1 概要

wav ファイルに保存された楽曲の再生とマイクの録音を同時に行いながら、楽曲のスペクトログラムと音声の音高 (基本周波数) および音量を同時に図示するカラオケシステムを実装した. 音量が一定以下の場合には基本周波数はプロットしないようにした.

python main.py [**filename**] [**vmin**] [**vmax**] [**thresold**] として実行すると, 楽曲のスペクトログラムを表示するウィンドウと, 音声の音高と音量を表示するウィンドウが現れ, 同時に音楽が再生される.

引数は以下の通り.

- filename: 楽曲が保存された wav ファイル
- vmin(Optional): スペクトログラムのカラーマップの最小値 (デフォルトは-10)
- vmax(Optional): スペクトログラムのカラーマップの最大値 (デフォルトは +10)
- thresold(Optional): 基本周波数をプロットする音量の閾値 (デフォルトは-20[dB])

再生する wav ファイルを data/God_knows.wav として、以下の環境で動作確認を行った.

- macOS Sierra version10.12
- python 3.6.7
- numpy 1.14.2
- matplotlib 3.0.1
- scipy 1.1.0
- sounddevice 0.3.12
- SoX

2 詳細

実装の上で最も注意したのが Player(楽曲の再生), Recorder(音声の録音), Plotter(グラフの描画) にあたる構成単位をどのように構成するかとそれらの間のデータのやり取りをどのようにするかであった. Player と Recorder は並列実行する必要があり, これは Thread ではなく multiprocess を利用することでそれぞれの処理の高速化を図った.

Plotter(matplotlib) はある 1 プロセスでの実行, InputStream はメインプロセスでの処理が制約となった. また, Player のデータも Recorder 同様 Stream から取り出したデータについて逐次処理することも考えたが, 複数の Stream に対して処理を行うと macOS のエラーが出てきて解決できなくなってしまったのでこの方針は却下した. 代わりにあらかじめ再生する wav ファイルの波形を全て取得して Spectrum を計算し, そこから時間経過に応じて最新の Spectrum を取得していくという方針に切り替えた.

最終的には図 1 のように構成した.

2.1 main.py

main.py では主にマイクからの音声の録音, グラフの描画および player.py の player_main 関数をマルチプロセスで呼び出すことを行なっている.

マイクから録音された音声は InputStream の callback 関数に渡され, 最新の波形が player_queue に入れられる. wav ファイルの再生では, 別プロセス (player_main) において最新の Spectrum が recorder_queue に入れられ, それをメインプロセスで受け取ってグラフの描画に用いている.

update_plot 関数, update_player_plot 関数はそれぞれ Recorder(マイクからの録音) の Frequency(基本周波数) と Volume(音量), Player(wav 楽曲の再生) の Spectrogram のプロットを行う. Frequency, Volume は RECORDER_N_FRAMES(=20) 個の要素の ax.plot (plotdata, plotvol) で構成されていて, Spectrogram は (PLAYER_LENFREQ * RECORDER_N_FRAMES(=100)) の np.array (player_plot) を ax.imshow で表示することで構成している.

これらの関数は matplotlib.animation.FuncAnimation 関数によって一定時間ごとに呼び出されて, グラフの更新を行っている.

update_plot 関数では, まず最新の波形として recorder_queue の中身を全て取り出し, RECORDER_FRAME_SIZE 個の framedata を構成する.

この framedata に関して get_loudness と get_frequency を呼び, その結果を plotvol, plotdata に追加し, また最も古い結果を削除してプロットを更新する.(RECORDER_N_FRAMES を保つ) ただし, ここで音量が thresold より小さいならプロットしないようにする.

update_player_plot 関数では, 最新の Spectrum を player_queue より取り出し, 結果を player_plot の最新として追加して, 最も古い結果を削除して画像を更新する.(plotvol, plotdata の処理と同様)

2.2 player.py

メインプロセスから player_main 関数が呼ばれる. ここでは Player インスタンスの初期化および Player.play 関数の実行を行い, 音楽の再生と Spectrogram の計算および最新の Spectrum を player_queue に入れる.

Player インスタンスの初期化時に calc_specgrams 関数で wav ファイル全てに対して一定時間ごとの specgram を計算して specgrams に格納する.

Player.play 関数ではまず playback 関数をマルチプロセスで呼び出す. playback 関数ではマルチプロセスで sox の play コマンドを実行している.(あらかじめ実行環境に SoX が導入されていることが必要である) playback が開始された時間が記録されていて, while ループで PLAYER_FRAME_SHIFT(=0.1) 時間経過するごとに specgrams のインデックスを 1 ずらして player_queue に入れることで現在再生している部分の specgram をメインプロセスに渡しているとしている.

2.3 helpers.py

get_frequency 関数ではフレームに分割された波形 wframe に対して, 自己相関を用いて基本周波数を推定する.

自己相関は $\tau = 0, 1, ...$ に対して

$$AC_r = \sum_t x_t x_{t+\tau}$$

と求められる.

はじめのピークは $\tau = 0$ にあり、2 番目のピークが基本周波数にあたる. 2 番目のピークを近似的に求める方法として、まず $\tau = i - 1$ と $\tau = i$ を比較して, $\tau = i$ の方が自己相関の値が大きくなったときにはじめのピークを抜けたと判断する. その後, 自己相関が最大となった τ に対応する周波数を基本周波数とした.

get_loudness 関数では wframe に対して, 以下の式で音量を計算する.

$$RMS = \sqrt{\frac{1}{N}\sum_t {x_t}^2}$$
$$Vol_{dB} = 20 \log 10 RMS$$

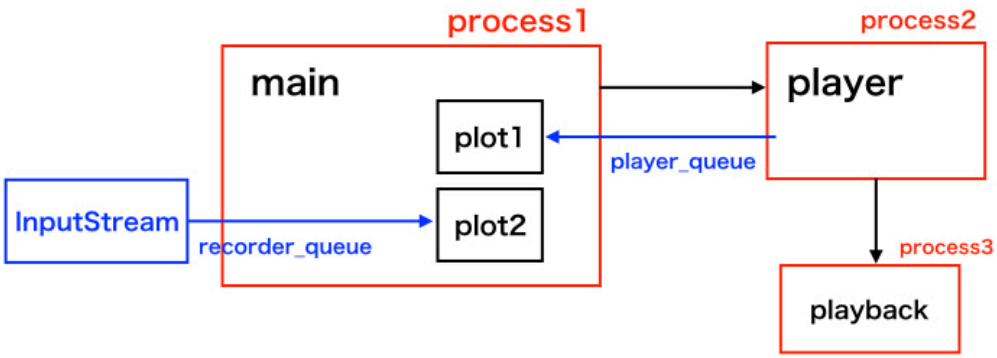
process1
process2
main
player
plot1
player_queue
InputStream
recorder_queue
plot2
process3
playback

図 1　プロセスの構成