Vol.2014-MUS-104 No.24
2014/8/27

LiVo: 母音列への歌詞アライメントによるリアルタイム歌唱合成演奏のための歌詞操作インタフェース

山本 和彦[1,a)]

概要：本研究では, リアルタイムに歌唱合成を用いた演奏表現を可能にするための, 片手での歌詞操作インタフェースを提案する. 本システムは, 一般的に楽曲の歌詞が演奏前にほぼ完成しているという前提のもと, ユーザによって連続的に入力された母音列から推測される最も尤もらしい位置へ歌詞をアライメントして出力する. 日本語では母音の数は 5 つ, 「ん」を含めても 6 つであり, それを入力するためのキーは片手の各指とほぼ一対一に対応させることができるため, ユーザはほとんど手の位置及び指とキーの関係を変える必要が無い. 最適な歌詞位置をシステム側で推定するため, 任意の歌詞位置への移動やユーザの入力ミスに対しても堅牢に動作する. また, 母音を入力するためのキーは鍵盤に割り当てられているため, 五線譜に記譜して練習することもできる. これにより, 実用的な速度で入力でき, かつリアルタイムでの歌唱合成演奏ならではの表現を可能にする自由度の高い歌詞操作を実現した.

キーワード：歌唱合成演奏，音声合成, 文字入力インタフェース，HMM，アライメント

LiVo: An Interface for Live Performance using Realtime Vocal Synthesis by Aligning Lyrics to Vowel Sequences

Abstract: In this study, I propose a novel interface for controlling the lyrics flow of a song by one-hand, that enables us to use singing synthesis at live performance. I assum the lyric of a song is almost completely made out before performance in common case. My system estimates the playing position in lyrics that fits the vowel sequences inputted by user. There are 5 vowels, or 6 vowels even including「ん」in Japanese, that matchs with the numbers of the fingers at one-hand. Then, user can fix the relationship between each finger of hand and each vowel input key while performance. Because my system estimates the optimum position of performed lyric, it automatically follows the intent of the user even if user jump the lyric position to perform or mistake. Additionally, the vowel input keys are mapped on common musical keyboard. Then, user can practice the skills for controlling lyrics by writhing it as normal musical score. As the result, a flexible lyric control interface that enables us to input the lyrics at practical speed, and play distinctive expression of singing synthesis as realtime performance are achieved.

Keywords: Live Performance, Singing Synthesis, Text Input Interface, HMM, Alignment

1. はじめに

VOCALOID[1] や UTAU[2] 等の歌唱合成ソフトの普及により, 歌唱合成をメインボーカルとして楽曲で利用することは一般的なものとなった. しかし, リアルタイムパフォーマンスで歌唱合成を使った表現は, 強い要望があるにも関わらず非常に限られた事例があるだけである. これは主に歌詞を実用的な速度で入力する方法が存在していなかったことに起因する. 本研究では, 歌詞と音高を同時にリアルタイムに入力することができるインタフェースを提案することで, この問題を解決することを目的とする.

リアルタイムに歌唱合成を使った演奏を行うために, 歌詞と音高を同時に入力するという試みは過去にも何度か試みられてきた. [4] では, 左手でキー入力をすることで文字, さらに同時に右手で鍵盤を使って音高を一音一音入力していくことでリアルタイムに歌唱合成演奏を行っている. し

[1] ヤマハ株式会社, 東京大学
Yamaha Corporation, The University of Tokyo
[a)] yamotulp@gmail.com

かし, 左手で子音と母音の組み合わせ, さらには濁音等の場合には 3 つ以上のキーの組み合わせを実用的な速度で入力するのは非常に困難であり, 簡単な童謡の演奏はまだしも実用的な演奏ができるとは言い難い. また, 複数キーの組み合わせによる文字の指定には, 少しでも各指の押下タイミングがずれると, 他の文字として誤認識されてしまうという根本的な問題もある.

フォルマント兄弟 [5] は歌唱合成演奏のための手法として過去最も成功した例である. 彼らは鍵盤やアコーディオンのボタンにおけるキーの組み合わせによって入力する文字を選択することで, 世に普及している楽譜として記譜できる形で高速な文字入力インタフェースを提案した. さらに, キーの組み合わせ方により, 微分音など歌唱としての歌詞以外のパラメータも同時に操作できるようにした点でも優れている. しかしながら, 例えば現在, ニコニコ動画 [3] に投稿されている歌唱合成を用いた楽曲の BPM のレンジはおよそ 80〜250 である. この速度で歌詞を入力することはフォルマント兄弟の手法でも不可能である. また, [4] でも問題となった複数キーの同時押しによる問題はフォルマント兄弟の手法にも存在する. さらに, この手法では, あらかじめ楽譜として記譜して練習することをせずに, ある楽曲の演奏を習得することは困難である. このため, 歌詞をリアルタイムに一音一音入力するにも関わらず, 即興で歌詞の構成や細部, または旋律を自由に変更するといったリアルタイム歌唱合成演奏ならではの表現が非常に難しいという問題もある.

そこで, 本研究では, ユーザの入力した母音列に対して事前情報として与えておいた歌詞をアライメントして出力することで, リアルタイムに日本語の歌唱合成を使った表現を可能にするための, 片手で操作できる実用的な歌詞操作インタフェースを提案する (図 1). 本稿では, 「文字の母音」とは例えば「か」なら「あ」, 「じゅ」なら「う」といったようにその文字から子音部分を取り除いたもののことを指す. 「ん」の母音は「ん」とする. また, 「母音列」とは, 例えば「かしょうごうせい」なら「あおうおうえい」のように歌詞の文字の母音が連続したものであると定義する. 日本語の母音は「あ」「い」「う」「え」「お」の 5 つであり, 「ん」を含めても 6 つである. 本研究では鍵盤にこれら 6 つの母音を割り当てて片手で入力し, またもう片方の手で鍵盤を使って音高を入力する. このとき, システムは, ユーザに連続的に入力された母音から, 最も尤もらしい歌詞の位置を推定し, その文字を発音する. 母音は高々 6 つしか無いため, ユーザは歌詞を入力するために, 片手の指と鍵盤の関係を, 「ん」を入力する場合以外は全く変える必要が無い. このため, 非常に高速に入力を行うことができる.

ここで, [4] やフォルマント兄弟の手法と異なる点として, 本研究では楽曲の歌詞は演奏前にほぼ完成しているものとする. 一般的に, 歌唱楽曲の歌詞は演奏以前にほとんど完成されている. アドリブで歌唱の旋律を自由自在に変化させることのできる歌手は数多くいる. しかし逆にアドリブで歌詞を最初から最後まで作りながら歌うことができる人間は非常に稀であると言ってよい. よって, 楽曲の歌詞は演奏以前にほぼ完成しているということを前提としても実用上問題は無い. さらに, フォルマント兄弟の手法も楽譜に記譜して練習することを前提としている以上, 歌詞は事前に完成しているとみなすこともできる.

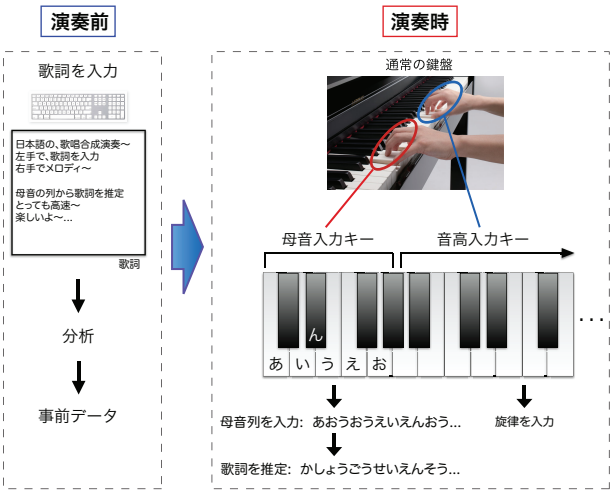

図 1 システム概要

本研究で提案するインタフェースは以下の条件を全て満たす.

(1) カラオケで一般的に歌われているような, どの日本語歌唱楽曲でもオリジナルのテンポで演奏できる.
(2) 旋律を演奏時に自由に変更することができる.
(3) 歌詞は演奏前にほぼ完成されてはいるが, 細部や構成は演奏時に変更することができる. 後戻りや歌詞位置のジャンプも可能であり, 多少の入力ミスも許容する.
(4) 歌詞も含めて五線譜に音符として記譜して練習することができる. 慣れれば記譜も必要としない.
(5) 習得の難易度は一般的な鍵盤楽器経験者にとってそこまで高度なものではない.

条件 (1) はシステムが真に実用的であるとするための最低条件であると考える. 条件 (2)(3) はリアルタイムの歌唱合成演奏ならではの表現をするために必要不可欠なものである. 条件 (4) はフォルマント兄弟の手法でも強調されているように, 多くの人が既に読み方を習得している楽譜というフレームワークをそのまま使って練習ができるということのメリットは非常に大きい. 条件 (5) に関しては, 本研究で提案するインタフェースの習得の難易度は J.S.Bach の 2 声対位法ピアノ楽曲と同程度かそれ以下であり, 決し

て高度なものではない.

2. 関連研究

2.1 歌詞入力手法

歌唱合成を使ってリアルタイム演奏するための既存の歌詞入力手法は 2 種類に大別される. 1 つは歌詞を予め全て打ち込んでおき, 鍵盤等の音高入力装置で発音する度に一文字ずつ単純に歌詞を再生していく手法 [6][7], 2 つ目は [4] やフォルマント兄弟 [5] のように同時に歌詞の文字 1 つと音高を入力しながら演奏するものである. 前者には歌詞を一切意識する必要がないので実用的な速度で演奏できるというメリットがある. しかし, 歌詞と対応する旋律が完全に固定されてしまうため, 旋律や歌詞を演奏時に全く変更することができないのに加え, 一度でもミスをしたら続行できないという問題がある. 一方後者は演奏表現の自由度にメリットがあるが, 前述したように操作が非常に困難で十分な速度で入力できないという問題がある. HANAUTAU[8] は後者に分類されるが, 音高を声で入力して両手を自由にし, 両手で QWERTY 配列キーボードを使って文字列をタイピングして入力することによって高速な入力を行っている. しかしタイピングがどんなに速いユーザでも一般的な楽曲を演奏するための十分な速度を実現することはできない. また, 発音タイミングの制御が非常に困難であるという問題もある. 一方, [9] ではフリック入力を併用している. フリック入力は文字入力の方法としては高速だが, それでも演奏として実用的な速度で入力するのは不可能である. また, ボタンに触れてからスライドさせるという 2 段階の操作が必要であり, リズミカルに入力する必要があるリアルタイムパフォーマンスには適さない.

2.2 文字入力手法

誤りを含む大まかな入力文字列からユーザの意図した文字列を推測して, その補助をしたり高速化したりする手法は, 現在の文字入力インタフェースの分野では一般的なアプローチである [10][11][12]. しかし, 従来の手法はどれも文字列を遅延評価, つまり, ある 1 つのまとまりをユーザが入力し終わってから遡って正しい文字列を推定するものであるため, リアルタイムで一文字ずつ発音しながら入力することが求められる歌唱合成演奏には利用できない.

2.3 楽曲アライメント手法

音楽 (音響信号, MIDI データ) に対して楽曲中の演奏位置を推定, 位置合わせ (アライメント) する研究は数多くある. 本研究のアプローチは, 何を何にアライメントするのかという違いがあるだけで, 本質的には楽曲アライメントの考え方に似ている. 楽曲アライメントの手法は, 音響信号間の各時刻の対における音響特徴量の距離を最小化する問題 [14][15] として解くものと, 確率的生成モデル [13] を

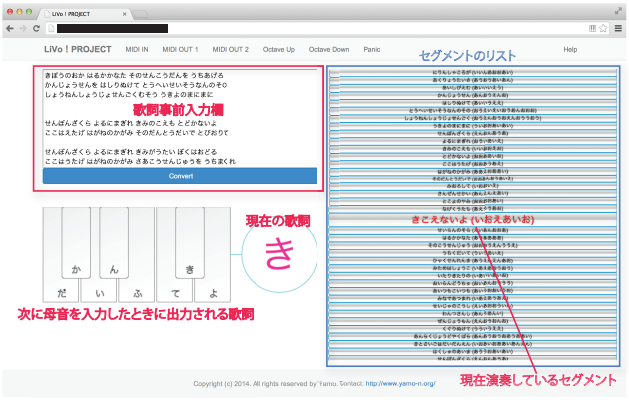

図 2 UI 画面

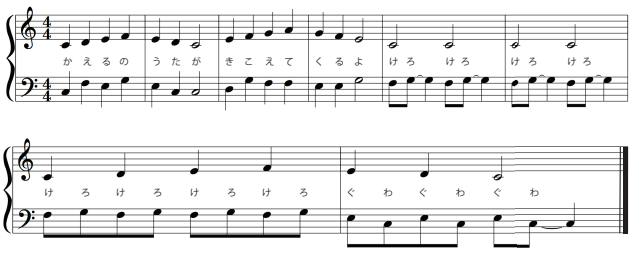

図 3 歌詞も含めた楽譜 (下段が歌詞)

利用するものの 2 通りがある. 前者は特徴量が適切に設定できた場合に性能が高いが, 演奏ミスや音色の違いなどの音響信号の不確定要素に弱いというデメリットがある. その一方で後者は, そうした不確定要素をシンプルに表現できる [16] というメリットがある. 本研究では歌詞の進行の仕方を確率的生成モデルで表現する.

3. 提案手法

図 1 が提案するシステムの概要, 図 2 がシステムの UI 画面である. 本システムでは, ユーザはまず演奏前に楽曲の歌詞を入力しておく. システムは入力された歌詞を分析して, 歌詞中のそれぞれの文字から他の文字への遷移のし易さ等のパラメータを決定する. 演奏時にはユーザは, 片手で鍵盤の一部分を割いて用意された母音入力キーを使って母音列を入力し, もう片方の手で残りの鍵盤を使って音高を入力する. システムは入力された母音列から予め入力されていた歌詞のどの位置をユーザが演奏しているのかを推定し, その位置の歌詞を出力する. ユーザが入力ミスをした場合や, 後戻り, 任意の歌詞位置への移動等を行った場合にも 0〜最悪 3 文字程度までの誤り文字を出力するだけで, その後は追従する.

音声合成音の発音は, 音高を入力する鍵盤が押されたときと, 鍵盤が押されたままユーザが新たに母音を入力したときの 2 通りのタイミングで開始される. 前者の場合は最後に入力した文字で, 後者の場合は新しく更新された文字で発音し直される.

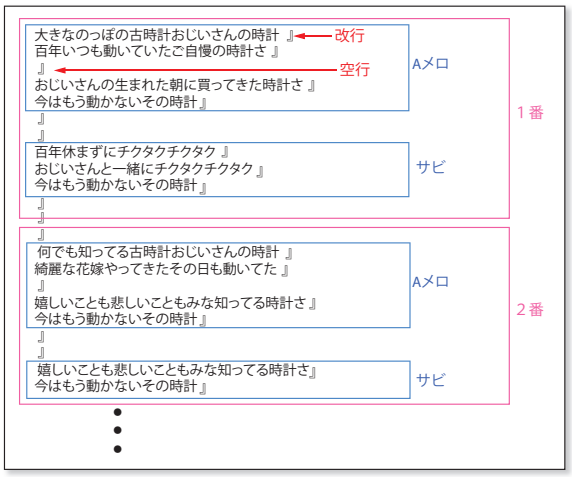

図 4 歌詞事前入力フォーマット
(作詞/作曲: 保富康午「大きな古時計」を例とした)

母音入力キーは音高を入力する鍵盤の一部を使って「あ」が C,「い」が D,「う」が E,「え」が F,「お」が G,「ん」が D#に割り当てられている. この割り当てる順番についてはどれが最適かという知見は本稿執筆時点では得られていない. 「ん」が D#であるのは手の中心位置に近い位置であるのが理由であり, 例えば A に配置するより格段に入力速度が上がることが分かっている. 鍵盤に母音が割り当てられているので五線譜に記譜して練習することができるというメリットもある. 例えば「かえるの合唱」(岡本敏明作詞) を歌詞付きで楽譜にすると図 3 のようになる. 歌詞の入力に 6 つの母音キーのみを使った単旋律しか要求しないため, フォルマント兄弟の手法での楽譜 [5] と比較しても非常に簡単な楽譜で表現できていることが分かる.

以下それぞれの内部処理について説明する.

3.1 事前歌詞入力

まず, ユーザはシステムに歌詞を入力する. 歌詞の入力形式は図 4 に示すように, サビや区切りのよいところで改行をすることが推奨される. 多く改行が入る (つまり空行の数) ほど大きな区切りであるとシステムは解釈する. ここで, (改行の数+1) を区切りレベルと定義する. また, 句読点は区切りレベル 1 として定義する.

システムは入力された文字列を最初に形態素に分割した後, 全てひらがなに変換し, 対応しない文字を除去する. 対応しない文字は句点, 句読点や伸ばし棒, 各記号や小さい「っ」である. なお, 小さい「っ」は音高を入力する鍵盤側でスタッカートをすることによって実現できる. アルファベットや数字は日本語読みに一文字ずつ変換されるので意図した通りに発音させるためにはユーザがカタカナやひらがなで入力しておく必要がある. さらにこの後, 一文字の形態素はその 1 つ前の形態素と結合させる. また「ゆらゆら」のように繰り返しのある形態素の場合は一度だけ (「ゆら」) に変換し, 繰り返し回数 (2) は記録しておく. この状態の各節をセグメントと呼ぶ. なお,「ん」から始まるセグメントは禁止している. 最終的に各セグメントは意味のある 2 文字以上高々 8 文字程度の文字列となる (図 5 最下段).

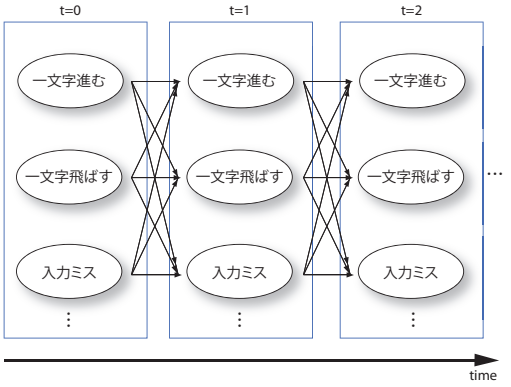

図 6 歌詞進行状態系列 (一部).

ここで, 一般的な歌唱楽曲において, 歌詞は大きな構造として A メロの 1 番, B メロの 1 番, サビの 1 番, A メロの 2 番…といった構造を持っている. さらに各構造は複数の文章から構成されており, その各文章も複数のセグメントの集合から構成されている. そしてセグメントはさらに複数の文字から構成されていると考えることができる. この階層的な構造は図 5 のようになる. システムはこの階層構造を, 区切りレベルから構築する. 具体的には, 大きな値の区切りレベルから順にセグメント列をクラスタリングして木構造を作成する. ここで, 各セグメントが先頭となる最も浅い階層の深さ (根は除く) のことをそのセグメントの階層レベル, と定義する. 図 5 の各セグメントの右上の値は階層レベルを表している. また, この木構造上で, ある異なる 2 つのセグメント間を移動するのに辿る枝の数を, セグメント間の距離として定義する.

3.2 HMM による歌詞文字列間移動のモデル化

演奏時のユーザによる歌詞位置の移動の仕方に対してはいくつかの仮定をすることができる. 例えば, 最も多く行われるのは当然一文字ごとに順番に進む, というものであろうし, 大きく位置を移動した直後にまた大きく位置を移動するということはあまり考えにくい.

そこで, 現在の文字から一文字進む, 一文字飛ばす, 入力ミスする... といった演奏時の歌詞の進行の仕方を, 状態数 S の Ergodic Hidden Malkov Model から生成された, 長さ T の状態系列 $Z(t = 0, 1...T)$ で表現する (図 6). これは演奏時のユーザの行動モデルと考えることができ, 最大 S 種類の進行方法を T 回選択して演奏を行うと仮定することに相当する. この HMM は初期状態 π, 状態遷移確率 τ に従って遷移し, 母音列を出力する.

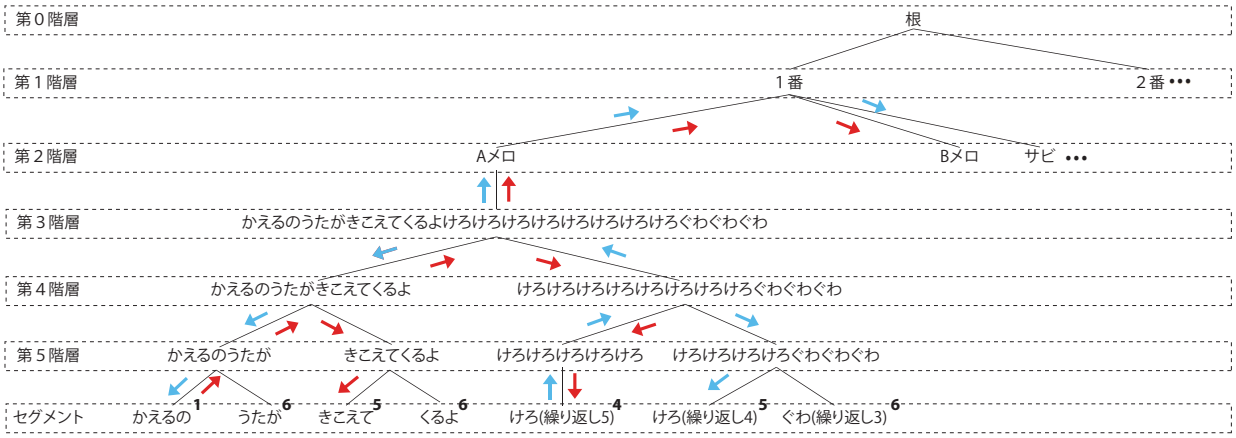

図 5 歌詞の階層構造

$$p(Z|,\pi,\tau) = \prod_{n=1}^{S} \pi \prod_{n=2,s'=1,S=1}^{N,S,S} \tau_s(s')^{Z_{s'}(n-1)Z_s(n)} \quad (1)$$

本稿では, 遷移確率 τ はルールベースで決定する. τ を何らかの事前データから学習, あるいは演奏時に強化学習させることは今後の課題である.

まず, 最も観測されるであろう文字列間移動の仕方は, 明らかに本来の歌詞で順番に並んだ文字への移動である. 旋律に対する文字数の調整や入力誤りによって一文字飛ばされる可能性もある. また, セグメントは意味を持った一区切りであるので, セグメント単位での繰り返しや演奏位置の移動が行われる可能性は高いとみなせる (セグメント途中から他のセグメントに移ると言語として意味を成さない). 歌詞独特の遷移方法としては, 一文字を伸ばしながら旋律のみを変化させる場合に, 現在発音している文字の母音成分のみを繰り返す可能性も高い. さらに, ユーザが演奏位置を大きく移動する場合, 階層レベルの観点からみて, 区切りの良いセグメントへ遷移する可能性が高いと考えられる (例えば, 1 番の A メロ→ B メロときて 1 番のサビを飛ばしていきなり 2 番のサビに遷移してそのまま短く終了するなど).

以上を踏まえて本稿では, 以下のいずれかの文字列進行に従ってユーザは歌詞中を移動するものと仮定する.

(1) 繰り返し有りセグメントの終端から先頭
(2) 次の文字に進む
(3) 歌詞中に他に重複した文字列がある場合, その最後の文字から直後の異なる文字
(4) 入力ミス
(5) 繰り返し無しセグメントの終端から先頭
(6) 現在の文字の母音のみへ
(7) 同じ位置にとどまる
(8) 同セグメント内で一文字飛ばした文字へ
(9) 歌詞中に他に重複した文字列がある場合, その最後一つ前の文字から直後の異なる文字へ
(10) 歌詞中に他に重複した文字列がある場合, その最後の文字から直後の異なる文字の次の文字へ
(11) 繰り返し有りセグメントの終端からセグメント 2 文字目
(12) 繰り返し有りセグメントの終端一つ前からセグメント先頭
(13) 歌詞中に他に重複した文字列がある場合, その最後 2 つ以上前の文字から直後の異なる文字へ
(14) 同じ親ノードを持つセグメント群それぞれの先頭へ
(15) 枝を辿って至ることのできる, 現在のセグメント階層レベルと同じか深い, 全ての階層のノード (階層 H_n) の下に連なるセグメント群のうち, 階層レベルが (H_n+1) のセグメントそれぞれの先頭へ
(16) 枝を辿って至ることのできる, 現在のセグメント階層レベルより浅い各階層のノードから伸びる枝ごとに探索し, 階層レベルが最小のセグメントそれぞれの先頭へ

リストの上方に羅列されているほど, 数多く観測されるものと仮定する. (4)〜(10) は同程度観測されるとする. (6) は次の文字の母音が現在の文字の母音と異なる場合のみ発生するとする. (3), (9), (10) は例えば, サビ 1 番とサビ 2 番の前半が全く同じで後半が異なる場合, サビ 1 番前半からサビ 2 番後半へ飛ぶなどの進行を表している. また, (14)〜(1 6) の場合は複数あるが, 距離が近い移動ほど多く観測されるものとする. (15) の一例を図 5 の中の赤い矢印に, (16) の一例を青い矢印で示している. HMM の状態遷移を考えると, リストの下の状態ほど自己ループの確率は低く, また上の状態ほど下の状態から遷移する確率は高いものとする.

以上の仮定のもと, τ を決定する. 方法は一通りではないが, 例えば, (i) から (j) の状態に遷移する確率を τ_{ij} とすると, $\sum_j \tau_{ij} = 1$, $\tau_{ii} \leq \tau_{jj}$ $(i < j)$, $\tau_{ij} \leq \tau_{ik}$ $(j < k)$, となる

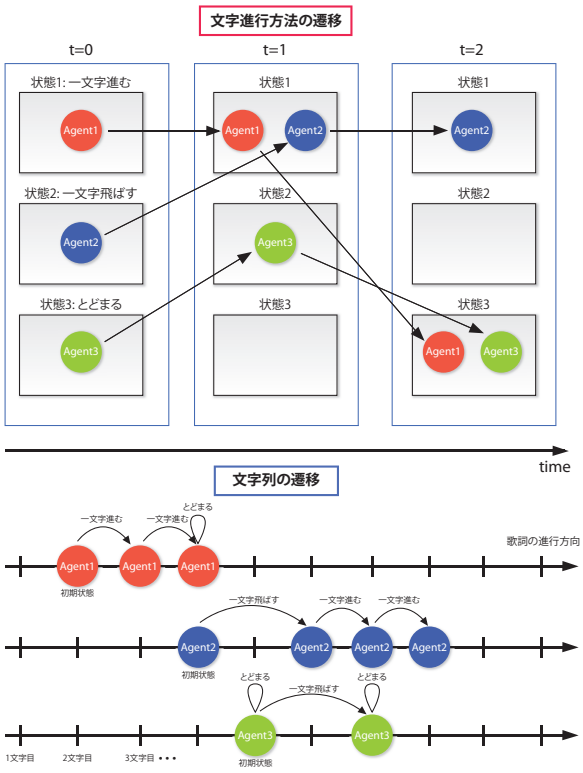

図 7 マルチエージェントによる探索

ように適当な分布から割り当てる.

3.3 マルチエージェント探索による母音列からの歌詞位置推定

演奏時, システムはユーザが入力した母音列から歌詞中のどこを演奏しているかを推測し, 最も尤もらしい歌詞をユーザが母音を入力する度に出力する. 本研究ではこれを, ユーザの入力した母音列を出力できる, §3.2 の HMM の Viterbi 経路を求めることで行う. ここで, §3.2 の HMM 中の各状態は, 歌詞中の位置というさらなる潜在変数を持つことに注意する必要がある. よって単純に Viterbi アルゴリズムで計算することはできない.

この問題は, マルチエージェント探索 [17] を利用すると効率良く解くことができる. マルチエージェント探索では, HMM 中の異なる状態を遷移し, それに従って歌詞中の異なる位置を走査する複数のエージェントを動的に生成する (図 7). 各エージェントには, 状態遷移と同時に状態遷移確率をもとにしたスコアを与えその累積を確信度とし, ある時点で最も確信度の高いエージェントの状態をその時刻で採用する. ユーザがある文字を入力した瞬間にシステムは, 全ての母音においてその母音が次の一手であったときに, 最もスコアが高くなるエージェントが指し示す文字を予測計算して表示する (図 2).

各エージェントはユーザが母音を入力する度に, 現在の状態 S_i から遷移できる状態 S_j のうちその母音を出力できるものの中で最も遷移確率の高い状態へ遷移する. また, 遷移と同時にその遷移元の状態 S_i から遷移しうる他の全ての遷移先のうち, 入力母音を生成できるものに対して新たなエージェントを生成して配置する. 各エージェントは遷移した状態が示す移動方法に従って歌詞中を走査する (図 7). 新たに生成されたエージェントのスコアは生成したエージェントのものがそのまま引き継がれる. 各エージェントはスコアが閾値を下回ると破棄される. あるエージェントがある時刻で他のエージェントと同じ歌詞位置となった場合は, その時点での確信度が低いほうのエージェントを破棄する.

各エージェントのある時刻 t のスコアは, HMM の状態遷移確率 $\tau(t)$ を用いて,

$$Score(t) = \prod_{n=t-t0}^{t} w(n)\tau(n), \tag{2}$$

$$w(-t0) = w(-t0+1) = \ldots = w(-1) = 1, \tag{3}$$

$$\tau(-t0) = \tau(-t0+1) = \ldots = \tau(-1) = 1 \tag{4}$$

で求められる. $t0$ はどこまで過去に遡ってスコアの累積を考慮するかを決定する定数である. 実験では過去 4〜8 状態程までの累積で十分であることが分かっている. あまり累積させ過ぎると大きな歌詞位置移動に対する追従性が悪化する. $w(t)$ はユーザによるコントロール性能を向上させるためのパラメータであり, 通常は 1 であるが, 以下の場合は異なる.

(1) 「ゆらゆら うごく」(母音列「うあうあ うおう」) のように, ある繰り返し有りセグメントの次のセグメントの先頭の母音が現在のセグメントの先頭母音と同じで, かつ異なる文字のとき, ユーザは入力する母音によって繰り返しをするか, 次へそのまま進むかを選択することができない. この場合, システムは元の歌詞での繰り返し回数を優先する. 具体的には, エージェントは繰り返した回数をカウントしておき, 元の歌詞での繰り返し回数を超えたときに $w(t)$ は 0.5 にする.

(2) 「ゆらゆら ゆれる」(母音列「うあうあ うえう」) のように, ある繰り返し有りセグメントの次のセグメントの先頭の母音が現在のセグメントの先頭母音と同じで, かつ同じ文字の場合, システムは「ゆ」の次に「あ」が入力されたか「え」が入力された時点で遅延評価することで対応できる. よってこの場合の $w(t)$ は 1 とする.

(3) 「ゆらゆら ゆかいな」(母音列「うあうあ うあいあ」) のように, ある繰り返し有りセグメントの次のセグメントの先頭の母音が現在のセグメントの先頭母音と同じで, かつ同じ文字で, さらに 2 つ目の母音が同じ場合, (1) と同様に処理する.

(4) 「ん」だけは独立したキーとして用意してあるため,

図 8 セットアップ

ユーザがこのキーを入力したときに「ん」を出力できるエージェントは確信度が高い．よってこの場合 $w(t)$ を 2 倍する．

さらに，3 文字程度連続して異なるエージェントが採用され続けた場合はユーザは全く別の場所に移動しようとしている可能性が高い．この場合，全てのエージェントを一度破棄した後，直前に入力された 3, 4 母音程の遷移が出現する歌詞中の全ての位置にエージェントを生成し直す．これは brute force なアプローチであるが，日本語の歌唱楽曲の歌詞は最も長いものでも一曲 1500 文字程度であるので計算コストが問題になることはない．オペラ等はさらに多くの文字数があるが幕によって分割することで対応可能である．待機させておくエージェントの総数は実験では歌詞の文字総数の 1/10 程度用意しておけば十分であった．

4. 実装

本稿執筆時点で，　リアルタイムで歌詞と音高を同時に入力して演奏できうるレベルのレイテンシで音声合成発音できる市販のシンセサイザーは，eVY1[18] およびその派生版である「ポケット・ミク」[19] のみである．実装したシステムでは Web ブラウザから Web MIDI API を使って USB 接続されたポケット・ミクを制御している．ポケット・ミクは発音し直しの場合にレイテンシが大きくなってしまうという問題があるが，複数台接続したポケット・ミクを発音の度に順番に切り替えて鳴らし，それをミックスすることで解決した．また，ノートオフに合わせて PC 側で息継ぎの音を鳴らして合成音声のリアリティを上げている．さらに，全ての文字に対して同じベロシティカーブを適用すると，聴感上の大きさにばらつきが出て不自然になってしまうため，母音単位で重み付けをユーザの演奏したベロシティに対して掛け合わせて対処した．母音列と音高を入力するためのキーボードは MIDI インタフェースで PC に接続され，キーボードの一部の C〜G を母音入力用に割いている (図 8)．歌詞の形態素分割と漢字のひらがな変換には MeCab[20] を用いた．

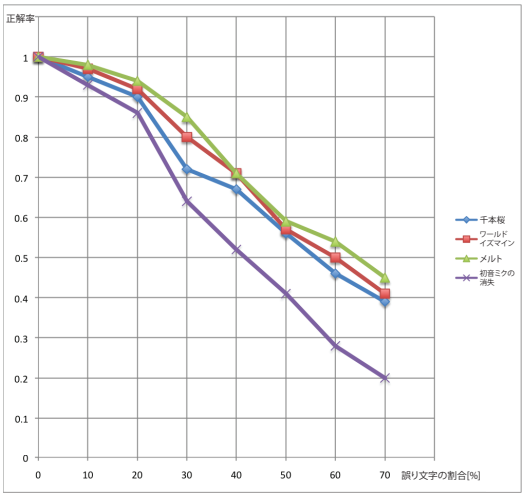

図 9 入力ミスが混入した場合の精度

曲名	作者	文字数	追従速度
千本桜	黒うさ P	616	3
ワールドイズマイン	supercell	628	2
メルト	supercell	432	2
初音ミクの消失	暴走 P	1373	4

表 1 実験に使用した楽曲と歌詞ジャンプ時の追従速度

5. 評価

5.1 アライメント精度

合成音声がメインボーカルとして用いられている楽曲 4 曲 (表 1) の歌詞を母音列に変換した後，途中一様乱数で生成した位置に一様乱数で生成した母音を挿入し，テストデータとした．ここで，誤り母音は 3 つ以上連続して配置されることを禁止した．このテストデータをシステムに入力した場合の歌詞の正解率を求めた．これは入力ミスに対する堅牢性に相当する．挿入する誤り母音の数の歌詞総文字数に対する割合と正解率の関係は図 9 のようになった．誤り文字の混入率が 30%を超えると急激に性能が低下している．「初音ミクの消失」の性能低下率が他に比べて著しいのは文字総数が 2 倍以上と，例外的に多いためである．しかし一方，10〜20%程度の入力ミスならばどの楽曲においてもほぼ 9 割以上の正解率を得ることができている．このことからも提案手法は入力ミスに対して堅牢に動作すると言える．

次に，元の歌詞の節単位で順番をランダマイズしてシステムに入力して，歌詞位置をジャンプしてから正しい歌詞に追いつくまでの誤り文字数の最大値を求めた．これは演奏位置の移動に対する堅牢性に相当し，数字が小さい程性能が良いことを示す．結果はこちらも楽曲の文字総数に依

存するが, 通常の歌唱楽曲なら最悪 2〜3 文字まで, 長大な楽曲でも最悪 4 文字遅れる程度で歌詞位置のジャンプに追従できている (表 1). よって実用上大きな問題になることはないと考えられる. さらに, 提案手法は, 誤り文字が出力された場合にも, 母音成分は演奏者の意図通りのものが出力されるため, その違和感は最小限に抑えられるというメリットもある.

5.2 演奏

実際にカラオケ曲として多くの人に歌われている「千本桜」(作詞/作曲 黒うさ P) を著者自らが練習し, オリジナルのテンポのまま演奏した (図 8). なお, 著者の鍵盤演奏技能はピアノ科の音大生レベルで音大受験生への指導実績もある程度である. 練習時間は一日 30 分程度× 1 週間である. この曲は, 歌唱合成が用いられている楽曲の中でもテンポがかなり速い曲で, かつ歌詞の文字数が多く文字の遷移スピードも最も速いものの一つである. よってこの曲が演奏できるならばほぼどんな楽曲でも演奏できると言ってよい.

実際の演奏の様子は (http://www.yamo-n.org/livo/) で公開しているが, オリジナルのテンポで問題無く演奏できていることが分かる. また, 旋律にはアドリブで装飾音などの変化を加えていたり, 途中でアドリブのスキャットも行っている. こうした表現は歌唱合成を用いたリアルタイムパフォーマンスならではのものであり, かつ従来の手法では実現できなかったものである. このことからも提案手法が新しい音楽表現にとって有用なものであることが分かる.

今後は筆者以外の一般的な鍵盤楽器経験者による演奏評価も行っていく必要がある.

6. まとめと今後の課題

本稿では, 母音列に対して事前情報として与えた歌詞の位置をアライメントすることで, リアルタイムに片手で実用的な歌詞操作を実現することのできるインタフェースを提案した. 演奏時の歌詞中の文字列進行は HMM でモデル化され, 入力ミスや任意の文字列の繰り返し, 大きな演奏位置移動に対しても堅牢に動作する.

しかし, 提案手法は日本語がモーラ言語であり, かつ母音がせいぜい 5 つ, 「ん」を合わせても 6 つ程度しか無いからこそ実現できたものである. よって, 英語をはじめとした多くの他言語へは提案手法は適用できないという問題がある. また, 現時点のシステムでは歌詞をどう階層構造化するかは, ユーザ側の入力に依存している. 区切り方によってはユーザの意図通りの出力が得られないこともあるため, システム側で自動的に最適な区切りを見つけ出すことは今後の課題である. さらに, 本稿の手法は経験則に基づいて決定されたパラメータが非常に多い. 自動的に最適なパラメータを学習できるようなフレームワークも求められている.

今後はソースコードをオープンソースで公開し, 多くのユーザが利用できるようにして提案手法を普及させていくと同時に, 実際にライブパフォーマンス等を行って実使用環境でのさらなる評価を行っていく.

参考文献

[1] ヤマハ株式会社: VOCALOID^{TM}, http://www.VOCALOID.com/.
[2] UTAU, http://utau2008.web.fc2.com
[3] ドワンゴ, ニコニコ動画, http://www.nicovideo.jp
[4] 山本和彦, 加々見翔太, 濱野桂三, 柏瀬一輝: リアルタイム日本語歌唱鍵盤楽器のための文字入力インタフェースの開発, 情報処理学会論文誌, Vol.54 No.4, p1373-1382 (2013).
[5] 佐近田 展康: 「兄弟式リアルタイム音声合成演奏システム」の概要と背景, 名古屋学芸大学メディア造形学部研究紀要 (2013).
[6] ヤマハ株式会社: 歌唱合成のための装置およびプログラム, 特開 2008-170592 (2008)
[7] ヤマハ株式会社: 歌唱合成制御装置および歌唱合成装置, 特開 2012-83569 (2012)
[8] 竹本 拓真, 馬場 隆, 片寄 晴弘: リアルタイムに初音ミクを歌わせるタイプソングシステム 「HANAUTAU」とそのアジャイル型開発事例報告, インタラクション (2014).
[9] ミクミン P: フリック入力, http://www.nicovideo.jp/watch/sm17357529
[10] Leah Findlater,1 Jacob O. Wobbrock,1 Daniel Wigdor, Typing on Flat Glass: Examining Ten-Finger Expert Typing Patterns on Touch Surfaces, ACM CHI (2011).
[11] Frank Chun Yat Li, Leah Findlater, Khai N. Truong.: Effects of Hand Drift while Typing on Touchscreens, Proceedings of GI (2013).
[12] Daniel Buschek, Oliver Schoenleben1, Antti Oulasvirta, Improving Accuracy in Back-of-Device Multitouch Typing: A Clustering-based Approach to Keyboard Updating,Proc. IUI'14, ACM Press (2014)
[13] 前澤 陽, 糸山 克寿, 吉井 和佳, 奥乃 博: 潜在共通構造モデルに基づく音響信号間アライメント, 情報処理学会音楽情報科学研究会, 2014-MUS-103 (2014).
[14] Dannenberg, R. B. et al.: Polyphonic Audio Matching for Score Following and Intelligent Audio Editors, ICMC (2003).
[15] Orio, N. et al.: Score Following: State of the Art and New Developments, NIME, pp. 36-41 (2003).
[16] Nakamura, T. et al.: Acoustic Score Following to Musical Performance with Errors and Arbitrary Repeats and Skips for Automatic Accompaniment, SMC, pp. 200-304 (2013).
[17] Masataka Goto and Yoichi Muraoka: Beat Tracking based on Multiple-agent Architecture - A Real-time Beat Tracking System for Audio Signals, Proceedings of The Second International Conference on Multiagent Systems, pp.103-110 (1996).
[18] eVY1 シールド, http://www.switch-science.com/catalog/1489/
[19] 学研, 歌うキーボード ポケット・ミク, 大人の科学マガジン (2014).
[20] MeCab: Yet Another Part-of-Speech and Morphological Analyzer, http://mecab.sourceforge.jp