ECE420 Final Project Proposal

Let Him Sing

Ethan Zhou. Eric Tang
{yz69, leweit2} @illinois.edu

I. Introduction

The goal of this project is to build an Android app to perform auto-tune function for user inputs to target music. The music can be recorded in real time or pre-processed. This app is designed to perform voice/background separation for the music, spectro-analysis for the music, and temporal/pitch correction for the user input. The algorithms used will be implemented from the ground up.

II. Assigned Project Lab Review

Achievements and evaluation: In the Assigned Project Lab, we aimed to design and implement the algorithm used for the core functions of the system in Python to test system design viability, this includes voice/background separation via REPET, voice component analysis via STFT/Autocorrelation, Pitch correction via TD-PSOLA, and Temporal correction via resampling. Our evaluation process involved testing different expected scenarios, ranging from the most extreme (AI-generated monotone with significant length/rhythm mismatch) to the most ideal (pure vocal version of the song). Since the goal of the project produce an output catering to human perception, we primarily try to hear and minimize noticeable artifacts within the output. Here are our findings:

1. Due to the sharp transition of target frequencies between the frames, and the TD-PSOLA implementation, some noise was introduced even for the ideal test cases.
2. Due to the differences in the sources of our test case, wav files converted from m4a files saw significantly worse results than wav files converted from mp3 or mp4
3. Due to the temporal correction being implemented directly by resampling (without interpolation), in the cases where the original audio length significantly differs from the target, significant noise (static) is introduced in the output.

4. Due to us not having a speech recognition algorithm in our system, temporal correction can only be applied over sentences (we detect pauses during the vocal components), this makes rhythm correction less accurate.

Potential improvements: After gathering and examining the results, we have identified potential improvements that could be implemented to the system to address these issues.

1. To address noise introduced during frame transitions, a low pass filter could be implemented and applied to the output, the cut-off frequency should be initially set to 500Hz and later be manually tuned.
2. Interpolation could be introduced to the system to combat heavy resampling caused by large temporal mismatch. The issue itself would also be less pronounced for human inputs.
3. A speech recognition library could be incorporated into the system if time permits for better rhythmic correction
4. The REPET algorithm could be applied to the synthesized vocal before output for denoising, we need to verify on hardware whether this approach meets timing constraints.

III. Project Structure & Implementation

Overview: The project consists of 2 inputs, A recorded target song and user singing. Both audio files will be resampled (if needed) to unify the sampling frequency to 44100. The target song will then be processed to separate the vocal component and music component. A spectro-analysis will be performed on the vocal component to obtain the frequency map for the song. The User singing will first be temporally corrected to match the length of the vocal component from the song, then will be pitch corrected to match the result of the spectro-analysis. A low-pass filter will then be applied to the synthesized voice track for denoising. The final output will be produced by overlaying the synthesized voice track with the music component. Below is an overview block diagram of the system.

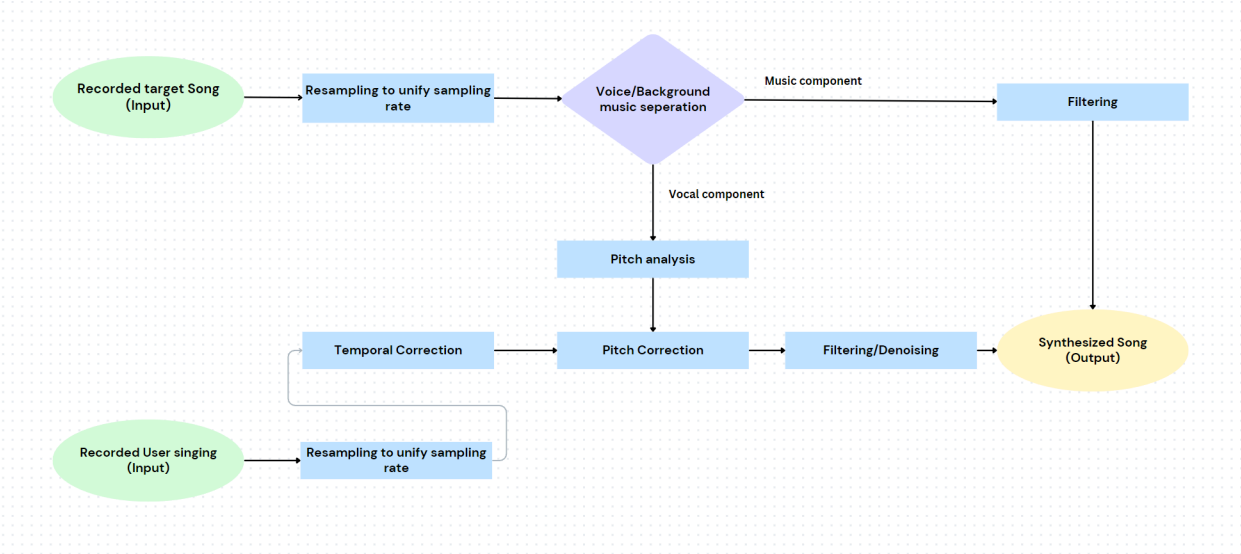

Figure 1. System Overview

Resampling to unify sampling rate: Since we are expecting audio data will be from a single primary source (the microphone of the Android device), we will simply apply resampling in the event that sampling rate differences arise.

Temporal Correction: We will use breaks in vocal tracks to identify and separate sentences, we will then apply resampling to adjust the length of each sentence from User singing to match the target song. Cubic interpolation will be used during upsampling to reduce artifacts and distortions.

A brief summary of Cubic interpolation:

The curvature of any curve is defined as $\kappa = \frac{y''}{(1+y'^2)^{3/2}}$,

Where y' and y'' are the first and second derivatives of y(x) with respect to x. To make the spline take a shape that minimizes the bending (smoothest curve), we will define both y' and y'' to be continuous everywhere. Each successive polynomial must have equal values, derivatives, and second derivatives when joining knots. As demonstrated by the equations below:

$$\begin{cases} q_i(x_i) = q_{i+1}(x_i) = y_i \\ q_i'(x_i) = q_{i+1}'(x_i) \\ q_i''(x_i) = q_{i+1}''(x_i) \end{cases} \qquad 1 \le i \le n-1.$$

$$q_1''(x_0) = q_n''(x_n) = 0.$$

Pitch analysis: We will divide each sentence of the target song into smaller frames around 40ms long (2048 samples long). We then perform autocorrelation to each frame to detect a fundamental frequency. We will assume the fundamental frequency is

unchanging for the duration of each frame. Since the maximum Beats per minute (BPM) for music is around 180, roughly 3Hz, our assumption for non-changing frequency wouldn't be very noticeable.

Pitch correction: We will divide each sentence of the user singing into smaller frames similar to the vocal component of the target song. We then perform autocorrelation to detect the fundamental frequency for each frame and regenerate each frame using TD-PSOLA. We will use a 3-frame buffer similar to in Lab5, for smooth edge transitions.

Filtering/Denoising: After synthesizing the new voice, we will need to perform filtering to reduce artifacts and statics. We will apply a low-pass FIR filter to the synthesized track to clean out high-frequency artifacts. Then we will apply REPET to remove low-frequency static. We will discuss REPET in more detail later.

Voice/Background separation: We utilize REPET to separate voice and background(music). The key idea is that music tends to have a repeating pattern, thus we can detect the repeating period and utilize it to create a soft mask. We implement this soft mask to get our music, or background noise. Then we just simply take the music part of the voice out of the original soundtrack. Which we can get a voice.
The key steps in pseudo-code are shown below:

l = length of b after discarding longest ¼ lag
b = beat spectrum of the original signal

Algorithm to calculate the repeating period p:
1. Initialize an array J with length l//3.
2. For each lag j from 1 to l//3 inclusive:
   a. Set delta1 to 2 if j is greater than or equal to 2, else set delta1 to j itself.
   b. Calculate delta2 as the floor of 3 times j divided by 4.
   c. Initialize I to 0.
   d. For each multiple of j, i, from j up to l in steps of j:
      i. Determine the index h1 where b has its maximum value in the range from (i - delta1) to (i + delta1), adjusting for boundary conditions.
      ii. Determine the index h2 where b has its maximum value in the range from (i - delta2) to (i + delta2), adjusting for boundary conditions.
      iii. Calculate the sum of b in the range from (i - delta2) to (i + delta2).
      iv. If h1 equals h2, increase I by the difference between b at h1 and the average value of b in the range from (i - delta2) to (i + delta2).
   e. Calculate the value of J for the lag j-1 as I divided by the floor of l divided by j.

Algorithm to calculate the soft Mask M:

r = number of repetitions when repeating period is p
N = FrameSize(we picked 1024)
n = N/2 + 1
m = number of segments of times

First Calculate repeating segment model S:
1. Initialize matrix S with the same shape as V, filled with zeros.
2. For each frequency i from 0 to n-1:
   a. For each lag l from 0 to p-1:
      i. Create a list 'values_at_l' containing elements from V at frequency i and lags 'l + k * p' for each k from 0 to r-2.
      ii. Set S[i, l] to the median of 'values_at_l'.

Second, we have to calculate the repeating spectrum model W:
3. Initialize matrix W with the same shape as V, filled with zeros.
4. For each frequency i from 0 to n-1:
   a. For each lag l from 0 to p-1:
      i. For each repetition k from 0 to r-1:
         - Calculate the index 'idx' as 'l + k * p'.
         - If 'idx' is less than the number of columns in V:
            * Set W[i, idx] to the minimum of S[i, l] and V[i, idx].

Finally, we get our desired output soft mask M:
5. Initialize matrix M with dimensions n by m, filled with zeros.
6. For each frequency i from 0 to n-2:
   a. For each lag j from 0 to m-2:
      i. If V[i, j] is not zero:
         - Set M[i, j] to the ratio of W[i, j] to V[i, j].
      ii. Else:
         - Set M[i, j] to 0.

IV. Project Milestones

Milestone 1: Since we have most of our system implemented in Python, we will aim to implement the improvements mentioned in section II, and we will try to implement other runtime optimization, including switching the frequency detection method to STFT, and optimizing the REPET from O(N^3) to O(N^2), This will be in python. We will also aim

to implement the “skeleton” of our Android app. With UI implemented, and audio-data passing straight through without processing.

Milestone 2: In the second milestone submission we will aim for an Android app with reduced functionality. With the REPET and pitch synthesis implemented, however, without any denoising, interpolation, and other performance-enhancing features. We will try to have some of the performance-enhancing features implemented if time permits.

Final Deliverable: The final submission will be the finalized Android app, containing the full system described in Figure 1. With two input options (recorded music or pre-processed music), and three output options (isolated background music, isolated vocal, fully auto-tuned music).

V. Minimum Viable Product

- The minimum viable product will be the Android app capable of separating the vocals from the background from any real-time recorded music.
- As a baseline goal, we aim to achieve similar results as the code provided by the paper.
- It is verified in the assigned lab that similar results to the paper can be achieved.

VI. Expanded Goals: Two Main Objectives

- High priority: implement the system as described in milestone 2: with temporal/pitch correction implemented without attempts at reducing distortions/artifacts.
- Low priority: implement the system as described in Figure 1: alongside the corrections, implement the noise canceling part of the system.

VII. Testing And Validation

- To demonstrate that the app works, we simply record a song and examine the separated voice/background music. Then we will record our singing to demonstrate the auto-tune feature.

- We will be using the Android device’s integrated microphone for inputs.

- For the voice/background music separation, the metric will be how cleanly the voice/background is separated, the baseline will be the result from the paper.

- For the auto-tune feature, The metric will be the perceived effectiveness in pitch correction, and the amount of distortions/artifacts present.

VIII. Contribution

- Ethan Zhou: Development of the system architecture, development, and implementation of supporting functions.
- Eric Tang: Development and implementation of the REPET algorithm.