Camera Guitar Tuner

Project for COMP 4102

Instructor: Rosa Azami

Haojin Deng 100996326

Jiarui Li 100994952

Meitong Liu 101014282

Chengyang Liu 101011773

Table of Content

Abstract:

Guitar tuner is a device that guitar players alway to measure the frequencies that are produced by vibrating strings on guitars. Our project can implement this goal of estimating whether the notes played is standard by analyzing the waveform images of vibrating strings.

Introduction:

The major algorithms we use in this project are the Hough Transform and template matching. The Hough Transform Algorithm is important in our project because it can help define the positions of the static guitar's strings. The positions of the static strings of guitar can provide us the base to track the positions of vibrating strings. After defining the strings, we acquire a part of the image of one vibrating string based on the Hough lines we have and take it as a template image. We input the template image and the source image into the template matching algorithm to get the matching results and then use the results to compute the wavelength. This process is a challenge for us because it is hard to recognize the Hough lines correctly. Also, it is difficult to distinguish the difference among the six strings of guitar and filtering the different disturbance terms in the input video.

Background:

There are many guitar apps tuned by the tone of the guitar. There are three disadvantages of tuning by the tone(sound): not accurate at a noisy place, cannot automatically distinguish strings if two strings are in the same range, can only tune one string at a time. So we want to develop a guitar tuner with a camera. The camera can automatically distinguish which string you

are tuning and based on the wavelength. Guitar players can even test which string is off tune by strumming. This method would work perfectly outdoors.

We found this idea from “Final Project Ideas File” in culearn. After we looked at the rolling shutter effect of guitar strings, we think it would be interesting if we create a guitar tuner by analyzing the wavelength of the string. Rolling shutter effect happened based on very high shutter speed. In a phone camera, if the shutter speed could be 1/2000, we can see the guitar string jitter like a wave[1]. It is because when we take a shot at the string, the scanner speed of the camera is the same as the string jitter speed. Based on the same shutter speed and same tone on the same string, we assume the wavelengths are the same. Then we can use the wavelength of the string on each frame to determine the tone. We choose OpenCV 3.4 as the video processing library and numpy to process data and matrices.

Approach:

In order to find the correct hough lines in the input image, we set different thresholds in the opencv function cv2.HoughLines several times and finally get the experimental data threshold = 150. After applying the function cv2.HoughLines, we get the eligible 2D array (table 1) of (rho, theta), which can be used to draw Hough lines we find in the input image. However, after we draw the Hough lines on the image, we find that for each string of the guitar, many Hough lines overlap as Figure 1. Therefore, we need to remove the redundant lines to ensure that we only draw one Hough line for each string of the guitar. We set a threshold to filter the 2D array of (rho, theta) so that only one Hough line will be kept in a contained distance.

```
============= RESTART: D:\camera-guitar-tuner\periodDetection.py =
[[[152.          1.5882496]]

 [[ 60.          1.6057029]]

 [[293.          1.5882496]]

 [[ 55.          1.6057029]]

 [[155.          1.5882496]]

 [[290.          1.6057029]]

 [[151.          1.6057029]]

 [[107.          1.6057029]]

 [[287.          1.6057029]]

 [[200.          1.6057029]]

 [[104.          1.6057029]]

 [[148.          1.6057029]]

 [[196.          1.6057029]]

 [[240.          1.6057029]]

 [[247.          1.5882496]]
```

Table 1

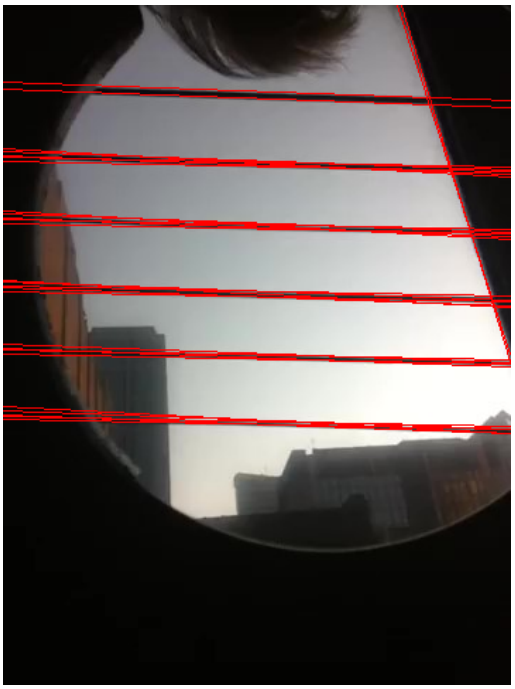

Figure 1

After removing the redundant (rho, theta) in the 2D array returned by cv2.HoughLines, we start to draw the Hough lines on the input image. In order to draw a line on an image, we need two points on the image. So, we set point1 (x1, y1) and point2 (x2, y2). According to the document of Hough Line Transform in OpenCV, a line can be represented as two form which are $y = mx + c$ in parametric form and $p = x\cos\theta + y\sin\theta$ where p (in our project is rho) is the perpendicular distance from the origin to the line (Hough line) and θ (in our project is theta) is the angle between this perpendicular line ι (from origin to the line) and horizontal axis. As we show in Figure 2, We can get a point0 (x0, y0) where the line ι is perpendicular to the Hough line and we can rewrite $p = x\cos\theta + y\sin\theta$ as $y = -\frac{\cos\theta}{\sin\theta}x + \frac{p}{\sin\theta}$. Since we have $y = mx + c$, we can get $m = -\frac{\cos\theta}{\sin\theta}$ and $c = \frac{p}{\sin\theta}$. Also, we know in Cartesian coordinate system, the slope of a line can be represented as $m = \frac{y1-y0}{x1-x0}$. We now have $\frac{y1-y0}{x1-x0} = -\frac{\cos\theta}{\sin\theta} = -\frac{1000\cos\theta}{1000\sin\theta}$. This result can be written as $y1-y0 = -1000\sin\theta$ and $x1-x0 = 1000\cos\theta$. Since we know the distance from the

origin to point0 is p, we can draw a perpendicular line from point0 to x-axis and a perpendicular line from point0 to y-axis. Then, we can get $x0 = p\cos\theta$ and $y0 = p\sin\theta$. Therefore, we have $y1 = y0 - 1000\sin\theta = p\sin\theta - 1000\sin\theta$ and $x1 = y0 + 1000\cos\theta = p\cos\theta + 1000\cos\theta$. In a similar way, we can get $y2 = y0 + 1000\sin\theta = p\sin\theta + 1000\sin\theta$ and $x2 = y0 - 1000\cos\theta = p\cos\theta - 1000\cos\theta$. Applying the results in the 2D array of (rho, theta) in a for loop with the above equations we get, we can draw all the hough lines correctly on the input image.

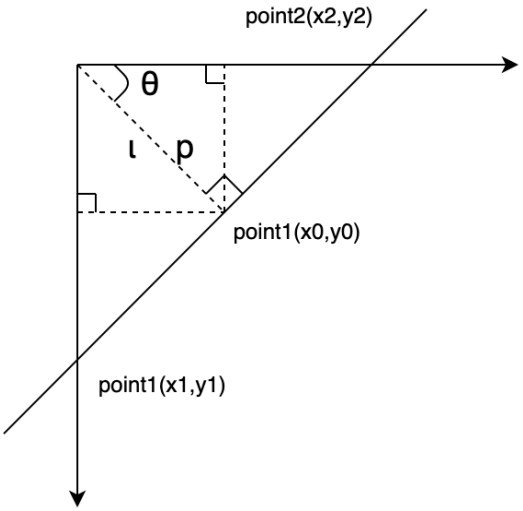

Figure 2

For the input frames, we use the mid points between the nearby Hough Lines to separate frames into 6 parts and each part contains one string. We want to separate the frame into 6 parts because the template may match to other strings and it would be confused for calculating the wavelength.

For template area, first, we calculate the midpoint of each Hough Line. Then we take a rectangle area around the midpoint of each Hough Line as our template. We intercept templates based on Hough Line because without applying force onto strings, part of the string must be inside of the template. We also added a threshold to prevent intercepting templates with no string or straight strings.

After acquiring the separate images of each vibrating string and the template images which are taken from these images, we apply the function matchTemplate() for each image and template image in a for loop. In the function, we firstly input the input image, the template image and the normalized cross correlation method cv2.TM_CCOEFF_NORMED into the opencv function cv2.matchTemplate. This function will return an grayscale image that each pixel in this image represents how much does the neighbourhood of the pixel match with the template image. Secondly, we start to extract the data that we need to draw the matching parts in our input image. Through several times testing, we set the threshold value to 0.9. Then, we collect all pixel coordinates which have values smaller than the threshold in an array and input this array in the function getResultAndWavelenth(). This function will firstly create a new array and secondly store the pixels that pass our filter. For example, we have two pixels with coordinates (x0, y0) and (x1, y1). If the value of x1- x0 is greater than 10, we will store this pixel into our new array. This step can help remove the repetitive matching area and finally give us all the top left pixels coordinates of the correct matching areas. Next, we use these pixel coordinates to calculate the distance between each two pixels (with the equation $distance = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$) and add up the distance. Finally we take the average distance as the wavelength of the vibrating string's wave. After that we return the array with filtered pixels and the average wavelength we calculate to the matchTemplate() function. This function will draw all the matching areas on the input image depending on the array with filtered pixels and output this image as well the wavelength.

Results:

We suppose to use the wavelengths of the strings to judge the musical nodes from the waveform images. The standard tuning defines the pitches as E, A, D, G, B, and E with open strings. Based on the formula $wavelength = speed / frequency$, each standard tuning has a fixed wavelength, which is used to identify whether the notes played is standard by comparing

the wavelength we get with the wavelength of standard tunings in this project. In this project, we used the Hough Transform and template matching method to find out the wavelength.

Firstly, we applied Hough Line Transform to find out the guitar strings and then delete redundant lines to maintain only one red line for each guitar string. The result of Hough Line Transform shows as Figure 3.

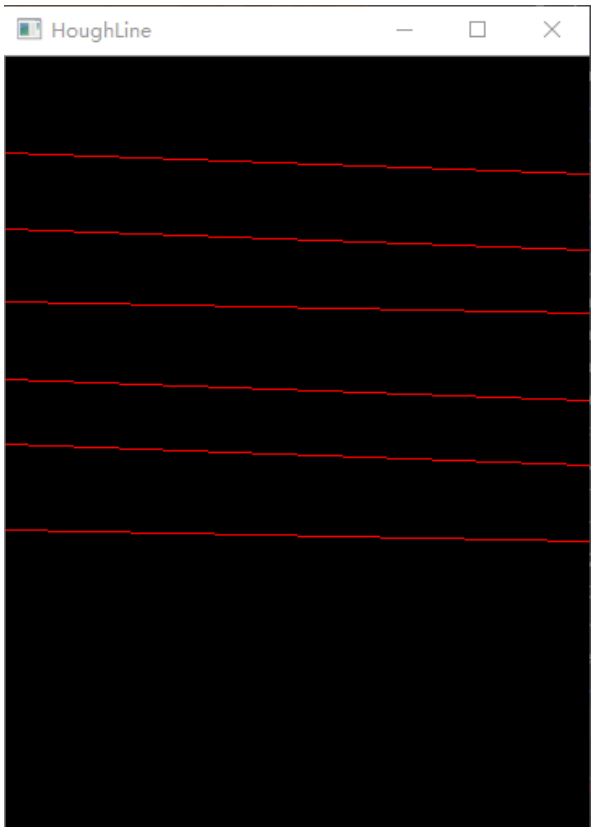

Figure 3

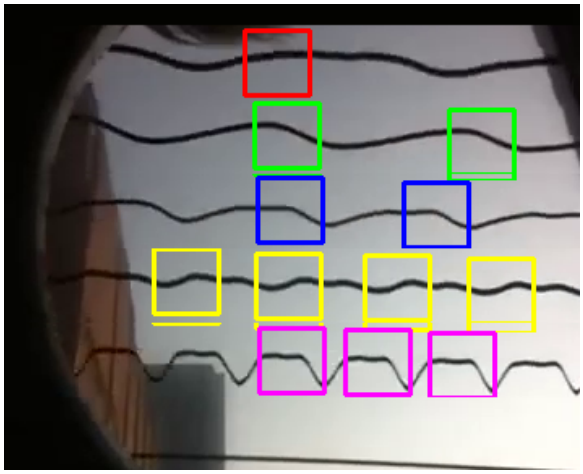

Figure 4

Then, we used the template matching method to find out the regions which are similar with the templates we chose for each string. Figure 4 shows an example of the template matching method. The wavelength we want is the distance between the templates, and then compare with the standard wavelength. But in view of the distance between camera and guitar strings might change in different videos, we also measured the distance between strings, since the distances between strings are the same in real guitar. The ratio of the wavelength to distance

of strings of each standard tuning is the measurement to judge whether the note is standard or not. Figure 5 shows the radios and the results in the frame displayed in Figure 4. The comparison between standard and current nodes and the result shows as Table 2.

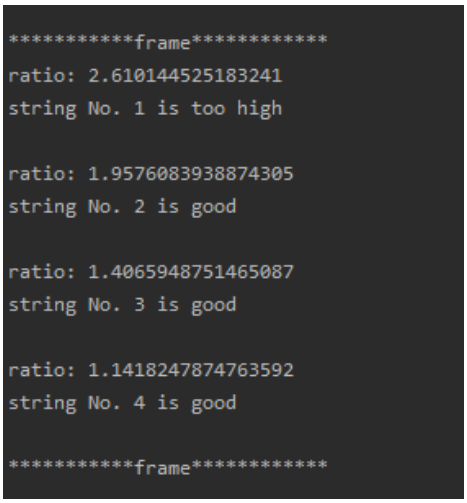

Figure 5

Standard tuning	Ratio for standard tuning	Ratio for current node	Result
E	2.775	nil	nil
A	2.195	2.61	too high
D	1.861	1.958	good
G	1.333	1.407	good
B	1.142	1.142	good
E	0.934	nil	nil

Table 2

List of Work:

Equal work was performed by both project members.

Future Work:

After we used the template method to match the same curves in guitar string, I found a new method to find wavelength. This new method is more accurate and contains more details but we don't know how to handle these data. As we tested, we found all strings can be detected as "edges". It means, I can calculate the direction perpendicular to the edge in every edge pixel by sobel x and sobel y based on following formula:

estimate the orientation of the edge normal

$$e_o(i, j) = \arctan \frac{J_y}{J_x}$$

Then in a guitar string "edge", the same direction pixels should be in the same position in different waves. Then from the same direction pixels on string we can calculate the wavelength. Figure 6 is an example of edge pixels which almost have the same direction as the axis-x. However it is very hard to process the data. We don't have any idea to only recognize pixels on guitar strings or only recognize one pixel at the same position in every period of wavelength.

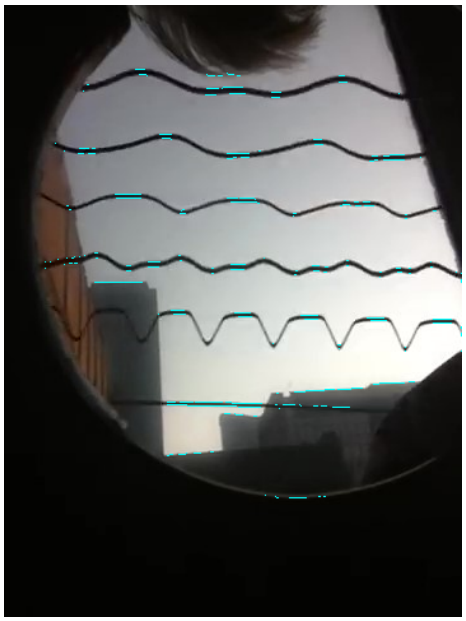

Figure 6

After we are able to make the guitar tuner by this method, we can even record all the notes played by the player on each string based on high accuracy. Then we can output the guitar tab for the song which the player is playing.

GitHub Page:

https://github.com/strollingorange/camera-guitar-tuner

References:

[1] *https://www.youtube.com/watch?v=TKF6nFzpHBU*

https://docs.opencv.org/master/d3/de6/tutorial_js_houghlines.html

https://docs.opencv.org/master/d4/dc6/tutorial_py_template_matching.html

https://www.idc.ac.il/en/schools/cs/research/Documents/thesis-shir-goldstein.pdf

https://isas.iar.kit.edu/pdf/SPIE01_BriechleHanebeck_CrossCorr.pdf