Auto-Tune

Collection Editors:
Navaneeth Ravindranath
Tanner Songkakul
Andrew Tam

Auto-Tune

Collection Editors:
Navaneeth Ravindranath
Tanner Songkakul
Andrew Tam

Authors:
Navaneeth Ravindranath
Blaine Rister
Tanner Songkakul
Andrew Tam

Online:
< http://cnx.org/content/col11474/1.1/ >

C O N N E X I O N S

Rice University, Houston, Texas

Collection structure revised: December 20, 2012
PDF generated: December 20, 2012
For copyright and attribution information for the modules contained in this collection, see p. 14.

Table of Contents

Chapter 1

Auto-Tune: Introduction[1]

Motivation

Music is governed by a chromatic scale of pitches, an exact ratio of harmonics which sound pleasing to the human ear when arranged in song. Unfortunately, most individuals, even those with pleasant singing voices, struggle to sing at exactly the correct frequency for a given note on the harmonic scale. Because the human ear can precisely detect small deviations in pitch, a singer even slightly distanced in pitch from a chromatic tone can result in music which is audibly displeasing to the listener.

To achieve correct pitches in their recordings, many contemporary popular artists such as T-Pain and Rebecca Black use automatic pitch correction software, or Auto-Tune, ensure that their vocal tracks are perfectly in tune. These programs shift the pitch of each individual note up or down in order to match a note on the chromatic scale, leading to an output which is more pleasing to the listener. The corrected notes retain the musical properties of the original by also shifting the higher harmonics of the note. However, many amateur musicians do not have access to this type of software, which can be expensive and difficult to use.

Solution

We have created a software solution which, given a vocal track which is loaded into MatLab, quickly and precisely detects and corrects a vocal track to notes on the chromatic scale. Our solution successfully pitch corrects a given vocal track by detecting the pitch of each note and shifting it to the nearest note on the chromatic scale. However, the phase of the original signal is not completely preserved, resulting in distortion in the output. Our solution is quick and easy to use, requiring no musical knowledge, and results in an in-tune, if somewhat distorted, signal.

[1] This content is available online at <http://cnx.org/content/m45355/1.1/>.

Chapter 2

Auto-Tune: Challenges[1]

2.1 Challenges

There were several challenges we had to consider when implementing our autotune function in matlab.

1. **Isolating a single note**

We need to divide the time-domain signal into windows small enough that they cannot contain more than one note. Once we have performed windowing in the time domain, we can take the FFT for each window to get the isolated spectrum of the note sung. We then compare it to the closest note on the chromatic scale to determine the amount of shift needed.

[1] This content is available online at <http://cnx.org/content/m45378/1.2/>.

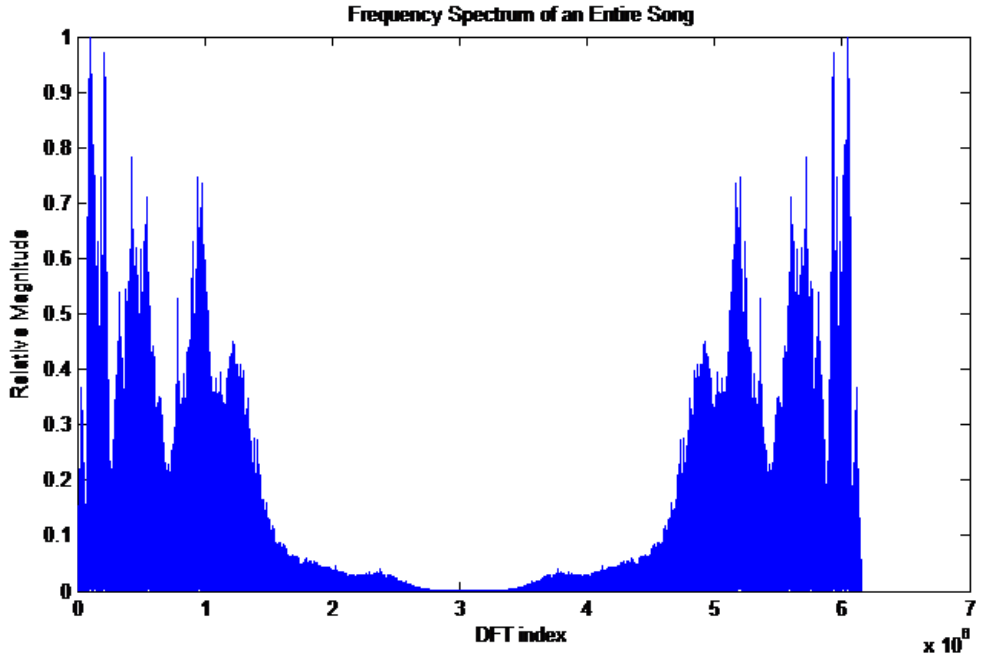

Figure 2.1: Spectral content of an entire song. Note the many peaks.

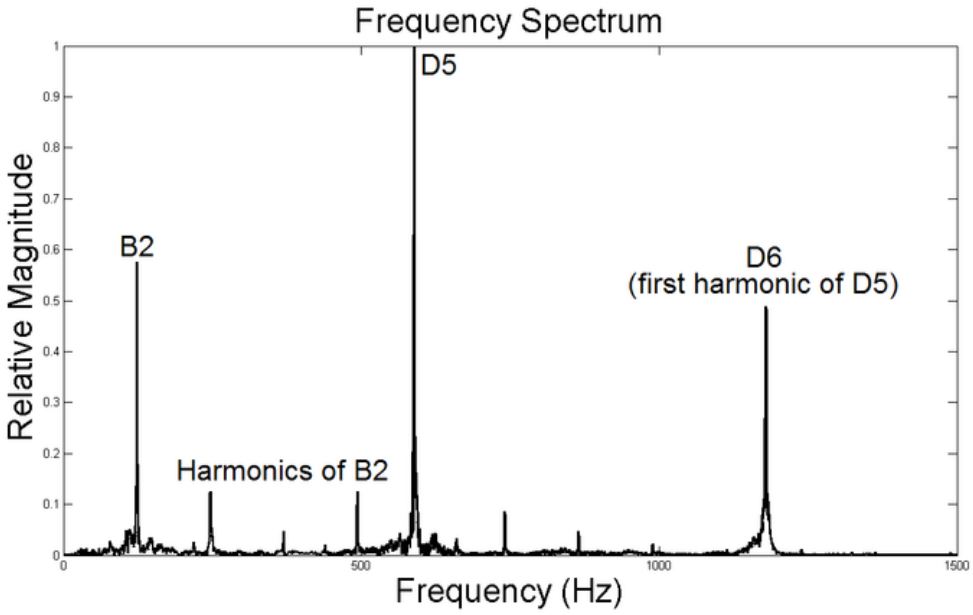

Figure 2.2: Spectral content of two notes and their harmonics. Note the distinct peaks. Figure reused from "Finding Piano Note Frequencies" connexions module by Scott Steger, http://cnx.org/content/m14191/latest/?collection=col10462/latest

2. **Transform Distortions**

Windowing the signal in the time domain requires the use of a filter. All filters create distortions in the frequency domain because of their non-ideal frequency response in the stopband.

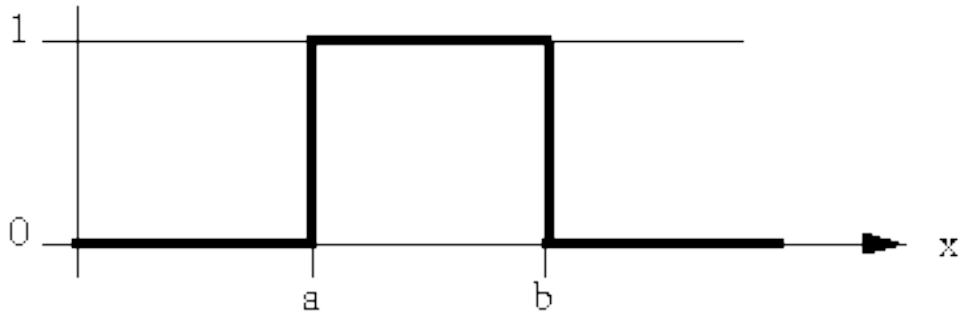

Figure 2.3: Ideal time-domain window http://accad.osu.edu/~smay/RManNotes/RegularPatterns/transitions.html [2]

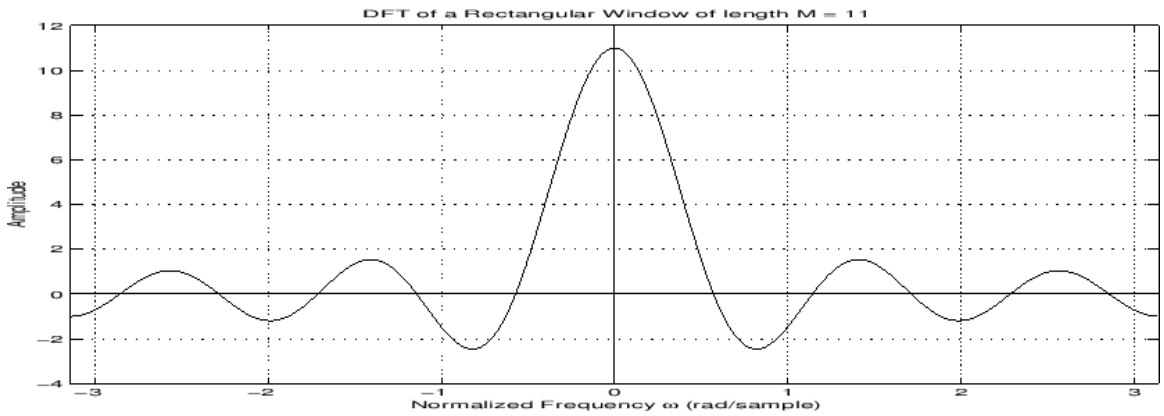

Figure 2.4: Frequency response of ideal time window. Note the nonzero amplitude outside the main lobe. https://ccrma.stanford.edu/~jos/sasp/Rectangular_Window.html[3]

In addition to these magnitude distortions, windows can have nonlinear phase. This changes the relative phase between the tones in each note. For any notes comprised of more than a single tone, this phase change will create audible distortion.

3. **Time Duration**

We change the pitch of our signals via the chipmunk effect; notes are shifted up or down by re-sampling, effectively playing the signal back at a lower or higher sampling rate. This method creates minimum phase distortion, but changes the time duration of the signal. In order to counteract this effect, we must stretch or compress each portion of the song to maintain its original time duration after re-sampling to change the pitch. We can then achieve our desired note shifting by playing the entire song back at its original sampling frequency.

[2] http://accad.osu.edu/~smay/RManNotes/RegularPatterns/transitions.html
[3] https://ccrma.stanford.edu/~jos/sasp/Rectangular_Window.html

Chapter 3

Auto-Tune: Implementation[1]

3.1 Overview of algorithm

When pitch shifting is mentioned, most people immediately associate it with frequency shifting. Frequency shifting can be easily achieved by simply modulating the input signal by a sinusoid; however, employing such a method creates a ring modulation effect, which is not the desired effect in this case. Thus, pitch shifting and frequency shifting are not the same thing. A true pitch shift can be realized by resampling the input signal. Unfortunately, this method changes the duration of the input signal, which is also not a desired effect. It turns out that a slight modification of the second method (resampling) can be used to accurately pitch shift a signal. In order to implement (crude) auto-tuning, we just need to break up the input signal into small windows and pitch shift each window by an appropriate amount. More sophisticated phase correction algorithms are required to remove the distortions that result.

Below is a schematic that summarizes our implementation.

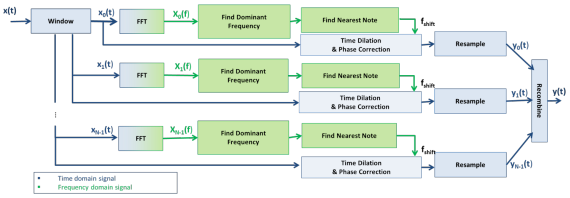

Figure 3.1

We will now outline our MATLAB implementation of auto-tuning. The algorithm can be broken down into three major steps:

3.2 1. Determining the shift ratio for a window

The input signal is first divided into windows of length 256, modulating by Hanning windows. To increase frequency resolution, the window is zero-padded so that its length is 512. The frequency spectrum of each window is then computed using a 512-point FFT. To find the dominant note in the window, the largest peak

[1] This content is available online at <http://cnx.org/content/m45377/1.2/>.

within a specified frequency range is selected. It does not matter whether we select the peak corresponding to the fundamental frequency or a harmonic since since both are expected to be out of tune by the same ratio. The frequency of the note is easily found from the index of the peak by a linear mapping: the first peak corresponds to a frequency of 0 Hz, and the last peak corresonds to the sampling frequency.

The next step is to find the frequency on the chromatic scale (440 Hz multiplied by integer powers of the twelvth root of 2) that the identified peak needs to be shifted to. To do this, we simply map the identified peak to the closest key on the piano and find the corresponding frequency of the note. The shift ratio is the frequency corresponding to the closest piano key divided by the dominant frequency in the frequency spectrum.

3.3 2. Pitch shifting a window

To pitch shift a window, we must first stretch/compress the window in time and then resample the window. In order to raise the pitch, we need to expand the window since we would like to resample at a higher frequency; similarly, lowering the pitch requires shrinking the window. For clarity, we will assume for the remainder of the section that we are interesting in raising the pitch for a given window. The steps involved in lowering the pitch are analagous.

In order to expand the window, we subdivide the window into smaller overlapping frames each of length 64, with 75% overlap, modulated by Hanning windows. Thus, each frame begins 16 samples after the previous frame begins. For a window of length 256, this will result in 13 frames. The 13 frames are then spaced out and added together so that the expanded window is larger than the original window by a factor of the shift ratio determined in the previous section.

We have now managed to stretch the window in time, but in doing so we have completely destroyed the linear phase of the window. Thus, the phase must be reconstructed. This is done by taking the FFT of each frame, adding the expected linear phase offset to the FFT coefficients in each frame by looking at the phase difference between the current frame and the previous frame, and finally taking an inverse-FFT to get the corrected frame in the time domain. We used an external package to handle these phase corrections.

To complete the pitch shift, we need to resample the window at a rate higher by a factor of the shift ratio. This is achieved by a simple linear interpolation. Note that the original length of the window is preserved since we have expanded the window and resampled the window using the same ratio.

3.4 3. Recombining the windows

Finally, the pitch shifted windows are combined together. Currently, there is no phase correction after recombination, and as a result, there is audible distortion in the output. Resolving the phase discrepancies for the entire signal is a rather challenging project since the phase is nonlinear. We encourage others to expand on and improve our implementation of this final stage of the algorithm by adding phase correction.

Chapter 4

Auto-tune: Experimental Results[1]

4.1 Experimental Results

Our implementation was able to correct the pitch of vocal samples, leaving the timbre of the voice mostly intact. Figs. 1 and 2 show spectrograms from a vocal sample of the song "Rolling in the Deep," originally recorded by Adele, as was sung by an anonymous woman and posted to YouTube. To listen to the results, click below. input [2] output [3]

External Image
Please see:
http://i.imgur.com/T1T6J.png

Figure 4.1: "Rolling in the Deep" input spectra.

External Image
Please see:
http://i.imgur.com/oZ79x.png

Figure 4.2: pitch-corrected spectra.

The figures show that pitch correction shifts the spectrum slightly, without significant distortion to its overall shape. It is desirable that the two figures are remarkably similar, as we wish to correct the pitch without significant distortion to the sound. Fig. 3 supports this claim by zooming in on the spectra of one window. We can see that a slight frequency shift has occurred, yet the shapes of the input and corrected spectra are identical. Also note that higher frequencies are shifted by a greater amount, which is necessary to account for the human brain's logarithmic perception of pitch. Recall that oversampling allows for pitch detection accurate to within 0.5Hz, with higher accuracies being possible through more severe oversampling and quadratic regression, at the cost of increased execution time.

[1] This content is available online at <http://cnx.org/content/m45379/1.2/>.
[2] http://www.mediafire.com/?rh8lkcpvcsh1cq9
[3] http://www.mediafire.com/?ucbo1ivb7xzbj8z

External Image
Please see:
http://i.imgur.com/oFCsj.jpg

Figure 4.3: Spectra of one window.

We admit one major drawback of our design, that the sound acquires a sort of robotic quality, partially due to the strict correction of tones to an unnatural level of accuracy. Many commercial systems take steps to correct the pitch only partially in some areas, such as in transitions between notes, to yield a more natural sound. This would be difficult, but not impossible to implement with the degree of automation that we desire. Still, best results require a human to work alongside the computer to tell where notes start and where they end, and how strictly to correct in different segments of the track, but this was not the goal of our user-friendly design.

Another serious issue causing the robotic sound is phase. While the phase vocoder attempts to preserve phase in the sub-windows during pitch shifting, we do not preserve phase from window to window, in which different shifts occur. Thus, there is a random incoherence every 256 samples that clearly affects the sound. While the spectral content of the signals has been shown to be accurate, the distorted phase remains a significant issue. Sophisticated algorithms have been developed to overcome this obstacle, but time constraints prevented us from implementing them across different windows, which undergo different shifts.

Finally, a less avoidable distortion comes from spectral leakage, due to windowing. Switching from rectangular to Hamming windows reduced this effect, but even with much more sophisticated designs, it is impossible to mitigate leakage completely. The corrected voice, however, remains quite recognizable, and we have achieved our primary goal of correcting pitch, with minimizing distortion only a secondary concern.

4.2 Conclusions and Future Work

We implemented auto-tune in MATLAB with a high degree of accuracy, requiring almost no input from the user. Oversampling yielded sufficient frequency resolution, while windowing resolved different notes in the time domain. Dominant frequencies were efficiently detected and matched to the nearest note on the chromatic scale, between which a shift ratio was determined. Using a phase vocoder and resampling, we achieved an accurate frequency shift in each window. In the future, much of the remaining distortion could be mitigated by utilizing phase-locking techniques between windows. Our open-source design provides an easy-to-use and accurate pitch correction.

Click here [4] to download an informative poster explaining our design.

4.3 Individual Contributions

The contributions of each group member are listed below: • Navaneeth Ravindranath: wrote 'find nearest note' function, drafted preliminary version of algorithm, wrote 'Implementation' Connexions module • Blaine Rister: help with algorithms and code, poster editing, plots, "Experimental Results" and "Installation Instructions" Connexions modules • Tanner Songkakul: prototyping helper functions, help with algorithms and code, poster creation, "Introduction" connexions module • Andrew Tam: prototyping helper functions, creating and managing poster graphics, "Challenges" Connexions module

[4] http://www.filedropper.com/poster_2

Chapter 5

Auto-Tune: Installation Instructions[1]

5.1 Installation Instructions

This[2] .zip archive contains the main function, autotune.m, as well as the helper routines pitchShift.m, fusionFrames.m, and createFrames.m. To install the program, first you must extract it from the archive. In Windows, this can be done by right-clicking on the .zip and selecting "extract here." Similar methods exist for other operating systems. Next, copy all of the files to your MATLAB directory, or any other directory in your MATLAB path. If you copy the files as part of a folder, then the folder must be included in your MATLAB path for MATLAB to find the autotune function.

The pitch shifting code is a slight modification of the "Guitar Pitch Shifter" from this site[3] . Note that while it is open-source, there may be legal issues with redistributing this code. Do so at your own risk. These issues can be avoided by substituting the pitch shifter for your own pitch shifting code, replacing the call to pitchShift() in autotune.m.

5.2 Using the Program

The autotune function takes in five arguments, X, pMin, pMax, w, and Fs.

X is the input signal, which is a matrix of amplitudes in either mono (one column) or stereo (two columns). This matrix can be extracted from a .wav file with the wavread command in MATLAB.

pMin is the minimum frequency in which we expect to find a peak, and pMax is the maximum, in Hz. We suggest passing in 60 for pMin and 1050 for pMax to cover the human vocal range, but this interval may need to be expanded for other instruments. In general, the narrowest possible interval covering all played notes is preferable, as this grants the greatest chance of correctly detecting the correct note.

w is the width of the window to correct as if it were an individual note. Note that the current version of the code rounds w to the next power of two, for simplicity. Theoretically, higher values of w would result in less distortion, but also poorer time resolution, where the boundaries between notes would be less precise. We found that 256 was a good value for this parameter.

Finally, Fs is the sampling frequency, in Hz, of the input signal X. This value can be obtained as the second return value of the wavread command, and is commonly either 22050 or 44100. Here is an example of the proper MATLAB commands to auto-tune a vocal sample:

```
    [X, Fs] = wavread('path to input file');  % Read input from .wav
out = autotune(X, 60, 1050, Fs);          % Process signal
wavwrite(out, Fs, 'path to output file'); % Write output to .wav
```

[1] This content is available online at <http://cnx.org/content/m45380/1.1/>.
[2] http://www.mediafire.com/?m627iuf74xxauij
[3] http://www.guitarpitchshifter.com/matlab.html

Alternatively, the output can be played from the MATLAB command line with the following command:

```
soundsc(out, Fs);
```

Note that the current version of the code compresses stereo input to mono output, and that execution time can be greatly reduced by processing only a portion of the input song. Please enjoy our code, and feel free to modify and improve upon it!

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

Attributions

Collection: *Auto-Tune*
Edited by: Navaneeth Ravindranath, Tanner Songkakul, Andrew Tam
URL: http://cnx.org/content/col11474/1.1/
License: http://creativecommons.org/licenses/by/3.0/

Module: "Introduction"
Used here as: "Auto-Tune: Introduction"
By: Tanner Songkakul
URL: http://cnx.org/content/m45355/1.1/
Page: 1
Copyright: Tanner Songkakul
License: http://creativecommons.org/licenses/by/3.0/

Module: "Auto-Tune: Challenges"
By: Andrew Tam
URL: http://cnx.org/content/m45378/1.2/
Pages: 3-6
Copyright: Andrew Tam
License: http://creativecommons.org/licenses/by/3.0/

Module: "Auto-Tune: Implementation"
By: Navaneeth Ravindranath
URL: http://cnx.org/content/m45377/1.2/
Pages: 7-8
Copyright: Navaneeth Ravindranath
License: http://creativecommons.org/licenses/by/3.0/

Module: "Auto-tune: Experimental Results"
By: Blaine Rister
URL: http://cnx.org/content/m45379/1.2/
Pages: 9-10
Copyright: Blaine Rister
License: http://creativecommons.org/licenses/by/3.0/

Module: "Auto-Tune: Installation Instructions"
By: Blaine Rister
URL: http://cnx.org/content/m45380/1.1/
Pages: 11-12
Copyright: Blaine Rister
License: http://creativecommons.org/licenses/by/3.0/

Auto-Tune
ELEC 301 Final Project

About Connexions
Since 1999, Connexions has been pioneering a global system where anyone can create course materials and make them fully accessible and easily reusable free of charge. We are a Web-based authoring, teaching and learning environment open to anyone interested in education, including students, teachers, professors and lifelong learners. We connect ideas and facilitate educational communities.

Connexions's modular, interactive courses are in use worldwide by universities, community colleges, K-12 schools, distance learners, and lifelong learners. Connexions materials are in many languages, including English, Spanish, Chinese, Japanese, Italian, Vietnamese, French, Portuguese, and Thai. Connexions is part of an exciting new information distribution system that allows for **Print on Demand Books**. Connexions has partnered with innovative on-demand publisher QOOP to accelerate the delivery of printed course materials and textbooks into classrooms worldwide at lower prices than traditional academic publishers.