**DeepStrum: An AI powered Guitarist Robot**

Kanishq Gupta Kartikeya Srivastava

Amber Tomar , Aditya Khariwal

**Intro:**

The guitar is one of the most interesting and renowned instruments. It has its own fan following and complications when it comes to learning to play this beautiful instrument. We would like to preserve the authenticity of the instrument and help guitar enthusiasts enjoy this without bothering themselves with the tedious job that is to learn playing it. This would also ease the complications of potential guitar learners as guitar tablature can help them with the task of searching for the tablature all over the web just to often be unable to find it.

We, therefore, wish to create a fully automated guitar-playing bot which would only require the user to request a song and then sit back while the robot does the rest. It aims to process the audio for simplification and de-noising to be able to form the tablature next. Finally, it would use this generated tablature to play the requested song.

The problem of creating a guitar-playing bot project involves developing an algorithm that can accurately simulate the techniques and sounds of a human guitar player. This is a challenging task, as guitar playing involves a complex set of physical movements and requires a deep understanding of music theory and engineering.

**Literature Review:**

A traditional split for MSS (Music Source Separation) methods is between spectrogram based and waveform-based models. The former includes models like Open-Unmix, a biLSTM with fully connected that predicts a mask on the input spectrogram or D3Net which uses dilated convolutional blocks with dense connections. More recently, using complex-spectrogram as input and output was favored as it provides a richer representation and removes the topline given by the Ideal-Ratio-Mask. The latest spectrogram model, Band-Split RNN, combines this idea, along with multiple dual-path RNNs, each acting in carefully crafted frequency band. It currently achieves the state-of-the-art on MUSDB with 8.9 dB. Waveform based models started with Wave-U-Net, which served as the basis for Demucs, a time domain U-Net with a bi-LSTM between the encoder and decoder. Around the same time, Conv-TasNet showed competitive results using residual dilated convolution blocks to predict a mask over a learnt representation. Finally, a recent trend has been to use both temporal and spectral domains, either through model blending, like KUIELAB-MDX-Net, or using a bi-U-Net structure with a shared backbone as Hybrid Demucs. Hybrid Demucs was the first ranked architecture at the latest MDX MSS Competition, although it is now surpassed by Band-Split RNN.

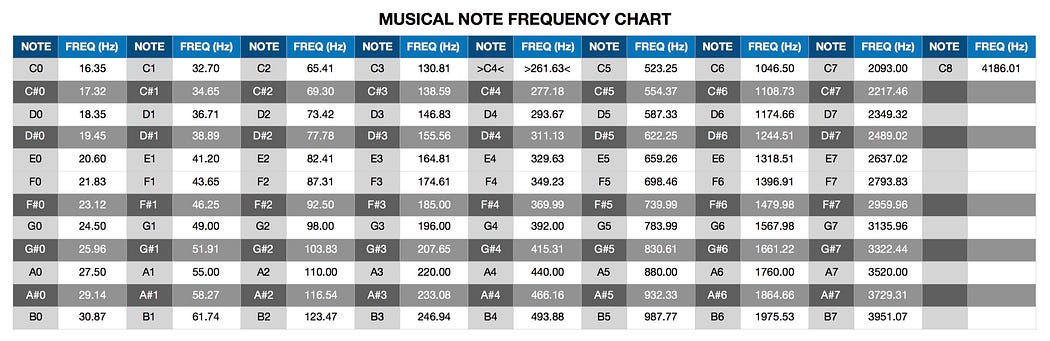
Both D3Net and Demucs were offering strong performance on MUSDB only. Band-Split RNN introduced an unsupervised augmentation technique requiring only mixes to improve its performance by 0.7 dB of SDR (Signal to Distortion Ratio).

**Methodology:**

**Audio Processing:**

Audio processing can be a challenging task but with the help of the python library ‘Librosa’ it is made easier. ‘Librosa’ is a powerful library with quite a lot of useful features. We are first required to separate the different components of the audio file for assuring the tablature generation. Therefore, the initial audio is processed to generate different files consisting of isolated vocals, drums and bass in turn leaving the initial file to contain a simplified version of the song. This audio file is then de-noised for better results.

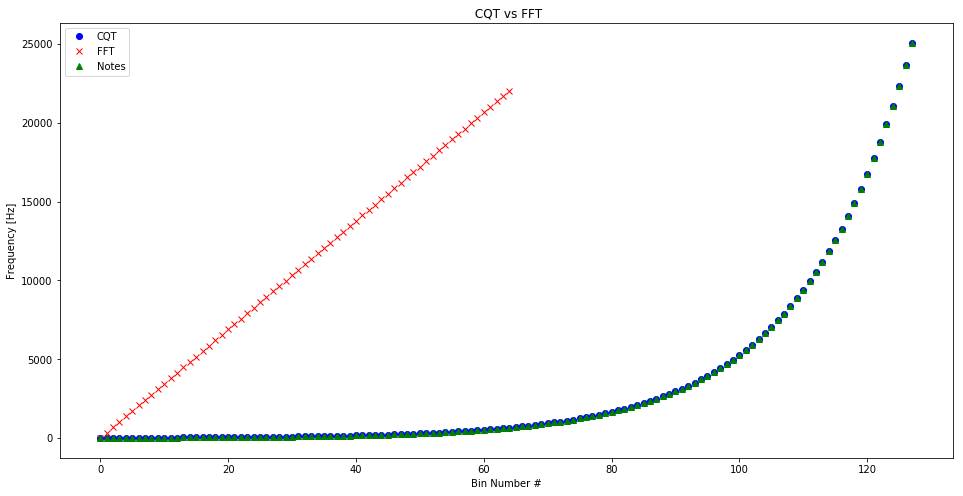
The step next to audio processing is the identification of musical notes for tablature generation, musical notes can be distinguished on the basis of frequency. Frequency identification can be helped by tools like Constant-Q Transform and Fourier Transform. We have used Constant-Q Transform over Fourier Transform. In order to understand the benefits of using the Constant-Q transform over the Fourier transform to select frequencies and create our input images, we must examine how musical notes are defined:



Since an octave spans twelve notes, we know that the frequency must double every twelve notes, which can be represented by the following formula:



By plotting this relationship, we can see that the graph below displays an exponential curve:



In the above graph, the musical notes are completely coincident with the results provided using Constant-Q Transform. Due to this exponential nature, the Constant-Q transform is better suited for fitting musical data than the Fourier transform, as its output is amplitude versus the log frequency. Also, the Constant-Q transform’s accuracy is analogous to the logarithmic scale and mimics the human ear, having a higher frequency resolution at the lower frequencies and a lower resolution at the higher frequencies.

**Stationary Noise Reduction**

* The basic intuition is that statistics are calculated on each frequency channel to determine a noise gate. Then the gate is applied to the signal.
* This algorithm is based (but not completely reproducing) on the one outlined by Audacity for the noise reduction effect.
* The algorithm takes two inputs:
  + 1. A noise clip containing prototypical noise of clip (optional)
    2. A signal clip containing the signal and the noise intended to be removed

**Steps of the Stationary Noise Reduction algorithm**

1. A spectrogram is calculated over the noise audio clip
2. Statistics are calculated over spectrogram of the noise (in frequency)
3. A threshold is calculated based upon the statistics of the noise (and the desired sensitivity of the algorithm)
4. A spectrogram is calculated over the signal
5. A mask is determined by comparing the signal spectrogram to the threshold
6. The mask is smoothed with a filter over frequency and time
7. The mask is applied to the spectrogram of the signal, and is inverted If the noise signal is not provided, the algorithm will treat the signal as the noise clip, which tends to work well

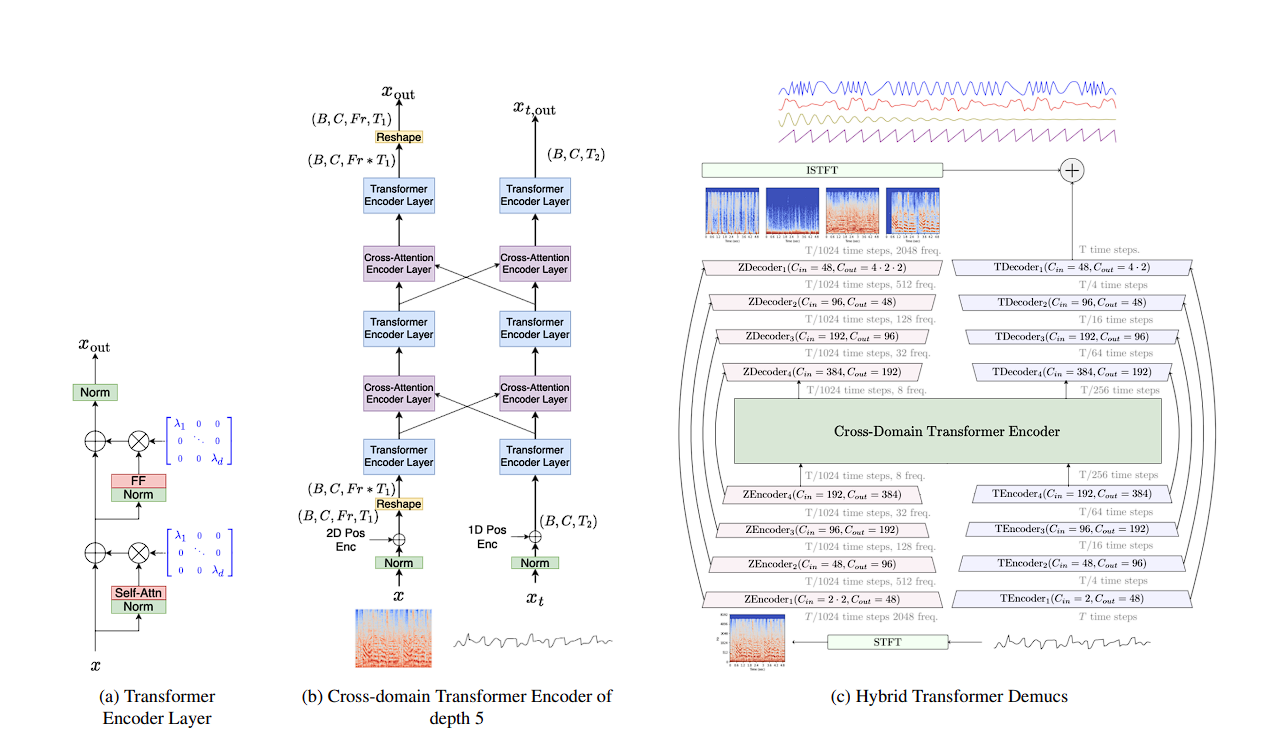
**Hybrid Transformers for Music Source Separation**

Hybrid transformers are a type of neural network architecture is applied to music source separation tasks. Music source separation refers to the process of separating individual sound sources, such as vocals, drums, and instruments, from a music mixture.

We introduce the Hybrid Transformer Demucs model, based on Hybrid Demucs. The original Hybrid Demucs model is made of two U-Nets, one in the time domain (with temporal convolutions) and one in the spectrogram domain (with convolutions over the frequency axis). Each U-Net is made of 5 encoder layers, and 5 decoder layers. After the 5-th encoder layer, they are summed before going into a shared 6-th layer. Similarly, the first decoder layer is shared, and its output is sent both the temporal and spectral branch. The output of the spectral branch is transformed to a waveform using the iSTFT(Short Time Fourier Transformer) and then summed with the output of the temporal branch to give the prediction.

Hybrid Transformer Demucs keeps the outermost 4 layers as is from the original architecture, and replaces the 2 innermost layers in the encoder and the decoder, including local attention and bi-LSTM( long short-term memory networks) , with a cross-domain Transformer Encoder. The advantage of using this is that it treats in parallel the 2D signal from the spectral branch and the 1D signal from the waveform branch, so it don’t requires careful tuning of model parameters and the cross-domain Transformer Encoder can work with heterogeneous data shape, making it a more flexible architecture.

The architecture of Hybrid Transformer Demucs



The self- attention Encoder layer is normalized before the Self-Attention and Feed-Forward operations. The two first normalizations are layer normalizations (each token is independently normalized) and the third one is a time layer normalization (all the tokens are normalized together). The input/output dimension of the Transformer is 384. The attention mechanism has 8 heads and the hidden state size of the feed forward network is equal to 4 times the dimension of the transformer. The cross-attention Encoder layer is the same but using cross-attention with the other domain representation. In the middle, a cross-domain Transformer Encoder of depth 5 is depicted.

This architecture benefits from large training dataset and outperforms Hybrid Demucs by 0.45 dB. Thanks to sparse attention techniques, model have scaled to an input length up to 12.2 seconds during training which led to a supplementary gain of 0.4 dB.

**Tempo Calculation and its importance:**

Tempo is the speed or pace of a musical piece, typically measured in beats per minute (BPM). It refers to how quickly or slowly the underlying pulse or rhythm of the music is played, and can vary depending on the style and genre of the music. Since, the model needs to be precise for such a challenging task it requires the knowledge of tempo of the song to process the time required to play each note. Without the knowledge of tempo, it would not be possible to make a model capable of playing a song with all its dynamics.  
‘Librosa’, being the powerful library it is, has got this covered as well. We have used an in-built function to help us with tempo calculation, i.e., ‘librosa.beat.beat\_track’.

**Mapping of Musical Notes:**

Once the frequency is achieved using the transforms, the next step of mapping needs to be done. Here, mapping refers to map the frequency of each corresponding note with the its corresponding string (Guitar strings, namely e,B,G,D,A,E which are distinguished by numbers, 1,2,3,4,5,6 respectively) and fret (starting 6 frets of the fretboard of a guitar).

This requires basic knowledge about the instrument since there are concepts like octaves which complicate the task. The mapped strings and frets are then to be used by the Arduino to convey signals to the hardware to play the instrument accordingly.

**Arduino:**

We aim to include 6 frets for all the 6 strings and this calls for the use of 6 servo motors for traversing between strings, 6 solenoids for pressing the respective string and then 6 servo motors for plucking the strings. This requires an Arduino code to operate these for different songs which refer to different tempos. For all the components to work with each other we have used PySerial to inter-connect these.

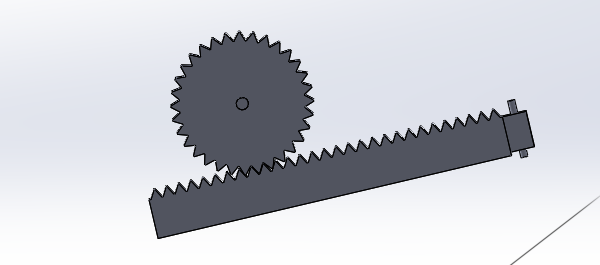
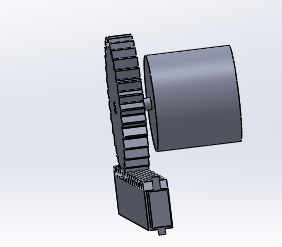
**Mechanical Model:**

In this we initially target 6 frets and all the 6 strings. Our initial aim was to look for the string pressing mechanism for the frets and to find optimum solution of motors for the translatory motion so that a smaller number of motors can be used. Our next aim was also to figure out the plugging mechanism of the strings.Possible methods were to scotch yoke mechanism and rack and pinion but scotch yoke is difficult to construct as well as control.

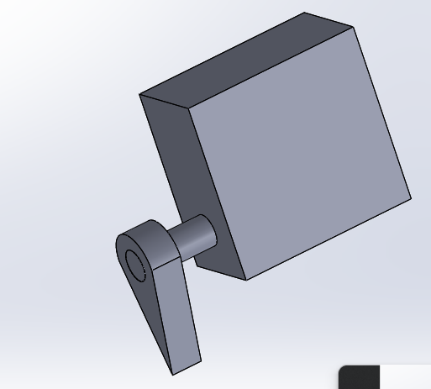
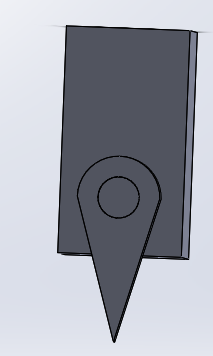
For pressing the strings, we have used SOLENOID mechanism and for translatory motion we have used rack and pinion mechanism. For plugging we have used 6 servo motors for 6 strings.

We have used this mechanism as this is fast, optimum, and simple mechanism. With this mechanism we can easily control our whole mechanical model.

So, we have decided to move horizontally in a single fret with solenoid attached at one end of rack. Similarly repeating this mechanism for 6 different frets.

For plugging mechanism, we have used 6 servo motors mechanism.

Technical details of mechanism

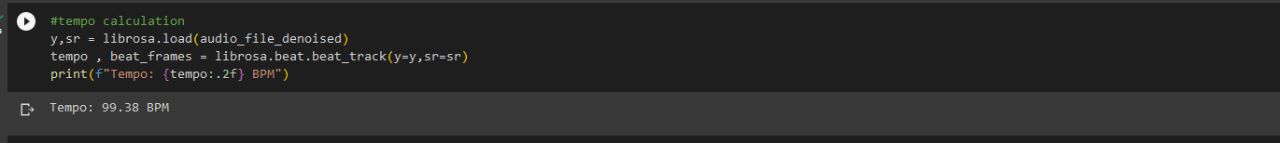
Length of rack = 150 mm

Diameter of pinion = 60 mm

Solenoid shaft length = 24 mm

Rpm of motor = 240 rpm

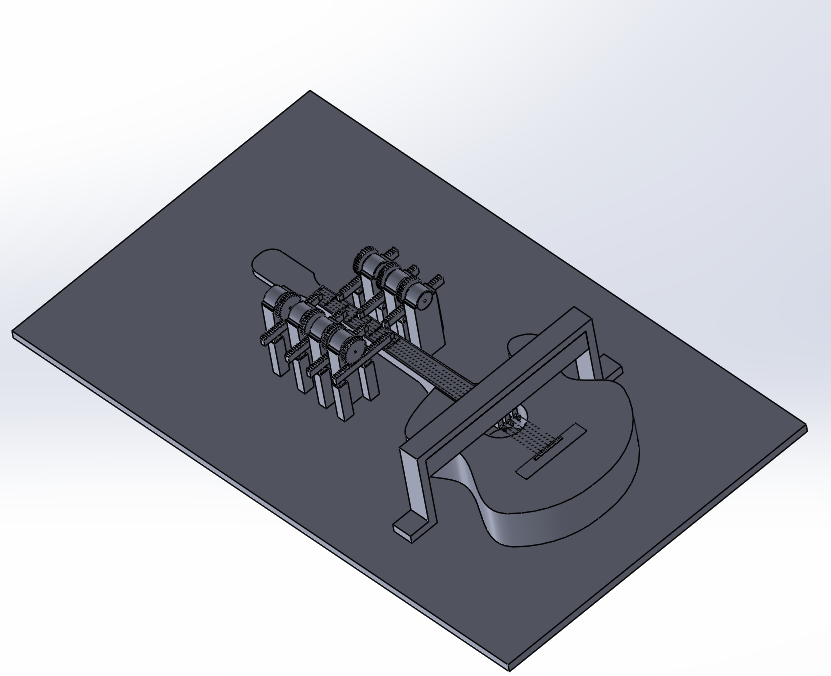
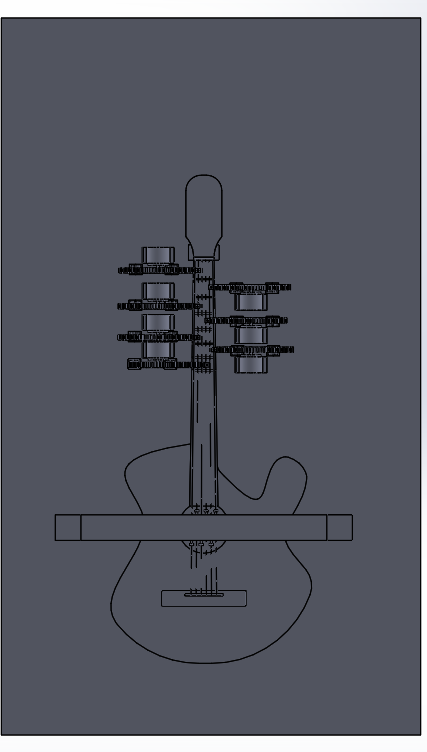
Time for solenoid to reach the desired location = 100 msec

Time for solenoid and servo motors application = t/2-100 msec (here t is the time available to play each note, it depends on the tempo of the song)

**Constructing cad model:**

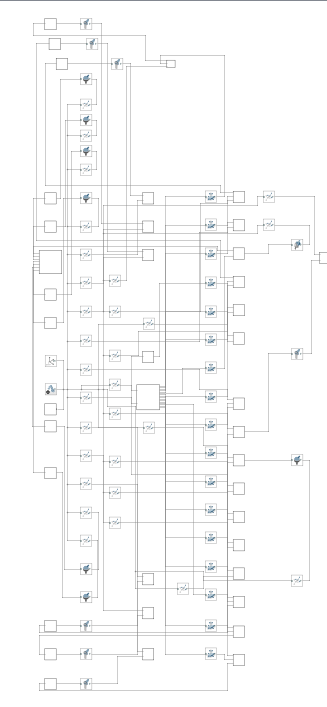
For this we have exactly replicated the cad of actual guitar with same dimension then we have designed rack and pinion mechanism with solenoid and then the plugging mechanism. And at last, we assemble all the parts.

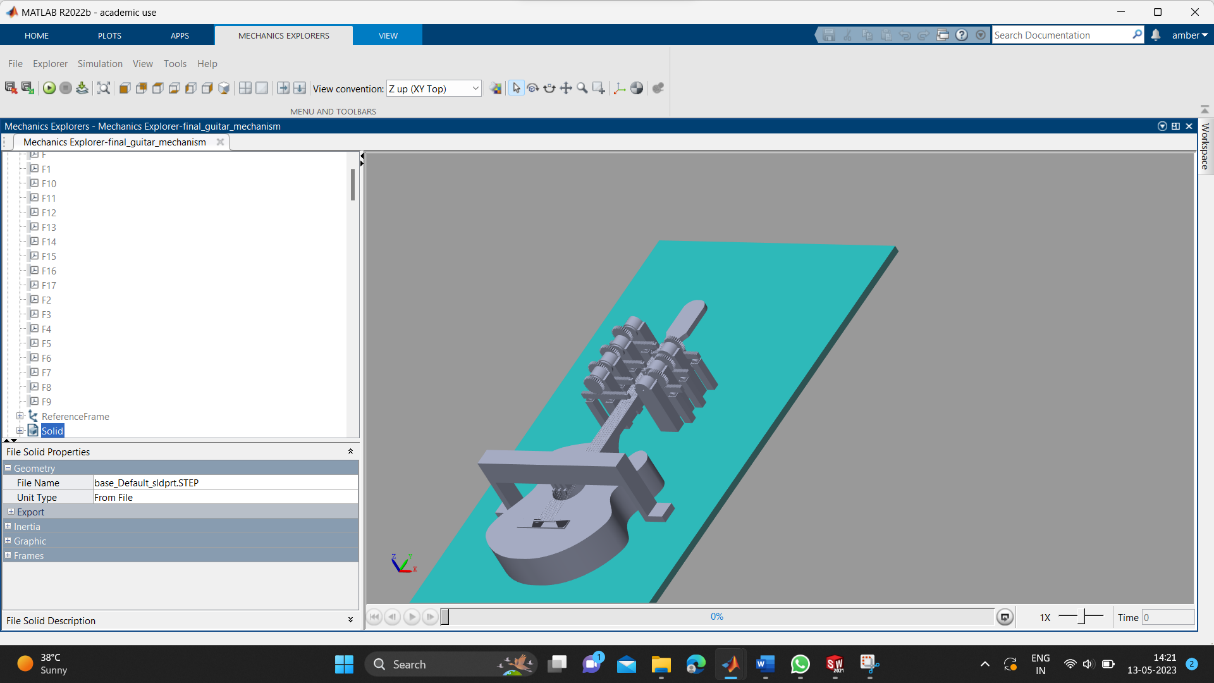
Final Output:



**MATLAB AND Simulink (Simscape multi body):**

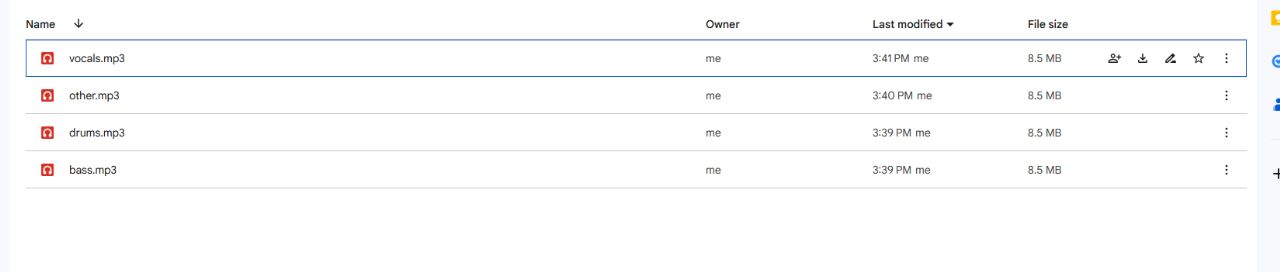
Then we have imported that cad model into the Simulink model with the help of simscape multibody link. The following model was created.



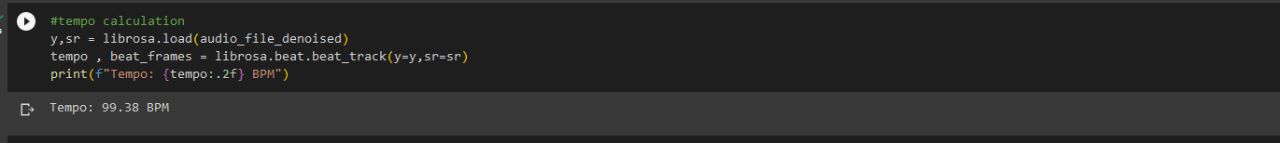


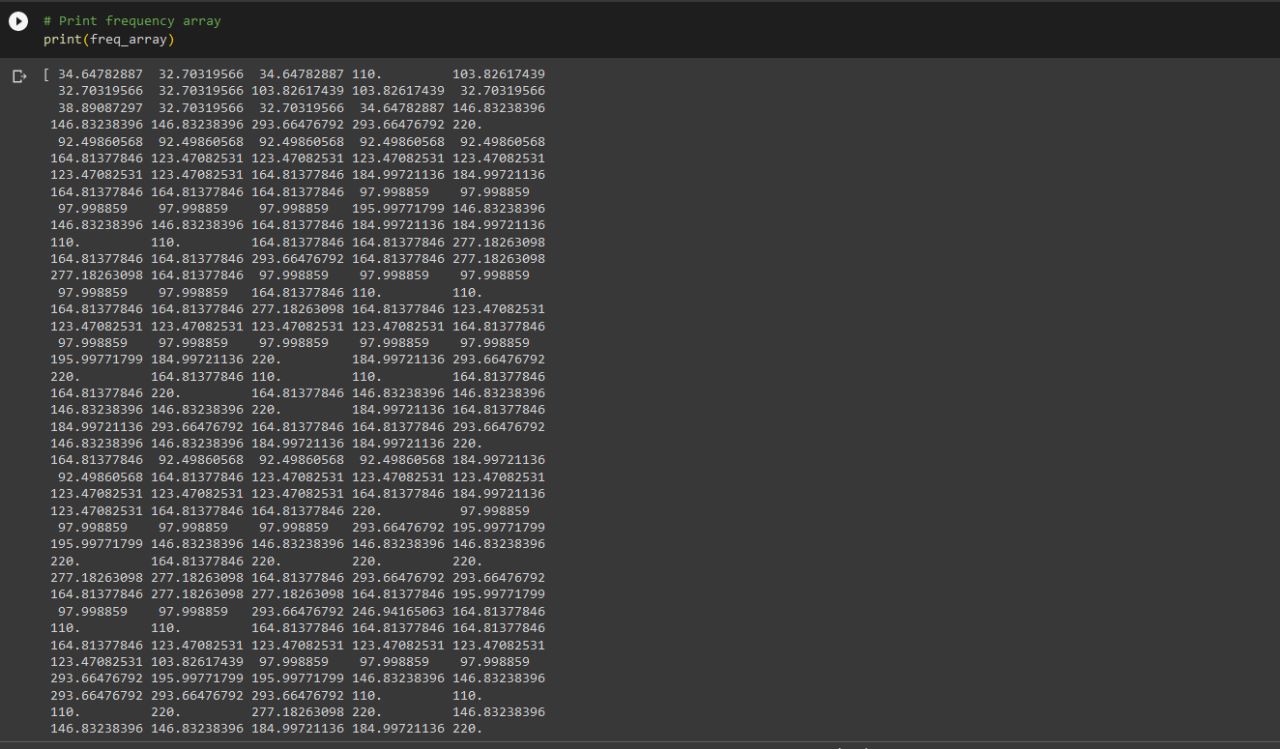
**Results:**

The source audio is processed using Demucs-



The tempo of the audio file using the beat-track function-

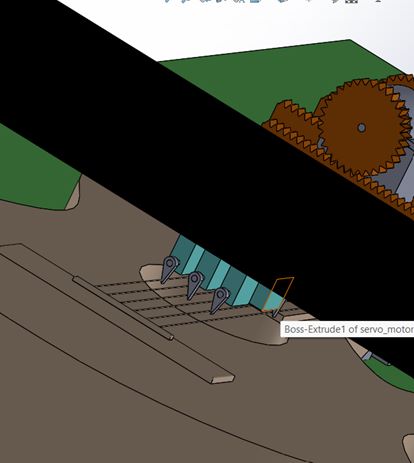
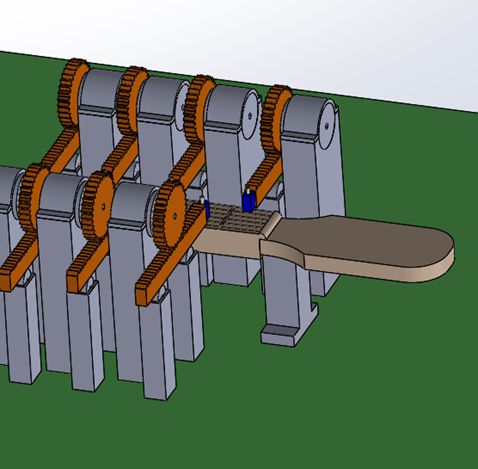


The frequency array is generated using ‘librosa’ inbuilt functions-

The results of CAD model-

A drawing of a guitar

Description automatically generated with low confidence

****

**References:**

<https://towardsdatascience.com/audio-to-guitar-tab-with-deep-learning-d76e12717f81>   
<https://project-archive.inf.ed.ac.uk/msc/20172175/msc_proj.pdf>   
<https://librosa.org/doc/main/generated/librosa.icqt.html>   
<https://escholarship.org/uc/item/6bw2065v>   
<https://musicinformationretrieval.com/>   
<https://github.com/facebookresearch/demucs>   
<https://github.com/timsainb/noisereduce>   
<https://github.com/GuitarsAI/BasicAutoTranscriptionRepo>