A Real-time Karaoke Scoring System Based on Pitch Detection

Minhao Zhang
University of Rochester
mzhang46@ur.rochester.edu

ABSTRACT

Currently there are many ways to give a karaoke user a score when he or she sings in a Karaoke system. Some of them are based on the loudness of the singers' voice, some based on tempo and some may have combinations of different method to give a precise evaluation of the singer's job compared to the original version. But among all the criteria, the pitch precision should be the most important factor that contributes to the user's score. Many application will also have graphs to show the user how well they do besides giving a final score only. Some are real time some are not. In our scoring system, the final score is based on the pitch precision only. The user will also have a real time feedback pitch difference in graph to show how much difference they have from the standard (original version) pitch while they sing. So when looking at the graph, the users can adjust their performance real time.

1. INTRODUCTION

This Karaoke scoring system can be taken down into 3 parts: the single pitch detection part, the real time sound feedback part and the graphs, plots and graphic user interface part. The pitch detection uses the YIN algorithm. The real time code just makes some short time frame. And then each frame is processed by YIN algorithm. The graphic user interface basically tried to make the system easy and intuitive for the users.

In later sections, this paper will focus on explaining each separate parts. For each one, the basic functionality and parameter will be showed and then followed by the testing procedure which tried to simulate the environment when all the parts are put together to work. Since this system is still a prototype currently, some drawbacks will also be discussed. In the end, it will also talk about the extensions, possible improvements and future work of this system.

2. THE PITCH DETECTION PART

2.1 Functionality

The pitch detection part uses the YIN algorithm. In other words, we detect the fundamental frequency of the user's voice and original song. Since most of the pitch depend on the fundamental frequency so this algorithm is good enough to do the job. In homework 2, this algorithm is successfully coded to work. But it's in the offline situation. The following part first discuss a little bit about the YIN algorithm and then some testing procedure and modification of the old code to make it work real time.

The YIN algorithm can be simplified into 6 steps.

1. The autocorrelation

   This step just breaks the whole signal into many windows and calculate the autocorrelation value of each window and plot the results of each window versus lagging samples the highest autocorrelation value corresponds to the position of the fundamental frequency. The equation used is (1).

   $$r' = \sum_{j=t+1}^{t+W-\tau} x_j x_{j+\tau} \qquad (1)$$

2. The difference function

   In this step we take the difference of the signal in each window. And plot the results. The following equation (2) is used.

   $$d_t(\tau) = r_t(0) + r_{t+\tau}(0) 2 r_t(\tau) \qquad (2)$$

3. Cumulative mean normalized difference function

   To make a better result in step2, we replace the difference function by the "cumulative mean normalized difference function".

   $$d'_t(\tau) = \begin{cases} 1, & if\ \tau = 0 \\ \frac{d_t(\tau)}{\frac{1}{\tau}\sum_{j=1}^{\tau} d_t(j)} & otherwise \end{cases} \qquad (3)$$

4. Absolute threshold

   The autocorrelation method is likewise to choose a high-order peak. So the threshold determines the power tolerated within a "periodic" signal.

5. Parabolic interpolation

   Based on the distribution of the results, we try to fit a parabola on it can take the local minimum as the final fundamental frequency.

6. Best local estimate

   Measure d'(T) that selects the best local estimate.

The six steps showed above are the basic for YIN. But in our system, we use only the first four steps. So we skip some explanations in step 5 and step 6. The original YI limits the error rate to 0.5%. But with the first four steps showed above, the error can be controlled to 0.78%. Table 1 shows the error rate of each step.

Version	Gross error (%)
Step 1	10.0
Step 2	1.95
Step 3	1.69
Step 4	0.78
Step 5	0.77
Step 6	0.50

Table 1. Table that shows the gross error rate for each step in YIN algorithm. Note only first 4 steps are used in this system.

As we know, one huge difference between a real time and an offline is the length of the signal in each processing iteration. There is only 1 frame in offline system but there are many more in real time system. So Table 2 shows the different working environment of the 2 different situation. And our job is to make sure the code works at the real time situation.

	Offline	Real-Time
Frame Number	1	Total/50ms
Frame Length	Whole song	50ms
Hop Size	10ms	10ms
Window Size	46.4ms	25ms
Depth	0.1	0.1
Sampling Rate	44100 Hz	44100 Hz

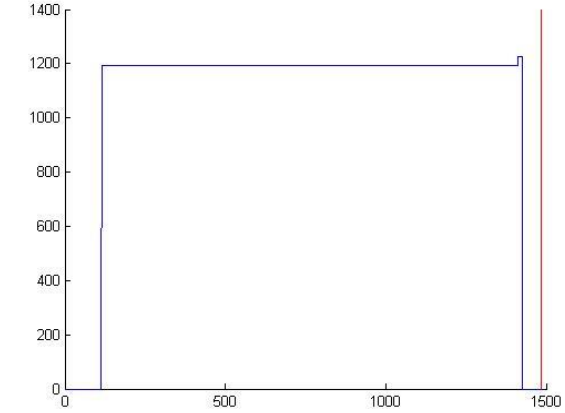
Table 2. Table that shows the environment parameter of different situation.

The final score is based on how close the user's pitch is to the original version's pitch. We only care about the relative difference of the 2 pitch contour so a ground truth answer is not necessary. Both version's pitch are detected by same YIN algorithm. The final score is in equation (4).

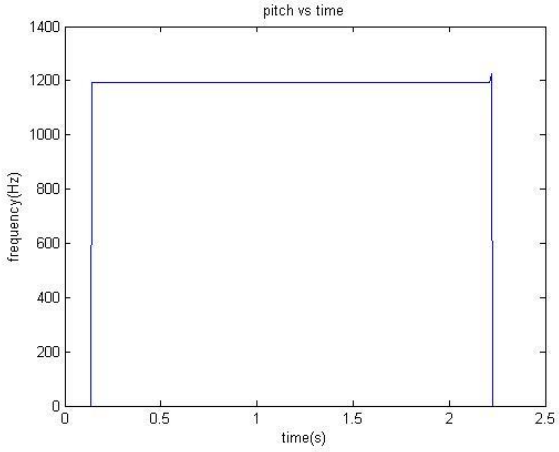
Score = 100- Sum of [P(orig) – P(user)]/total samples (4)

2.2 Testing Procedures

In order to have 50ms time frame, we change the time unit to signal samples unit. We use 20ms to test in this case. We first broke the whole song into connected frames with each one 50ms long. Then we run the YIN with the parameters in the third column in Table 2. Then we connect the pitch output in time order and compare it with the offline result. Figure 1 shows the single note testing result. We use the note: "BbClar_mf_C6B6_4.wav". As we can see from Figure 1, two pitch results are almost the same. So long as the offline algorithm gives us the correct results. The real-time code should be able to precisely detect the pitch that user sings. The x axis of real time is in samples. Also the length of samples are different is because the difference in hop size and window size. Same test procedure also applied to a human voice singing recording, which also shows a good result.



(a)



(b)

Figure 1. (a): The real time pitch detection result. The red vertical line at the end is the time cursor. The graph is originally an animation. This is the snapshot of the last time frame. (b): the offline pitch detection result

3. REAL-TIME IMPLEMENTATION

3.1 Functionality

The real time part's job is to let the user hear back their voice when they talk to the microphone. Meanwhile it should store the data of the user's singing signal, more specifically, the pitch data.

The time latency is 50 milliseconds without hearing back any artifacts. A 50ms is a good frame length, it will give only very short time delay when you hear your voice in the headphone. This part is hard to change since Matlab is a slow processing tool. If we code it in C++, the time latency

can be shortened. But it should not be too short, otherwise the YIN can barely work.

This part is coded using *audiovideo* class of Matlab 2012. But it cannot run beyond this Matlab version due to some modification and function removal in later versions.

3.2 Testing Procedures

This part does not have a systematic testing steps. We basically built the code, run it and listen to feedback from headphone. Note, we have to use a headphone to test it, since a speaker will get you into an infinite sound feedback which is pretty annoying. By listening to it we can roughly determine the time latency. And by change the parameter in the function, we can change how much latency we want.

Since we cannot compare exactly the data we received from microphone. So we just plot the data real time and roughly determine the correctness of the data stores. Figure 2 is one snap shot when a user is singing. And by observing the graph, the result makes sense to us. So this part should be able to work.

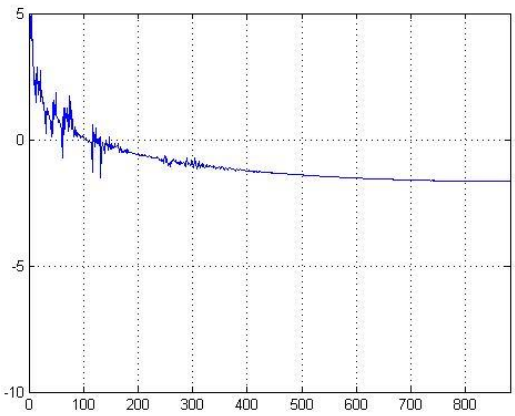


Figure 2. Vertical axis is in log scale, horizontal is in samples, right now it's a 20ms time frame. With Fs = 44100, it should have 882 number of points.

4. GRAPH, PLOT AND GUI

4.1 Functionality

The GUI lets user choose if they want the accompaniment. Then the user can choose when they are about to sing after they press the begin button. After the user began, the same graph with original pitch on it will have a red vertical line showing the time frame they are on, which across the pitch plot, so the user will know at which pitch they are on. Meanwhile somewhere on the red vertical line, the pitch that the user sing will be plot on the graph. Now by just following this time frame line, the user can compare at this single moment how close his or her pitch is to the original on. When they are done singing, they just need to hit the stop button. The track of how they sang will be left on the graph. There are also sliders to see the history of the pitch contour. They can zoom in and move around the final plot. The basic function to build the UI buttons are *uicontorl()*. And the some call back functions are related to the sliders and buttons to control the system. Some buttons do not have a call back functions like the next song button and previous song button. The reason they are there is to provide a complete out appearance of the system. And other people can add functions to them. Figure 3 shows what the GUI looks like at the very beginning.

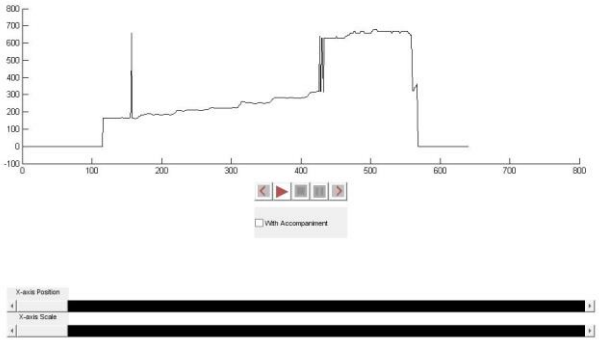


Figure 3. This is the very beginning when running the system. The black pitch contour is the original version's pitch. The button below the plot let user control the system. The 2 black bars are the sliders to zoom in/out and move around the plot when they user is done singing.

The main feature of the graph part is animation. Since the song's length may differ from each other and we sing it real time, there are 2 parts in the graphs need to be moving. First everything in the plot need to be moving like the pitch result, the time frame cursor. Second, the axes need to be moving. This is due to some long time song, if we make a really long time axis, the content in the graph will be hard to see, so it will not help the users to adjust their voice based on the graph.

In dynamic graphs implementation, one important design will be the efficiency. Plus we want it work in real-time, the efficiency is even more important. But creating part of the figure and axes are taking a lot of time. So in this implementation, instead of using *plot()* function in each drawing process, we use only one plot for each graph object. And we use the *set()* function to change the data of the drawing to make change with time. This way it saves a lot of time.

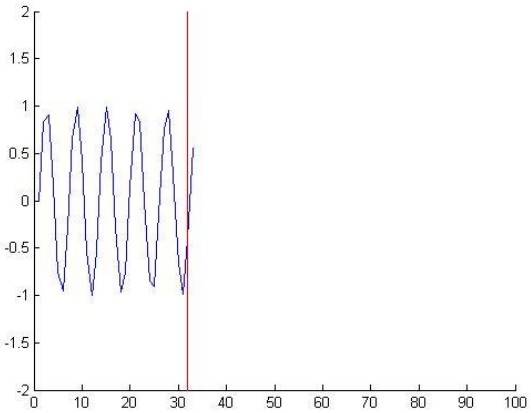
4.2 Testing Procedures

In order to test the dynamic graph, we made a sin function with the frequency to be the time value. We also plot the red vertical line together with it. So we know were exactly the time is. The animation can be described like an oscilloscope showing a sin function wave with a low frequency from an AC power source (like a function generator). Figure 3 shows some snap shots of this dynamic plot. The graphs are in time order. And the time cursor is moving

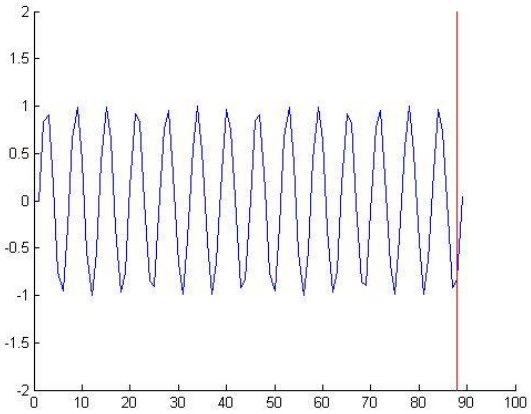
continuously. Note the time axis is in the unit of 0.1 second.

Besides, this part has successfully worked with the pitch detection part. Figure 2(a) is got from the dynamic graph. Figure 5 also shows a final version of the dynamic pitch plot with both original version and the user's singing version on it.

For GUI testing, we just build it and play around with it. We tried different combinations to press the buttons in order to find some bugs. And after fixing some bugs, the GUI basically works fine.



(a)



(b)

Figure 4. The dynamic graph plotting a sine function

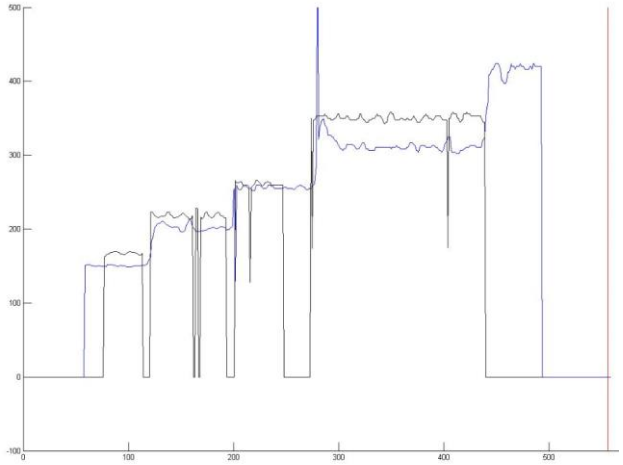


Figure 5. Black pitch contour is the original version's pitch. The Blue contour is the user's pitch. Time axis is the horizontal axis in samples. Vertical axis is in Hz, shows the frequency of the pitch.

5. FUTURE WORK

Since currently we just build the core part of this pitch scoring system. It is not very practical. For example it is very sensitive to noise. The user has to be really close to the microphone in order to send a strong signal. Or they have to have a good microphone not to picking the sound from the other directions.

So in future, more work can be done to make the pitch detection more robust.

The cover song played in the background is very well aligned with the time progress showed in the plot which can bring confuse to the users. It is cause by the time latency of the real time system.

A future work can be trying to reduce this synchronization.

Besides one can also try to code this system in other source code like C++, so it will have low latency and better graphic user interface.

6. REFERENCES

[1] A. Cheveigne: "YIN, a fundamental frequency estimator for speech and music," *2002 Acoustical Society of America*, pp. 1917–1930, 2001.