for each string. The offsets represent how much the string is stretched due to tuning a given peg.

$$T_{str}(x) = \sum_{i=1}^{6}(T_{i,0} + k_i \cdot (x + o_i))$$

$$T_{spr}(x) = T_{spr0} - k_{spr} * x$$

$$\tau_{net}(x) = T_{str}(x) - T_{spr}(x)$$

$$\tau_{net}(x) = 0$$

8.3 Assumptions

We made several assumptions about the physical laws of the system. In particular, we assumed Hooke's law for both the springs and the strings. This means that non-linearities of the strings will not be captured. We also assume that the strings have a constant linear mass density, which is reasonable. Finally, we assume that the displacement in the strings due to bridge rotation is equal to the (negative) displacement in strings. This small-angle approximation is acceptable because we are only interested in the effective spring constants from our assumptions, not necessarily the real-world constants.

9 Simulation Parameters

In order to run the simulation, we need to decide on several parameters that govern the physical behavior of a guitar. Based on our model, there are several sets of parameters which are equivalent. This means that we can determine one set of parameters from another. We will discuss two such possible implementations and their differences. Also note that we parametrize some physical constants to compensate for our assumptions. The first solution has 5 + 6 * 3 = 23 parameters and the second solution has 3 + 6 * 4 = 27 parameters.
Measurement Parameters

1. String lever length
2. Spring lever length
3. Young's modulus for steel
4. Spring constant
5. String point density
6. For each string
   (a) Diameter
   (b) Scale length
   (c) Balanced frequency

Equation Parameters

1. String lever length
2. Spring lever length
3. Spring constant
4. For each string
   (a) Stiffness

(b) Scale length

(c) Linear mass density

(d) Equilibrium tension

We note several tradeoffs between these two sets. First, the measurement parameters are easier to interpret physically, and it is easy to tell if a parameter is unreasonable. Whereas for the equation parameters the linear mass density and equilibrium tension is not widely available for many guitars. The measurement parameters also allow for better constrained optimization but requires expert knowledge to initialize. The measurement parameters are fewer, which also means we need less data to optimize for them. Both of these parameter sets have less than $6^2 = 36$ parameters. This is the number of parameters if we were to consider the pairwise effect on one string on another. We hope that this reduction in parameters although increases the complexity, means less data is needed to optimize.

10 Simulation

As described earlier, the simulation has several main functions. We model turning a tuning peg as an offset, which representing shortening the string. First, we need to find the equilibrium bridge rotation given the offsets. This can be solved analytically since we assume Hooke's law, however in the future we will need the bisection or Newton's method to solve for non-linear string effects. Second, we need to be able to determine the offsets when tuning all strings to specific frequencies. Finally, we need to determine the offsets when tuning one string to a specific frequency. We find these offsets by running optimization algorithms on the offsets with the objection function being the difference in frequencies.

11 Parameter Calibration

The second component of the algorithm is finding the optimal parameters. In essence, we need to check whether the parameters chosen match the user's guitar. However, we cannot directly measure any physical parameters. We instead infer from the several data-points, where each data-point is a grouping of 6 string frequencies. Each grouping is separated by turning one tuning peg or tuning one string each time. We generate the artificial data by running the simulation with the prior variables. These variables are selected by averaging real guitar measurements. We compute the squared loss between the artificial data and the real data, measuring the likelihood that the prior parameters match the real parameters. To minimize the loss, we use constrained optimization where we allow the parameters to vary 80% from the prior parameters. We then run the multi-variable optimization algorithm on the parameters.

12 Parameter Loss Function

We need to estimate the loss between the real guitar parameters and the simulation parameters with limited data. We propose an algorithm which prevents overfitting during training. We perform a random walk on our dataset, or randomly sample the next data-point k-units ahead or k-units behind, where k belongs to a gaussian distribution. This gives a sequence of six-tuples. If we define the distance in time between each data-point to be k, then when simulating each data-point we need to tune at most k of the strings. This method can generate an exponential number of sequences from a limited number of data-points, and the randomness prevents overfitting.

13 Finding Intermediate Tunings

Finally, we need to use the simulation of the guitar to tune a real guitar. Recall the coupled nature of the floating-bridge, and how the final tuning of a string is not equal to the intermediate tuning of a string. This is because when tuning the kth string, all strings $> k$ will affect the tuning of k. However, we know that the offsets or displacement from turning a tuning peg is invariant. In other words, if you knew how much to turn each peg ahead of time, you could solve the system in one pass. The algorithm performs this exact calculation. We first solve for the offsets of the desired tuning. Then we solve for the offsets of the initial tunings. Finally, in any order we can replace one of the initial offsets with the final offset. This will produce a sequence of initial tunings which will converge to the desired tuning, assuming the simulation is accurate.

14 Optimization Algorithms

In most components of this algorithm, optimization algorithms are utilized. For the simulation itself, we use Nelder-Mead. This is a heuristic direct search method, and does not use the gradient. Other methods could be used, but this method ensures it is likely for the algorithm to find the closest offsets to the initial offsets which satisfy the necessary frequencies. In the case of tuning one string, we use a single variable method called Brent's Algorithm. Brent's Algorithm tries to use interpolation (secant or quadratic) for faster convergence but falls back to bisection when necessary to ensure the method always makes progress. Note the objective function uses optimization algorithms itself, so it is difficult to explicitly find the gradient. Finally, the parameter sweep uses the L-BFGS-B (Limited Memory Broyden–Fletcher–Goldfarb–Shanno Algorithm with bounds). This quasi-Newton algorithm had the lowest loss compared to other methods like differential evolution and Bayesian estimation. Differential evolution was too expensive for a layered optimization problem, and the Bayesian estimation algorithm could not escape the initial local minimum prior distribution. For other guitars, it may be effective to run Bayesian estimation to find the best starting point for L-BFGS-B. Also note in the simulation parameter sweep, it is not possible to know beforehand whether a set of parameters is valid. This means that we can obtain a negative tension at equilibrium, in which case we return the maximum penalty.

15 Supplemental Algorithms

15.1 Karplus-Strong

We use the Karplus-Strong algorithm to efficiently generate string-like timbre when playing back audio. This allows the user to use their hearing to tune the guitar. This physically-based algorithm stores a initial burst of noise in a buffer (simulating the pluck). It then continuously loops and updates with new values, simulating vibration and decay. Finally, it repeats values after a delay that matches the period of the desired frequency.

15.2 Pitch Estimation

There are several options to find the pitch of notes in sound waves. We tested several methods including time-domain methods like Autocorrelation and Cumulative Mean Normalized Difference (CMND) methods, which are not accurate with noisy signals and octaves. Frequency-Domain Methods like the Fast Fourier Transform (FFT) require a fixed-number of samples and are also susceptible to spectral leakage. We decided to use machine learning methods like the Python library CREPE, because it is trained on real data and is resilient to noise. We use other hybrid methods like YIN in the JavaScript front-end.

16 Results

In parameter optimization, we achieved sub-one hertz accuracy with 4 data points from a real guitar. Currently we are not able to test the algorithm on more than one guitar. Qualitatively, we can explore a tuning scenario where the naive method fails. In particular, we analyze a common scenario where the E-string is too sharp. We perform the naive and improved algorithms on the same initial tunings and compare their errors. The following results are achieved using 20 data points.

String	True Tuning (Hz)	Naive (Hz)	New (Hz)
1 (E4)	329.63	337.84	326.52
2 (B3)	246.94	258.22	244.37
3 (G3)	196.00	209.16	192.92
4 (D3)	146.83	155.70	145.79
5 (A2)	110.00	116.54	109.10
6 (E2)	82.41	84.07	81.64

Table 1: True tuning vs. Naive and New algorithm outputs

String	Abs Error (Naive)	Abs Error (New)
1 (E4)	8.21	3.11
2 (B3)	11.28	2.57
3 (G3)	13.16	3.08
4 (D3)	8.87	1.04
5 (A2)	6.54	0.90
6 (E2)	1.66	0.77

Table 2: Absolute errors between algorithm outputs and true tuning

String	Rel Error (Naive)	Rel Error (New)
1 (E4)	2.49%	0.94%
2 (B3)	4.57%	1.04%
3 (G3)	6.71%	1.57%
4 (D3)	6.04%	0.71%
5 (A2)	5.95%	0.82%
6 (E2)	2.02%	0.93%

Table 3: Relative errors between algorithm outputs and true tuning

17 Summary

We created an original method to solve an intricate real-world problem. We wanted to tune the floating-bridge guitar in one pass, however we did not want to gather a large number of real-world data points. Thus we created a simulation to make artificial data. However, we wanted a fully general method which worked on multiple guitars (and variable number of strings) without performing physical measurements of guitars. Thus, we needed to optimize for the correct physical parameters by computing the likelihood by comparing the difference in data.

18 Optimization and Improving Accuracy

There are several optimizations which would speed up the runtime of the algorithm and make the model more accurate. First, We would like to reduce number of parameters in the model. This can be done by finding dependent parameters in the equations or making more assumptions. We would also like to increase the number of data points for all guitars, then fine tune on a specific guitar. This would allow the user to not have to input many data-points to calibrate their guitar. We will also consider replacing layered optimization with constrained optimization. Currently many of the objective functions require optimization algorithms themselves, which leads to costly evaluation. It may be faster to solve for several variables at the same time, and rewrite constrained optimization as unconstrained using the Lagrangian. For example, we would solve for the offsets and bridge angle at the same time, with a penalty term for non-equilibrium. In the same vein we also would like to make functions differentiable and use gradient-based optimization methods for significantly faster convergence. Finally, we would like to capture the non-linearities of string forces with respect to displacement. This would increase the number of parameters by 6, but would allow for less bias.

19 Future Work

There are several key improvements which will be made in the near future. First, we will use a more robust pitch finding technique. Currently, general pitch estimation is sensitive to noise and makes it difficult to tune. We will explore specific algorithms designed for tuning guitar frequencies. Next, we will use a pitch finding method which implements polyphonic pitch detection. This will speed up calibration by six times, allowing the user to input the frequencies of all six strings at once. We hope to test the algorithm on a larger set of guitars and see if the parameters will converge. We also will explore simpler methods without physically-based parameters. For example, we could explore a matrix which represents the pair-wise effect of one string on another. Real-time calibration would allow the algorithm to improve while the user is performing a regular tuning. And finally, we would like to generalize the algorithm to guitars with a variable number of strings.

20 Long Term Vision

We would like to integrate this algorithm into more useful devices for musicians. This includes publishing an app on an app store or website, embedding into a hardware guitar pedal, or embedding into a hardware clip-on tuner. In order to integrate this algorithm on embedded system, we would need to reduce the complexity of algorithms greatly.