論文 / 著書情報
Article / Book Information

題目(和文)	ジャンルと打拍フォームに基づく打楽器学習支援に関する研究
Title(English)	
著者(和文)	辻靖彦
Author(English)	
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第5836号, 授与年月日:2004年3月26日, 学位の種別:課程博士, 審査員:
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第5836号, Conferred date:2004/3/26, Degree Type:Course doctor, Examiner:
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

博士論文

ジャンルと打拍フォームに基づく
打楽器学習支援に関する研究

東京工業大学 大学院社会理工学研究科

人間行動システム専攻

00D40045

辻 靖彦

指導教官　西方敦博

目 次

1章

序論

1.1 研究背景

現在、小，中，高等学校の音楽の教科書では、童謡やクラシック以外にビートルズなどのポピュラー音楽が採用され始めている。従来の童謡やクラシックなどと同様に、ポピュラー音楽を用いた音楽教育、音楽の授業が今後ますます増加していくと考えられる。ポピュラー音楽という呼び方は非常に広い定義であり、その中には多種多様なジャンルの音楽が存在している [5]。本来、ポピュラー楽曲のジャンルとは音楽性のみならず、精神性、地域性、ファッションなどさまざまな要因が統合された一つのスタイルであり [5]、前衛的な音楽とは、ありきたりの耳慣れたコード進行や、ジャンルといった規制の枠から逸脱しようとするところに存在する [6]。そういった意味で音楽をジャンルの枠にはめてしまうことは危険なこととも言える。しかし、授業の現場などでポピュラー音楽を学習する際には、その楽曲のジャンルは切り離せないものであり、ジャンルごとにその音楽的特徴を学ぶことは、学習としてだけではなく、作曲や編曲の授業場面などでも有効であると考えられる。また、最近の音楽教育に対する研究事例として、いかにして音楽学習を進めるかというよりむしろ、音楽に興味の薄い生徒に対して、演奏、鑑賞の双方でどのように音楽経験を豊かにさせるかという方向へ進んでいる [11]。そういう意味でも、比較的生徒が興味を抱きやすいポピュラー楽曲に関して、ジャンルを通じて体系的に学習することは、生徒の音楽経験を豊かにさせるきっかけになりやすいと考えられる。また、生徒に興味を抱かせる意味では、最近になってコンピュータや MIDI （Musical Instrument Digital Interface）、DTM（Desk Top Music）を用いた授業も増加してきており、教育分野における学術的研究でも、そういったものを用いた研究が数多く行われている [11][13][14][52]。合唱の音楽授業で DTM を実践的に用いた例もある [12]。そういった背景をふまえると、音楽教育の分野におけるコンピュータへの期待は大きく、生徒に対しても、興味を抱かせるメディアの一つとしての期待を背負っていると言える。

一方、小、中、高等学校の音楽の授業において、ギターやドラムなどの、楽器演

奏を用いた授業が増加してきている。楽器演奏を音楽の授業で行なうことは、生徒が日常聴く楽曲を演奏できるため、生徒にとって親しみやすい授業内容と考えられる。また、楽器の演奏を音楽の授業で用いることは、音楽観賞、音楽の歴史の学習などの、座学中心の授業とは異なり、能動的な活動が求められるので、生徒に興味を持たせやすいと考えられる。ところで、楽器演奏においては、リズム能力が重要であると一般に言われている。楽器演奏の授業において、その指導方法に対してさまざまな実践的研究が行なわれている [50]。

以上より音楽教育的な背景として、

- ポピュラー音楽のジャンルを学習する、扱うことは音楽教育的に有益
- MIDI や DTM など、コンピュータを用いた音楽教育は、授業の多様化や、生徒のやる気を引き出す要素があるという意味で有益ということが上げられる。
- 楽器演奏の授業は、生徒に興味を持たせやすい

が上げられる。

音楽のジャンルとは、認知心理学でいうところの概念（concept）、つまりカテゴリー（category）にあたる。従って、楽曲をそのジャンルに分類することとは、認知心理学における「カテゴリー化（categorization）」であると言える。認知心理学においてカテゴリー化においては、さまざまな仮説が存在する。その中の主なものとしては、アリストテレスを起源とする、各要素が一連の特徴を持ち、すべての要素が平等であるとする Classical View、各カテゴリーがそのカテゴリーのすべての性格的特徴を含む理想像（プロトタイプ）を持つとする Prototype View（Posner & Keele,1968）、新しい要素を、それまで蓄えられた要素（事例）と比較することによってカテゴリー化する Exemplar View（Medin & Smith,1984;Mervis,1990）、各カテゴリーがスキーマであるとする Schemas View（Bartlett,1932:Rumelhart & Norman,1988）、カテゴリー化は人々の知識や世界観と関係があるとする Knowledge-Based View（Barsalou,1983;Keil,1989;Murphy & Medin,1985）、の５つが存在するが、いずれも問題を抱えており、普遍的な理論としては確立されていないのが現状である [3]。

以上のことを踏まえて、カテゴリーとして音楽のジャンルを考えると、それは特定の音楽的ルールによりすべてを記述できるものではなく、また、必ずしもその境界がはっきりとしているわけではなく、また、人の持つ知識によってその分類が異なる可能性があることが考えられる。こういったさまざまな問題を考えると、楽曲のジャンルへの分類という認知プロセスは一筋縄でいくものではなく、人間の観察が必要であると考えられる。

以上、認知心理学的背景として、

- 楽曲のジャンルは音楽的ルール、つまり規則がすべて明示できるものではない可能性がある
- ジャンルの境界は必ずしもはっきりしない可能性がある
- 人のもつ知識、文化、世界観などによって、楽曲のジャンルへの分類が異なる可能性がある
- 「人間が、楽曲をそのジャンルへ分類する」という認知プロセスを考えるには、さまざまな人間を対象にした何らかの実験が必要である

ということが上げられる。

現在、ポピュラー楽曲を扱った情報処理的研究としては、演奏の表情付け [41]、演奏者の演奏ルールの抽出 [43]、作曲や編曲支援 [39][44]、インタラクティブセッションシステム [28][29] など、さまざまな研究が行われている [1][2]。しかし、そういった研究の中で楽曲のジャンルそのものの特徴について扱っている研究は少ない。またジャンルを扱ってはいても、１つのジャンルに実験対象範囲を限定して行われている研究が多く見られる [28][29][31][41]。ポピュラー楽曲の複数のジャンルを扱っている研究では、南高ら（1989）の開発した、「曲風を考慮した自動作曲システム-MAGIC」が上げられる [44]。このシステムは、ジャンルなど曲風別のヒューリスティックなルールを用い、そのルールに即した曲風で作曲や編曲を支援するものである。このようなルールをベースとしたシステムは、システムにとって既知であるような曲風やルールを再現する際には有効であるが、生成される曲が一般的なものになってしまう懸念があり、また、システムの未知な曲に対する柔軟性に問題があると言える。また、音楽のジャンルとは楽曲を要素とする、一つのグループである。そして、楽曲をジャンルに分けることは楽曲のグルーピング（群化）であると言える。現在、音楽情報処理の分野において、グルーピングに関してはさまざまな研究が行われている。その中で主流になっているのが、ニューラルネットワークを用いたグルーピングである。具体的には、SD 法によって得られた、形容詞による曲印象などの感性情報によってグルーピングをする研究が主流である [34][37][39][40]。ここで、ニューラルネットワークとは、生物の脳内の電気信号によるシナプス間の情報伝達アルゴリズムであり、情報処理の分野では、それを人工的にコンピュータに模倣したものを指す [19][20]。ニューラルネットワークは非線形事象に対応し、未知曲の再生にも効果を発揮するので、ルールをベースとした手法や、判別分析などの線形的手法よりも汎用性の高いシステムの構築が可能である。しかし欠点として、処理の内部がユニット間の結合係数で表現されるため、ブラックボックス化されており、データ解析が困難な特徴がある [26]。しかし、中間層が 1 層の 3 層フィードフォワード型など、特定の構造を持つニューラルネットワークでは結合の重みに着目し、因子のネーミングを行い、意味を同

定するなどのデータ解析手法がいくつか存在 [22][23] し、実際にさまざまな研究で行われている [21][27][28]。

その他の研究例としては、計算機を用いた音楽学習の支援を目的とする研究が行なわれている。例えば、和音の学習を目的とした研究 [52] などがある。楽器演奏の支援を行なう研究の一つとしては、ドラムパッドによる人の演奏情報を処理しフィードバックする研究がある [53]。一般に打楽器（ドラム）の演奏においては、良い音を出すためにも、またリズムにおいて重要視されるいわゆるノリやグルーブ感を出すためにも、演奏の際のストロークフォームが重要である [57][58]。

以上、音楽情報処理的背景として、

- ポピュラー音楽のジャンル推定そのものを扱った研究はほとんどない
- ルールをベースにした方法での楽曲のジャンル推定は、未知な楽曲に対する柔軟性に問題がある
- ニューラルネットワークは、ルールベースや線形的な手法よりも未知な要素に柔軟性という意味で強力であると言うことが上げられる。
- 楽器演奏支援を行う研究の一つとして、ドラムパッドによる人の演奏情報を処理しフィードバックする研究があるが、一般に打楽器（ドラム）の演奏においては、演奏の際のストロークフォームも重要である

ということが上げられる。

1.2　本研究の目的

以上のさまざまな背景をふまえて本研究は、楽器演奏支援の一つとして、打楽器の学習支援を考える。具体的には、

- ポピュラー音楽のジャンルの特徴
- リズムと打拍フォーム

による打楽器学習支援を実現することを目的とする。

そのために、ジャンルに基づく打楽器学習支援の前提として、

- ポピュラー楽曲のジャンル推定

の実現を試みる必要がある。ニューラルネットワークをベースにした楽曲のジャンル推定システムの構築を第 1 段階の目的とする。その手法及び結果に基づき、ジャンルによる打楽器学習支援を考える。楽曲のジャンル推定を行うことができれば、その学習したニューラルネットワークの中間層から特徴を解析してユーザ

へフィードバックすることにより、学習支援システムを構築することが可能となるからである。

また、リズムと打拍フォームに基づく打楽器学習支援の前提として、

- 打楽器経験者と未経験者の演奏及び打拍フォームの違いの分析

を行なう必要がある。演奏及び打拍フォームの違いが明らかになれば、それを基にフィードバック手法を決定することにより学習支援システムを構築する事が可能となるからである。

尚、本論文で ”打楽器 ”とは、基本的にドラムを意味する。また、”打楽器経験者 ”とは、ドラム経験者のことをさす。しかし ”打楽器未経験者 ”は、パーカッションや太鼓などのすべての打楽器を対象に特別な訓練経験を持たない者をさす。”ドラム ”ではなく ”打楽器 ”という用語を使う理由は、2つある。1つは、本論文で分析に用いる MIDI の 10 チャンネルのパートは一般に打楽器パートと呼ばれるが、そのパートはドラムだけでなく、パーカッション全般を含むからである。2つ目としては、本論文で行なった被験者課題が、右手に持ったスティックによるドラムパッドの打拍に限定しているからである。”ドラム演奏 ”とは本来、右手による打拍のみならず左手、右足、そして左足を用いてスネアドラム、タム、そしてシンバル類を強弱をつけて叩くことにより、複雑な演奏を行なう。そのような意味では、本研究で扱った被験者課題はドラム演奏のごく一部を扱っているに過ぎない。その一方で、手に持ったスティックによる打拍動作は、撥 (バチ) を用いる打楽器に共通する基本動作であり、本論文で述べる手法はドラム以外のいくつかの打楽器にも応用可能であると考える。その可能性を考えて、本論文ではドラムではなく打楽器という用語を用いている。

1.3　本論文の構成

本研究の構成を図 1.1 で示す。

第 2 章では、ポピュラー音楽の熟達者はどのような要因でジャンルを推定するかを調査するために、演奏経験 10 年以上のポピュラー音楽を熟達している被験者を対象に、インタビュー方式の予備実験を行う。ジャンルを推定する要因は、楽曲の局所的情報の組み合わせであるが、ルールを抽出するのは困難であることを示す。さらに、本研究のジャンル推定システムについて、図を交えて述べる。本システムは、楽曲のスタンダード MIDI ファイルを入力とし、推定したジャンルを出力とする。さらに、本システムを構成する 2 つの柱である、「特徴量抽出モジュール」と「ニューラルネットワークによるジャンル推定モジュール」について述べる。

第 3 章では、本システムと、被験者の両方にスタンダード MIDI ファイルの楽曲に対して、ジャンルを推定させる実験を行う。本システムによるジャンル推定

実験では、演歌、ジャズ、ハードロック、ヘヴィメタルの、各ジャンル30曲ずつ、合計120曲のスタンダードMIDIファイルから特徴量を抽出し、それをニューラルネットワークに学習させジャンルの同定を行い、同定を行うには、「楽器と音色」と「リズムの分布」と「音名の分布」の3種類を組み合わせた場合のみ、学習が成り立つことを示す。さらに、線形的手法の判別分析を各曲の特徴量を説明変数として同様にジャンルの同定を行い、ニューラルネットワークによるジャンルの同定結果と比較をして、ニューラルネットワークによる手法の優位性を示す。被験者によるジャンル推定実験では、音楽経験が平均的と思われる大学生25名を対象に、スタンダードMIDIファイルの楽曲8曲を聴取させ、選択方式でジャンルを理由と共に推定させる。さらに、演奏経験年数、鑑賞の自己尺度などの音楽経験と、推定正解率との相関や、被験者がジャンルを答えた際の確信度とジャンルごとの推定正解率との相関を調べる。そして、被験者によるジャンル推定実験で用いた8曲に対して、それを未知曲として本システムに認識させた際のジャンル推定率を求め、被験者と本システムのジャンル推定正解率の比較を行い、本システムの優位性を最後に示す。

第4章では、3章で行なったニューラルネットワークによるジャンル推定実験の際、120曲を学習したニューラルネットワークに対して、内部の各リンクに蓄えられている重みの情報を分析し、因子のネーミングを行なう。

第5章では、打拍における教示の効果を示す目的で、打楽器経験者による、演奏する際の打拍フォームに関する教示が打楽器未経験者の演奏に及ぼす影響について調べた。打楽器未経験者3名を被験者として、教示なしで1分間のリズムパターン演奏を8回続けて行なった場合と、打楽器経験者の教示を与えた上で8回続けて演奏させた場合の比較を行なった。

第6章では、フォームの改善により間接的に演奏者の演奏能力を向上させるために、簡易モーションキャプチャシステムを用い、打楽器演奏時の腕の位置情報の取得を試みた。そして演奏時の手首の位置情報波形データから3種類の特徴量を抽出し、打楽器経験者と未経験者によって演奏の際の右腕のストローク波形に違いがあるのかを統計処理によって分析した。分析には、始めに特徴量ごとに、経験者と未経験者とのグループ間でのt検定を行ない、特徴量ごとの有意差を定量的に求めた。続いて、ステップワイズ法による判別分析を行ない、両グループ間での線形的な判別を行ない、正準判別関数係数を調べ、判別の際の特徴量の組合せについて検討し、両グループにどのような波形が有意に見られるかを検討した。そして、被験者の演奏の分布を可視化する目的で、得られた3種類の特徴量からクラスター分析を行ない、演奏の分布を観察した。

第7章では、打楽器演奏時の手首の位置情報と、打拍時刻、打拍の強さ等の演奏情報の同時取得を行う。そして前章と同様に、取得データからストローク誤差、振幅RMS値、リズム誤差、ベロシティ（打拍の強さ）の4種類の特徴量を抽出し、打楽器経験者と未経験者によって演奏の際の右腕のストローク波形及び演奏に違

いがあるのかを統計処理を用いて分析する。処理には、経験者と未経験者とのグループ間での t 検定を行ない、特徴量ごとの有意差を定量的に求める。

さらに、ストローク波形と演奏に関連があるかどうかを調べるために、ストローク誤差とリズム誤差、振幅 RMS 値とベロシティの各平均値に関して、相関係数を求めて散布図を調る。

第 8 章では、打楽器演奏能力と、ジャンル推定能力の関連性について検討する。打楽器演奏能力とジャンル推定能力の相関性を仮に示せれば、打楽器演奏の支援だけでなく、ジャンルを学習支援することも打楽器学習において有効である事が示唆されると考えられる。始めに、前章で行なった実験の被験者の一部を対象に、打楽器パートのみのジャンル推定実験を行う。そして打楽器経験者と未経験者において、打楽器演奏能力とジャンル推定正解率とを比較する。

第 9 章では、6、7 章の結果を基に、音長の偏り、ストロークフォームのずれ、手の振り幅、打拍の強さをフィードバックする打楽器学習支援システムの開発及び評価を行う。

第 10 章では、本研究で得られた結果をまとめ、今後の課題を述べる。

以上が本論文の構成である。

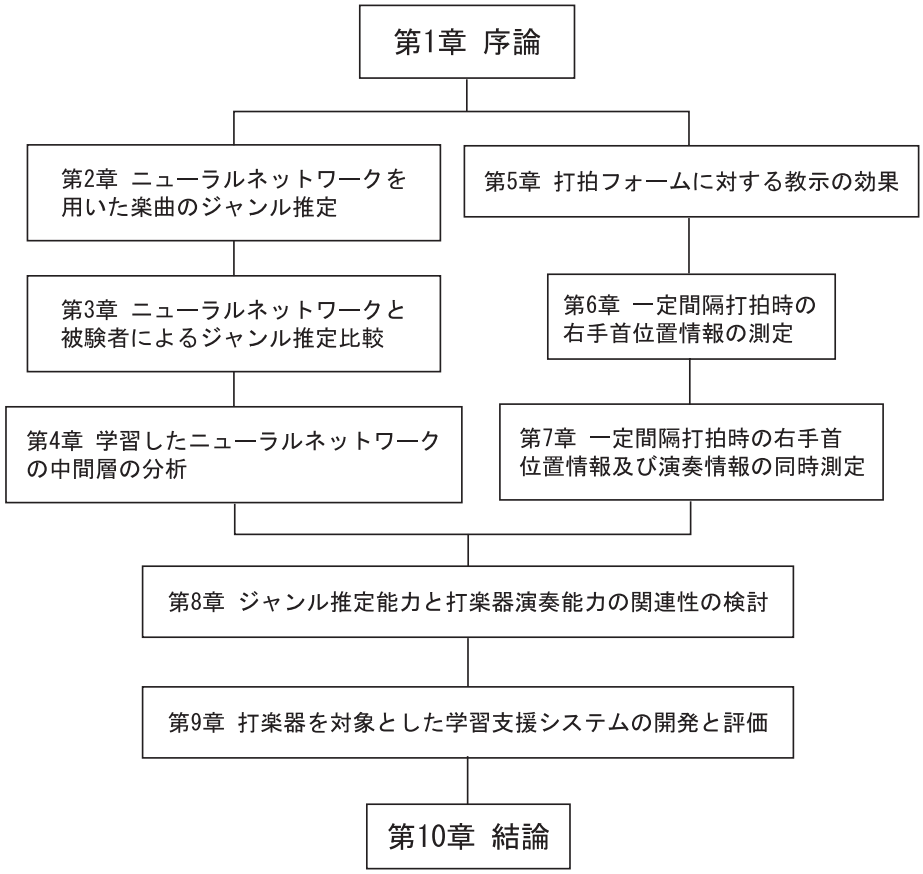

第1章 序論
第2章 ニューラルネットワークを用いた楽曲のジャンル推定
第5章 打拍フォームに対する教示の効果
第3章 ニューラルネットワークと被験者によるジャンル推定比較
第6章 一定間隔打拍時の右手首位置情報の測定
第4章 学習したニューラルネットワークの中間層の分析
第7章 一定間隔打拍時の右手首位置情報及び演奏情報の同時測定
第8章 ジャンル推定能力と打楽器演奏能力の関連性の検討
第9章 打楽器を対象とした学習支援システムの開発と評価
第10章 結論

図 1.1: 本論文の構成

2章

ニューラルネットワークを用いた楽曲のジャンル推定

2.1　はじめに

本章では、楽曲のジャンルとしての特徴をフィードバックさせる学習支援を目標とした、コンピュータによってジャンル推定を行わせるためのシステムについて述べる。2.3 で本システムの構成、2.4 で特徴量抽出モジュール、2.5 でジャンル推定モジュールについて述べる。

2.2　予備実験

楽曲のジャンルの特徴をコンピュータで抽出するためには、コンピュータで楽曲を正確に分類できることが必要である。従って、コンピュータで楽曲のジャンルを分類することをまず考えなければならない。そのために、本章では、ポピュラー音楽の経験が豊富な人、いわゆる熟達者はどのような要因でジャンル推定という認知プロセスを行うかを調査するために、ポピュラー音楽の熟達者を被験者とした、インタビュー方式による予備実験を行った。

続いて、実験方法、実験の結果、本章のまとめを述べる。

2.2.1　方法

本実験は以下の手続きで行われた。

被験者　ポピュラー音楽の経験が充分にあると思われる、楽器演奏経験 10 年以上の 4 名

実験手続き　ポピュラー音楽の経験が豊富だと思われる被験者に、ハードロック、演歌、ジャズなどさまざまなジャンルに関して、そのジャンルだと感じる

要因について思うところをインタビューした。サンプル資料として、5ジャンルで合計10曲のWAVEファイル（CD）、ポピュラーコード理論の文献、ジャンルについて書かれた文献[4][5][46][47][48][49]を用意した。

2.2.2　結果

実験の様子をDigital Videoで撮影し、実験終了後にプロトコル分析し、ジャンルに関して述べている部分的発話を抽出した。その結果を基にして、以下のような知見が得られた。

知見1 ジャンルを推定する要因は、部分的な情報であることが多い

具体例

- Ｉ「(ハードロックは、ドラムが）８ビートで‥。」
  Ｔ「ミディアムテンポで…。(コード進行で）無理な展開が無くて‥.」
  Ｉ「あとは‥(ギターが）リフ中心の構成っていうか、コードでこう‥進んでいくんじゃなくて、ギターとかのリフで‥。」

- Ｉ「あとは、聞いた感じがハードっていう…やっぱり。ドラムの音だったり、ギターの音だったり‥。」
  Ｔ「音色が結構あるね。」

- Ｔ「ジャズを見破るのは‥やっぱ、あれじゃん。ドラムで、スネアが１，３（泊）にちゃんと入っていない曲を‥。ちゃんと入っている曲は少ないんじゃないかな。」
  私「１，３って‥２，４（拍）??」
  Ｔ「あ、１，３じゃない。２，４。」
  Ｉ「それ入れたら多分ジャズっぽく聞こえなくなる。」
  Ｔ「それは他のジャンルには絶対に見られないよね。」

知見2 ジャンルを推定する要因である部分的情報は、曖昧な情報が多い

具体例

- Ｉ「あとメロディ的にはやっぱり‥、あんまりメロディっていう感じ、要素はポップスとかに比べたら少ないと思うけど。何か曖昧な言い方

だけど‥。音の上がり下がりっていうか、そういう流れていく感じではない。」

- Ｉ「やっぱりベースとドラムでほとんどジャズっぽいって感じるはずだけど…。」
- M「音使い的な面で。例えばハードロックの特徴はきっと…」
  Y「ペンタ（ペンタトニックスケール）ですか??」
  M「うん、ペンタなのかな。でもペンタだと‥。ペンタは、進行が決まってて、だから、ツェッペリン（レッドツェッペリン）とかは、かなりブルーズの進行をそのまま持ってきて使ってるよね?? だからブルーズとハードロックがどう分かれるかっていうのが難しいところなんだけど、あれは何で聞いてパッと分かるんだろう?? 親父臭いからかなあ。親父臭いってどこからくるんだろう（笑）??」
  Y「めちゃめちゃ曖昧じゃないですか（笑）。」

- Ｉ「この悲しげなアコースティックギターは何か、演歌っぽいよね。」
- M「どのジャンルの曲もシングルカットしようとすると、コード進行とか似てくる。」
  Y「似てくるっていうか、良いのは（ジャンルとして特徴的なのは）出てこないかもしれない‥。」

知見 3 ジャンルを推定する要因は個人個人によって異なる

具体例

- Ｉ「(ハードロックの特徴として）あとは、聞いた感じがハードっていう…やっぱり。ドラムの音だったり、ギターの音だったり‥。」
  T「音色が結構あるね。」

- M「…　俺は、どっちかっていうと、その、音作りとかでは、あまりジャンル分けを推定しない人間なんですよ．コード進行とか、乗ってるメロディの乗り方とか、リズム、とかでジャンル分けをするという立場からお話しするんで…。　…」

この発話から、被験者 M は楽器や音色といったことからはジャンル推定を考えないと言っているのに対して、被験者 T と I は、楽器の音色から判断する点もあると述べているので、個人個人によって要因が異なっている事が分かる。

- （ポピュラー音楽のジャンルについて書かれた本を見ながら）
  M「それも納得いかねえなぁ。でもこうやってやっぱり、誰かがなんとかって言っても納得いかない人がでてくるんだろうな。」

知見 4 ジャンルを推定する要因は、常に寄与するとは限らない

具体例

- T「この曲ってメジャー（メジャーコード）じゃない？」
  I「演歌っていうとマイナー（マイナーコード）っていうイメージがあったけど‥。」

- T「マイナー（マイナーコード）のドミナントモーションが演歌っぽさ…別に演歌でしか使っていないわけじゃないんだけど、バッハとかも普通に使われているんだけど、演歌でも良く使われる。っていうか演歌っていうと日本の心みたいに言われるじゃん。でも音楽的には全然日本的ではなくて、西洋的なんだ。クラシックに近いというか‥。で、（演歌は）乗っかっているメロディがペンタトニックで、コブシがきいている…。」

知見 5 人はジャンルを要因の組み合わせで判断している

具体例

- 私「ジャンル分けは全体的にどう考えたら良いですか？」
  M「コード進行っていういのがまずあるでしょ？そして、リズムっていうかビートと…使っている楽器っていうのも確かにあるけど。やばい発想が貧困だ、これしか思い浮かばない…。」
  Y「テンポじゃ無理ですよね.」
  M「あ、でも実はテンポも入るかも。」
  Y「演歌で 180 とか無いし…(笑)。」

知見 6 ジャンル間の境界は曖昧である。従って人間にとって難しい

具体例

- 私「パンクとハードロックって区別つくかな？」
  I「難しいところだな‥。」

私「コードでは特徴ある？」
T「ペンタ（ペンタトニック）は使われてる？」
I「何かね、（ハードロックよりも）明るい感じだね（笑）。」

以上の結果から考察すると、ジャンルを推定する要因であるところの部分的情報というのは曖昧な情報が多く、そのためジャンルの特徴をすべてルール化することは極めて難しい。従って、完全にルールに基づいてジャンルを推定するということは、推定という意味では非常に困難であり、かつ向いていないのではないかと考えられる。

そこで本研究では、曖昧な情報、感性的な情報を扱うのに向いていると言われているニューラルネットワークを利用することを提案する。具体的には、楽曲のスタンダードMIDIファイルからジャンル推定に必要だと思われる部分情報を抽出し、それをNNの入力値とする。その理由は、予備実験において、ジャンルを推定する要因のほとんどが、局所的、部分的な情報であるというインタビュー結果を得たからである。

2.3　本システムの構成

本研究の目標とするジャンル推定システムの構成を図に示す。流れとしては次のようになっている。

システムの流れ

1. ユーザが特徴を調べたい楽曲のスタンダードMIDIファイルを入力する
2. 特徴量抽出モジュールにより、スタンダードMIDIファイルからいくつかの特徴量を抽出する
3. その特徴量を、既に楽曲とそのジャンルについて学習させたニューラルネットワークへ入力する
4. システムは、ニューラルネットワークによって出力された結果（つまり、推定したジャンル）をフィードバックする

ただし、あくまで本システムの最終目標はジャンルを推定することではない。学習させたニューラルネットワークからその楽曲のジャンルとしての特徴をフィードバックさせることを目標としている。その学習したニューラルネットワークの中間層を分析した結果を用いることにより、推定したジャンルだけでなく、入力した楽曲のジャンルとしての特徴もフィードバックすることが可能となる。そし

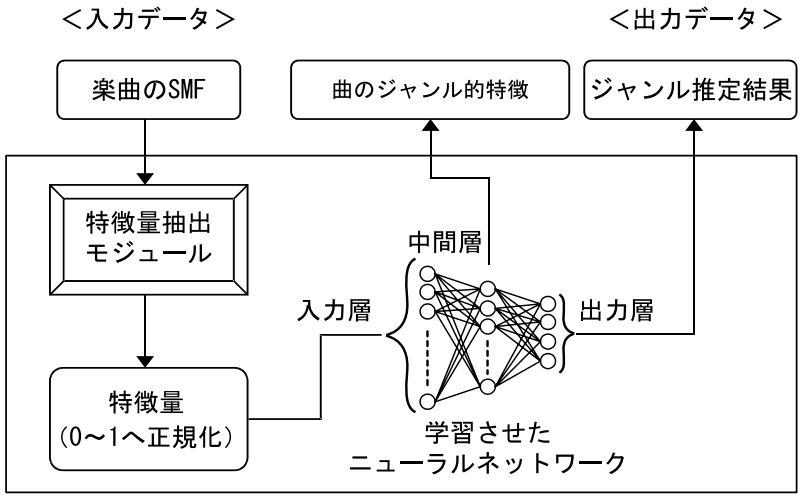


図 2.1: システム構成図

てこのことによりユーザはジャンルについての学習をすることができると考えられる。

2.4　特徴量抽出モジュール

本節では、スタンダード MIDI ファイル形式の楽曲のファイルを入力すると、ニューラルネットワークの入力情報として適切な、その楽曲の部分的、局所的な情報を抽出するモジュールについて説明する。

本研究で言うところの特徴量とは、スタンダード MIDI ファイルの楽曲データの中でジャンルの推定に重要だと思われる、楽曲の局所的な情報の事を指す。楽曲からそういった部分的なデータのみを抽出し、ニューラルネットワークに学習させることにより、人間が楽曲を聞いてその楽曲のジャンルとしての特徴を学習してゆくというプロセスと近い、もしくは同等のこと模倣できるのではないかという考えに立っている。

本研究の特徴量抽出モジュールでは、これらの楽譜情報のうちからジャンルを決定するのに重要と思われる局所的情報を抽出している。具体的には、プログラムナンバーから抽出した楽器の種類と音色（以下 楽器と音色）、ノートオンの拍節における位置の統計から抽出した１小節あたりのリズムの分布（以下 リズムの分布）、ノートナンバーから抽出した音名の分布（以下 音名の分布）、の３つの情報を本モジュールでは使用する。これらの特徴量は、ニューラルネットワークの入力値に対応させるために、0～1 の範囲で正規化した。なお、本研究では General Midi 対応のスタンダード MIDI ファイルを用いている。また、各モジュールを開発するのに、SunOS 5.6 上で C 言語を用いた。

スタンダードMIDIファイル形式のファイルには、ノートオン（各音譜の鳴り始めの時刻)、ノートオフ（鳴り終わりの時刻)、ベロシティ（音譜の強弱)、ノートナンバー（音の高さ)、プログラムナンバー（楽器の種類と音色)、などさまざまな音楽的情報が、直接的な音の波形データとしてではなく、言語として記述される形で含まれている。

その中で、本研究で扱うスタンダードMIDIファイル形式のデータの種類は以下の通りである。

- プログラムナンバー
- ノートオンの位置情報
- ノートナンバー

続いて、本研究で扱う3種類の特徴量について述べる。

2.4.1　楽器と音色

表面的な情報といえるかもしれないが、第2章の予備実験の結果から、楽器とその音色は楽曲のジャンルを推定する上で重要な情報であるとの知見が得られた。

ここでは、本論文で「楽器と音色」と名付けた特徴量の抽出の手法について述べる。楽器と音色は、楽曲のスタンダードMIDIファイルの中から、MIDIイベントの一つである、プログラムチェンジ（PC）という命令を参照し、チェンジされるプログラムナンバーを読み出して、そのファイルで用いられているプログラムナンバーを調べている（図2.2参照)。

ここで、プログラムナンバーとは、楽器の種類と音色を1〜128の数字で表現したものであり、大まかな楽器としての分類として、General MIDI仕様では、表3-1の様になっている。一つの数字で楽器の種類だけではなく、その音色も表している。例えば、25〜32のプログラムナンバーは、ギターの楽器におけるそれぞれの音色を表している（表2.1参照)。

プログラムナンバーが31であれば、それはDistortionGt（ギターのディストーションという音色）を指す。その表し方として、整数型変数inst[1]〜inst[128]の128個の配列を用いる。それぞれの変数は、そのプログラムナンバーの楽器が曲中で使われていれば1、使われていなければ0の値を取る。0と1以外の値は用いない。例えば、先ほどあげたギターのディストーションという音色がその楽曲で使われていればinst[31]=1、使われていなければinst[31]=0と表現する。

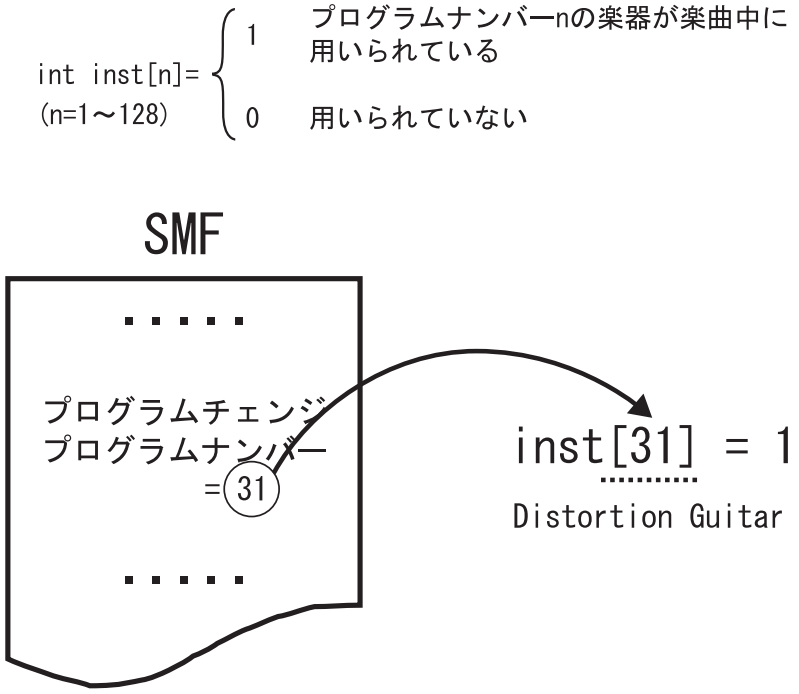


図 2.2: 楽器と音色

2.4.2　リズムの分布

本項では、「リズムの分布」と名付けた特徴量の抽出の手法について述べる。リズムの分布は、スタンダード MIDI ファイルの中から、楽器の打鍵情報であるノートオンの MIDI イベントを探し、その 1 小節における位置を 16 分割で抽出し、それをその楽曲全体を通して加えたものである（図 2.3 参照）。

具体的には、実数型変数 beat[0]～beat[15] の 16 個の配列を用い、それぞれの変数が、1 小節中のそのタイミングでノートオンイベントがその楽曲の中で行われた回数を表す。例えば beat[0] には小説の 1 泊目の頭にノートオンイベントがその楽曲で行われた回数、がカウントされる。これも他の特徴量と同様に、学習させるニューラルネットワークの入力に適切であるように、0～1 の値へ正規化を最後に行っている。これにより、その楽曲全体を平均化したリズム、つまりノリがどのようなものかが分かる。

2.4.3　音名の分布

本項では、「音名の分布」と名付けた特徴量の抽出の手法について述べる。音名の分布は、スタンダード MIDI ファイルからノートオンイベントを参照し、その中のノートナンバーという情報を抽出する（図 2.4 参照）。ノートナンバーとは、中

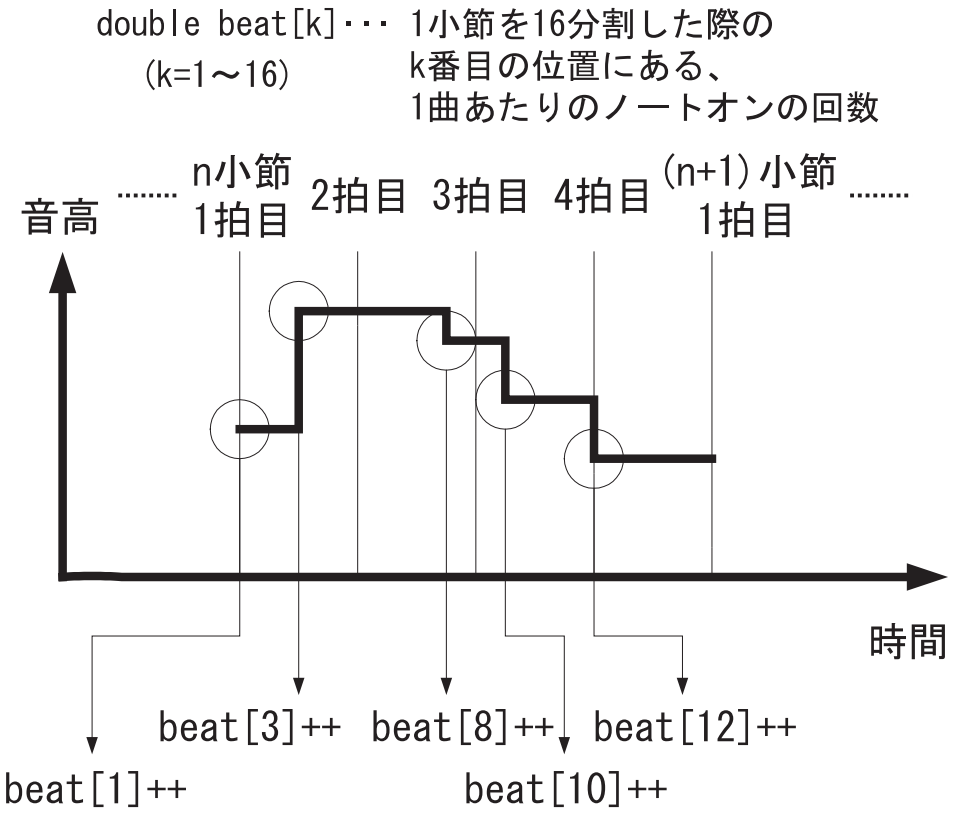

double beat[k]･･･ 1小節を16分割した際の
(k=1～16)
k番目の位置にある、
1曲あたりのノートオンの回数
音高
n小節
1拍目
2拍目
3拍目
4拍目
(n+1)小節
1拍目
時間
beat[1]++
beat[3]++
beat[8]++
beat[10]++
beat[12]++

図 2.3: リズムの分布

表 2.1: General MIDI Sound Set の概要

プログラムナンバー	楽器の名前
1〜8	piano
9〜16	chromatic percussion
17〜24	organ
25〜32	guitar
33〜40	bass
41〜48	strings & orchestral instruments
49〜56	ensemble
57〜64	brass
65〜72	reed
73〜80	pipe
81〜88	synth lead
89〜96	synth pad
97〜104	synth sfx
105〜112	ethnic misc
113〜120	percussive
121〜128	sfx

央C（C4）を60とし、1〜128の数字で鍵盤の鍵の位置を表したものである。ノートナンバーn（n＝1〜128）から音階n（n=1〜12）を抽出するには以下の式で求めることができる。

具体的には、整数型変数note[1]〜note[12]の12個の配列を用い、12音階のそれぞれの音がその楽曲中で使われた回数を表している。例えば、note[1]は、「ド（C）」の音がその楽曲中で使われた回数を表している。和音の場合は、その使われた音を別々にすべてカウントする方法を本研究ではとっている。

2.5　ジャンル推定モジュール

本項では、ジャンルを推定するニューラルネットワークの入力情報、出力情報、また、ニューラルネットワークの構成について述べる。

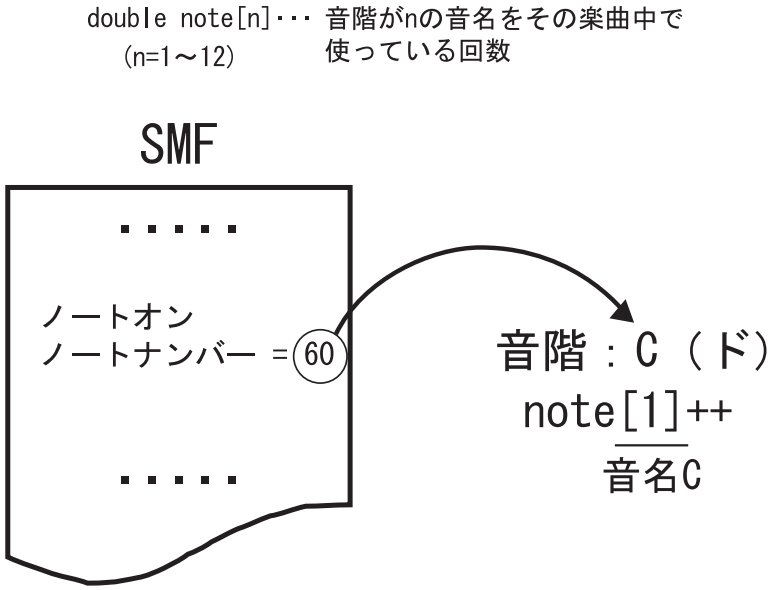


図 2.4: 音名の分布

表 2.2: ジャンルの割り当て方

教師信号	ジャンル
10000	演歌
01000	ジャズ
00100	ハードロック
…	…

2.5.1　入力情報と出力情報について

本研究で用いられるニューラルネットワークの入力情報は、前項で抽出した特徴量を用いる。具体的には、inst[n] で 128 個のユニット、beat[n] で 16 個、note[n] で 12 個のユニットで、合計 156 個の入力ユニットを用いる。これらの値はすべて 0～1 の値に正規化されている。また、出力情報、教師信号は推定するジャンルを表す。具体的には、表 2.2 のようにジャンルと数字を対応させ、1 つのユニットが 1 つのジャンルを表すやり方を本研究では採用した。

2.5.2　ニューラルネットワークの構成

本研究で用いるニューラルネットワークの構成は図 2.1 で見た通りである。これは、後に中間層からのジャンルとしての特徴を抽出することを容易にするために、中間層が 1 層の 3 層階層フィードフォワード型を採用しており、誤差逆伝播法（Back Propagation Algorithm）で学習する。伝達関数にはシグモイド関数

$$f(x) = \frac{1}{1 + e^{-x}}$$

を用いる。入力情報は前項で述べたように、General MIDI 対応のスタンダード MIDI ファイルから抽出した以下の特徴量、楽器とその音色、リズムの分布、音名の分布、の 3 つを用い、教師信号にはジャンルの種類を用いて学習させる。収束の判定には 2 乗誤差平均を用いた。

2.6　2 章のまとめ

本章では、目標とする本システムの概要を、図を交えて説明した。

本システムは、入力に楽曲のスタンダード MIDI ファイル、出力にその楽曲の推定したジャンルと、その楽曲のジャンル的特徴がユーザにフィードバックされる。

すなわち流れとしては以下のようになる。

システムの流れ

1. ユーザが特徴を調べたい楽曲のスタンダード MIDI ファイルを入力する
2. 特徴量抽出モジュールにより、スタンダード MIDI ファイルからいくつかの特徴量が抽出される
3. その特徴量を、すでにある程度の楽曲とそのジャンルについて学習させたニューラルネットワークへ入力される
4. システムは、ニューラルネットワークによって出力された結果（つまり、推定したジャンル）をフィードバックする

さらに、スタンダード MIDI ファイルから局所的情報である特徴量を抽出するための、特徴量抽出モジュールについて説明した。そのモジュールでは、スタンダード MIDI ファイル形式の楽曲から、以下の特徴量を抽出することにより、「楽器と音色」、「リズムの分布」、「音名の分布」の 3 種類の情報を獲得している。

- プログラムナンバー

- ノートオンの位置情報
- ノートナンバー

また、本システムで用いている、ニューラルネットワークをベースとしたジャンル推定モジュールについて説明した。

3章

ニューラルネットワークと被験者によるジャンル推定比較

3.1　はじめに

本章では、第2章で述べた本研究のモジュールによるジャンル推定と、人間の被験者によるジャンル推定実験の結果の比較を行う。仮に本モジュールで用いた、学習したニューラルネットワークが人間の被験者よりも高い確率でジャンルを推定できるならば、本モジュールの学習支援システムとしての有用性が示唆されたと考えられる。始めに、本モジュールによるジャンル推定実験に関して述べる。

3.2　本システムによるジャンル推定実験

ここでは、本モジュールによるジャンル推定実験の目的、手続き、結果と考察について述べる。

3.2.1　実験の目的

本実験の目的は、第2章の予備実験の結論に基づいて、人間がポピュラー楽曲のジャンルを推定するのに重要だと思われる部分的、局所的情報（特徴量）をニューラルネットワークに学習させることにより、楽曲のジャンルを同定させ、人間が楽曲を聞いて、その局所的な情報によりジャンルを推定するという一連のプロセスをコンピュータに模倣させることである。ニューラルネットワークに学習させきることができれば、楽曲のジャンルとしての特徴が入っていると思われる中間層を解析することにより、特徴をフィードバックさせることができると考えられる。

3.2.2　実験の手続き

ここでは、本モジュールによる実験の手続きを述べる。

実験手続き

演歌、ジャズ、ハードロック、ヘヴィメタルの各ジャンル30曲ずつ、合計120曲のスタンダードMIDIファイルから、「楽器と音色」、「リズムの分布」、「音名の分布」の3種類の特徴量を抽出し、楽曲ごとに特徴量とそのジャンルをニューラルネットワークに学習させ、ジャンルの同定を行った。なお、ニューラルネットワークへ学習させる特徴量はそれぞれ、

＜1＞「楽器と音色」のみ

＜2＞「リズムの分布」のみ

＜3＞「音名の分布」のみ

＜4＞「楽器と音色」と「リズムの分布」

＜5＞「リズムの分布」と「音名の分布」

＜6＞「楽器と音色」と「音名の分布」

＜7＞「楽器と音色」、「リズムの分布」と「音名の分布」

の7種類で行った。

3.2.3　実験結果と考察

始めに、中間層の個数を固定して、＜1＞～＜7＞のニューラルネットワークに対してそれぞれ120曲の特徴量と各曲のジャンルを学習させた。その結果、「楽器と音色とリズムの分布」を入力情報としたネットワーク＜4＞と、「楽器と音色、リズムの分布、音名の分布」を入力情報としたネットワーク＜7＞の2つにおいて、2乗誤差平均が収束し、学習が完了した（表3.1）。

続いて中間層に関する実験と考察を行う。表3.1によると、＜4＞、＜7＞以外のニューラルネットワークは学習回数が1万回を超えても学習しきらなかった。この結果より、少なくとも＜4＞、＜7＞のニューラルネットワークは、この4ジャンル合計120曲のジャンルを推定するのに、必要最低限の特徴量を含んでいることが分かる。続いて、＜4＞、＜7＞のニューラルネットワークにおいて、最適な中間層の個数を求めるために、個数の値を変動させてそれぞれニューラルネットワークに学習を行わせた。その結果が図3.1と表3.2である。

表 3.1: ニューラルネットワークによる同定結果

入力値（又は説明変数）	入力層-中間層-出力層	同定割合	同定率	収束状況
＜1＞ 楽器と音色	128-20-4	119/120	99.20%	X
＜2＞ リズムの分布	16-10-4	97/120	80.10%	X
＜3＞ 音名の分布	12-10-4	103/120	85.80%	X
＜4＞ 楽器と音色、リズム	144-30-4	120/120	100%	約 650
＜5＞ リズム、音名	28-30-4	111/120	92.50%	X
＜6＞ 楽器と音色、音名	140-30-4	119/120	99.20%	X
＜7＞ 楽器、リズム、音名	156-30-4	120/120	100%	約 1100

注）X … 1 万回を超えても収束しない

この結果によると、学習しきったニューラルネットワークの中で最も少ない中間層のユニット数は、どちらも 5 個であることが分かる。この結果から、この 120 の楽曲の中で、ジャンルを推定する特徴、要因となっているものは 5 つの因子であることが推測できる。この＜4＞、＜7＞のニューラルネットワークの 2 乗誤差平均の学習状況は、図 3.2 のようになった。ここで、この＜4＞、＜7＞のニューラルネットワークのように、本当に「楽器と音色」、「リズムの分布」、「音名の分布」を組み合わせて学習させる必要があるのかを確かめるために、その他の＜1＞、＜2＞、＜3＞、＜5＞、＜6＞のニューラルネットワークにおいて、中間層の個数を 5 個に固定して学習を行わせた。その結果が表 3.3 である。

この結果により、＜1＞、＜2＞、＜3＞、＜5＞、＜6＞のニューラルネットワークが学習しきらなかったことから、この各ネットワークは、ジャンルを推定するのに必要な特徴量を入力情報に含んでいないことが分かる。従って結論として、この 120 曲でジャンルを推定するのに必要な特徴量は、「楽器と音色」、「リズムの分布」の 2 つか、「楽器と音色」、「リズムの分布」、「音名の分布」の 3 つを組み合わせたものである、ということができる。

3.3　被験者によるジャンル推定実験

ここでは、被験者によるジャンル推定実験の目的、手続き、結果について述べる。

3.3.1　実験の目的

本実験の目的は、システムによるジャンルの推定率を比較するために、一般的な被験者はどのくらいポピュラー音楽のジャンルを推定することができるのかを

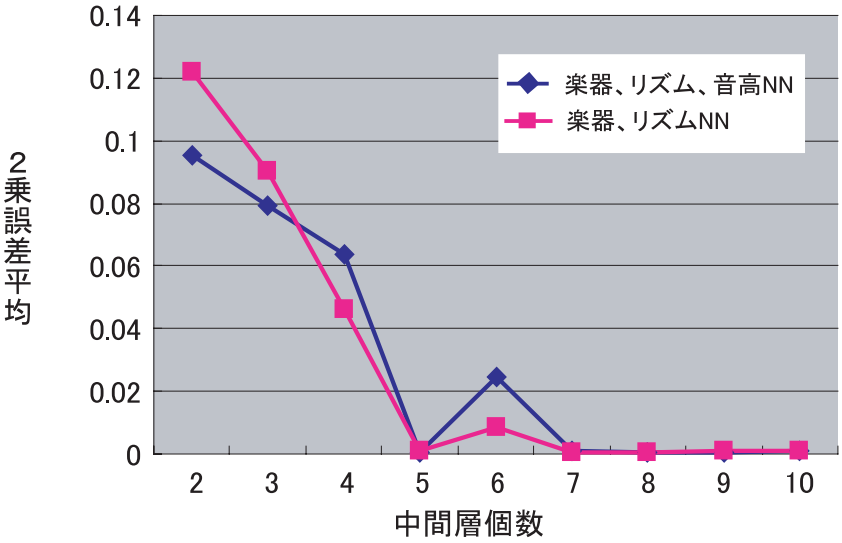


図 3.1: 中間層各ノード数による学習の収束

表 3.2: 各中間層の個数による学習の収束状況

中間層個数	＜7＞2乗誤差平均	学習回数	＜4＞2乗誤差平均	学習回数
2	0.095464	X	0.122113	X
3	0.079253	X	0.090409	X
4	0.063746	X	0.046333	X
5	0.000418	2873	0.000782	5444
6	0.024687	X	0.008408	X
7	0.000938	429	0.000445	3774
8	0.000706	1596	0.000602	692
9	0.000700	2641	0.000899	1355
10	0.000879	369	0.000887	203

注）X … 2万回を超えても2乗誤差が収束しない

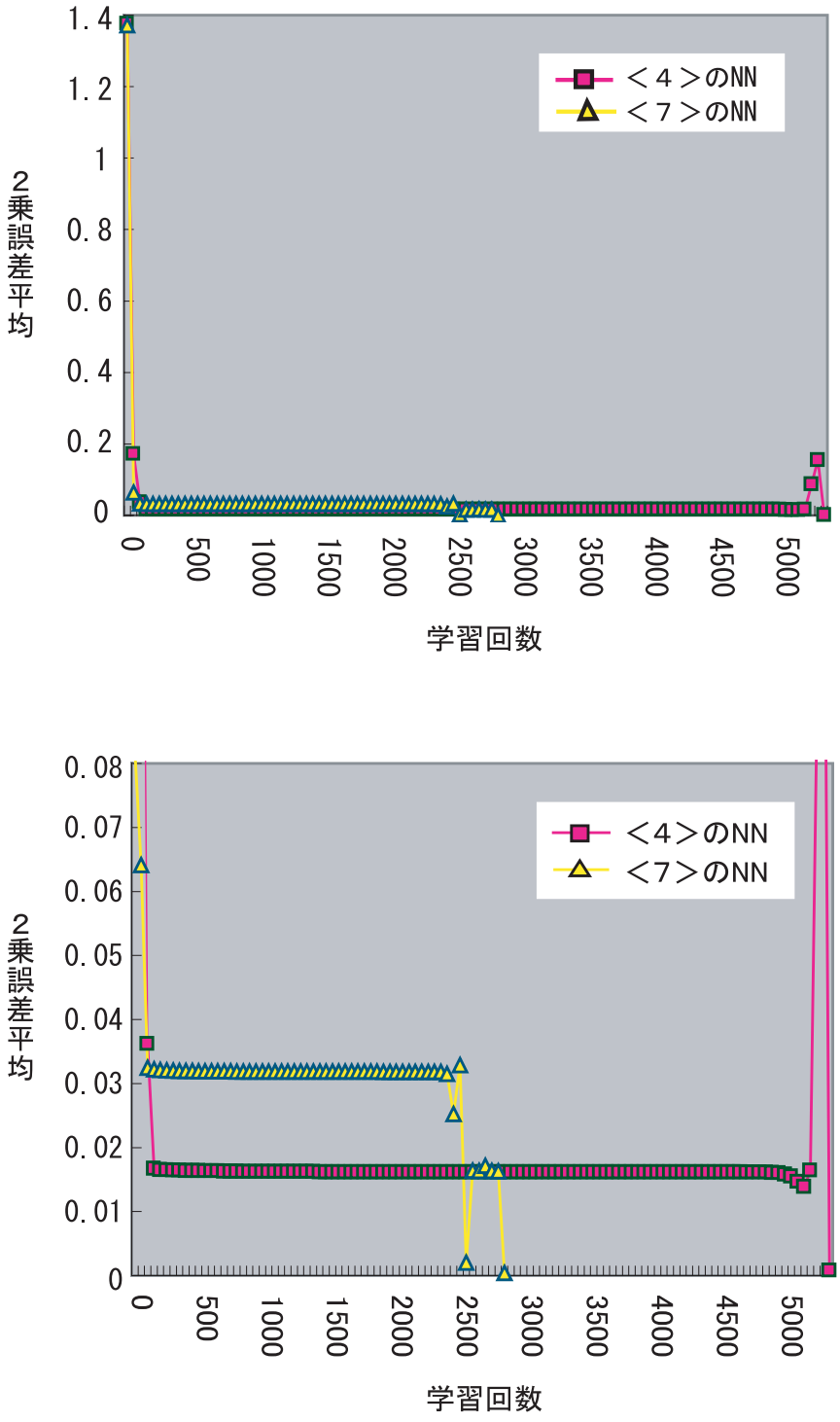


図 3.2: ニューラルネットワークの 2 乗誤差平均の変化
（上図と下図は、縦軸の縮尺が異なる同一のグラフである）

表 3.3: 中間層が5個の各ニューラルネットワークの学習

入力の特徴量	入力層	中間層	出力層	同定割合	同定率	収束状況
＜１＞楽器と音色	128	5	4	119/120	99.20%	X
＜２＞リズムの分布	16	5	4	94/120	78.33%	X
＜３＞音高の分布	12	5	4	71/120	59.17%	X
＜４＞リズム、音高	28	5	4	93/120	77.50%	X
＜５＞楽器と音色、音高	140	5	4	119/120	99.20%	X

注）X … 学習回数が1万回を超えても収束しないことを表す

調べることを中心に置いている。その他の目的として、一般的な被験者は、人はどのようなプロセス、又は要因で楽曲のジャンルを推定するのかを調べること、個人の音楽的経験がどのくらいジャンルの推定率と関連があるのかを調べること、ジャンルを推定する際の自信（確信度）がジャンルによってどのように違うのかを調べることが上げられる。

3.3.2　実験手続き

被験者　女子大学生25人

実験手順　はじめに被験者個人の音楽経験をアンケートに記入させた。個人の音楽経験とは、楽器の演奏経験、持っているCDの枚数、普段よく聞く音楽のジャンルの5段階評価、作曲経験などである。その後で、演歌、ジャズ、ハードロック、ヘヴィメタルの4ジャンルの各ジャンル2曲ずつ、合計8曲のスタンダードMIDIファイルをMIDI音源によりランダムな順番で被験者に聴取させた。そして各曲について、そのジャンルを推定させ、推定したジャンルをその要因、理由と共にアンケートに記入させた。要因は、予備実験の結果に基づいた、ジャンルを推定するのに有効だと思われる部分情報、楽器の種類、コード進行、フレーズ、打楽器のパターンの4項目についての5段階尺度で被験者に答えさせた。

3.3.3　実験結果と分析

全体の推定率は表3.4のようになった。全体では66.50%、ジャンル別には、演歌が94%、ジャズが78%、ハードロックが54%、ヘヴィメタルが40%となった。

続いて、比較に入る前に、ジャンルの推定が成功した際の要因を調べるために、

表 3.4: 全体のジャンル推定率

	演歌	ジャズ	ハードロック	ヘヴィメタル	合計
正解数/問題数	47/50	39/50	27/50	20/50	133/200
正解率（%）	94%	78%	54%	40%	66.5%

個人個人のジャンルの正解率と事前にとったアンケートの音楽経験との相関を求めた。具体的には、楽器の経験年数と正解率との相関、4 ジャンルを良く聞くかの 5 段階尺度のすべての平均と全体の正解率の相関、持っている CD の枚数と全体の正解率の相関を調べた。

その結果、Peason の相関係数はそれぞれ 0.037、0.238、0.298 となり、いずれの場合も相関は見られなかった。これにより、楽器の経験年数、4 つの各ジャンルの音楽を良く聞くかという自己申告の尺度、持っている CD の枚数が必ずしも直接的にジャンルを推定する事に結びつくとは限らないことが示唆された。

さらに、ジャンル推定が正解した際の要因の分布と、不正解であった際の要因の分布との有意差の検定を行った。まずは要因の分布の平均を求めたところ、演歌の要因平均の分布は図 3.3、ジャズの要因平均の分布は図 3.4、ハードロックの要因平均の分布は図 3.5、ヘヴィメタルの要因平均の分布は図 3.6 のようになった。

正解、不正解の場合における、各要因の平均に関して、別々に t 検定によって有意差を調べたところ、ハードロックとジャズにおいて、「楽器の種類」という要因を、不正解者よりも正解者の方が、有意に重くとらえている傾向のあるという結果が得られた（それぞれ両側検定で 5%有意傾向）。

そこで、個人個人のデータに対して、要因の分布を正解と不正解に分類し、判別分析を行った。しかし、要因の分布から正誤への判別の結果、有意な結果は得られなかった。このことから、今回選び出した、ジャンル推定の際の 4 つの要因の分布、は直接正誤には結びつかない、もしくは音楽経験が一般的な被験者の内省には結びつかないことが示唆された。

続いて、正解時における確信度の分布の違いを求めた。始めに、ジャンルの推定が正解した場合の確信度の値の分布を、ヒストグラムで表したところ、図 3.7 のようになった。続いて、この 4 ジャンルのそれぞれの確信度の分布に対して、一元配置の分散分析を行い、さらに多重比較を行うことによって、それぞれのジャンルごとに確信度の分布の違いを調べた。その結果、ジャズ—演歌、演歌—ヘヴィメタルに対して、それぞれ有意水準が 1%、5%の有意差が見られた（表 3.5、表 3.6）。これにより、被験者によるジャンルの推定が、同じ正解した場合であっても、ジャズと答えた場合と演歌と答えた場合、また、演歌と答えた場合とヘヴィメタルと答えた場合、において、確信の度合いが異なっていたことが示唆された。

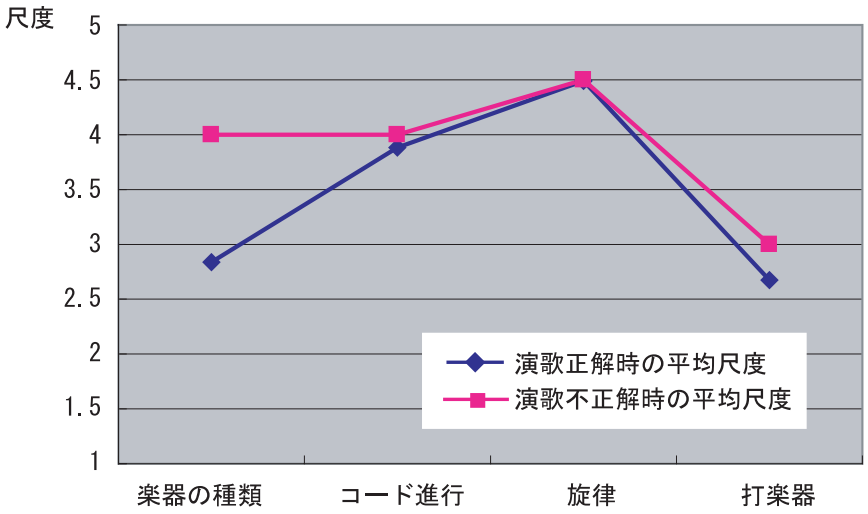
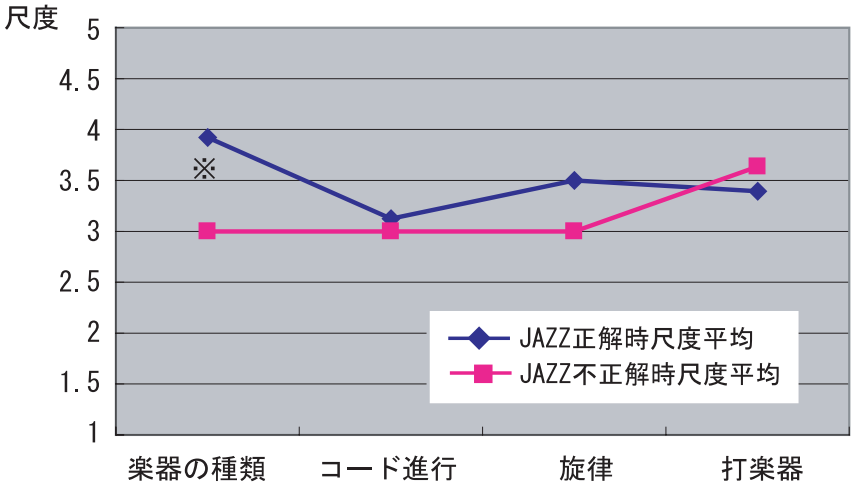
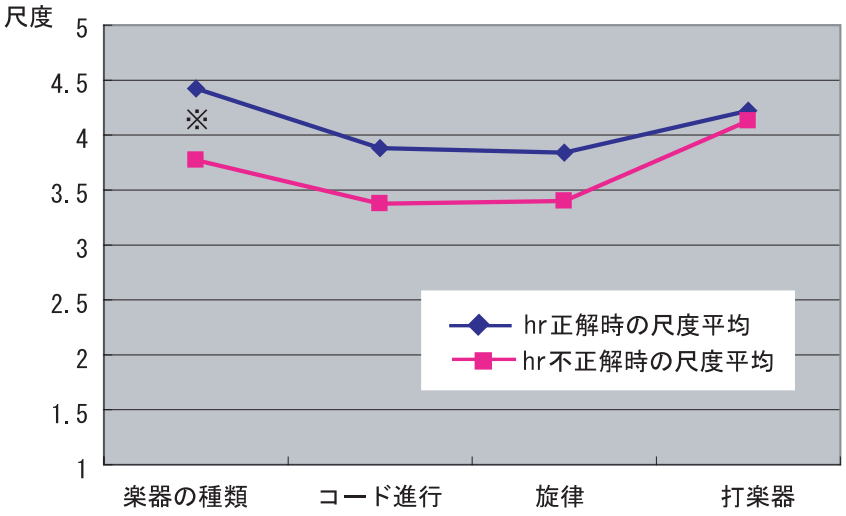


図 3.3: 演歌判定要因の平均分布



※…5%有意傾向

図 3.4: ジャズ判定要因の平均分布



※…5%有意傾向

図 3.5: ハードロック判定要因の平均分布

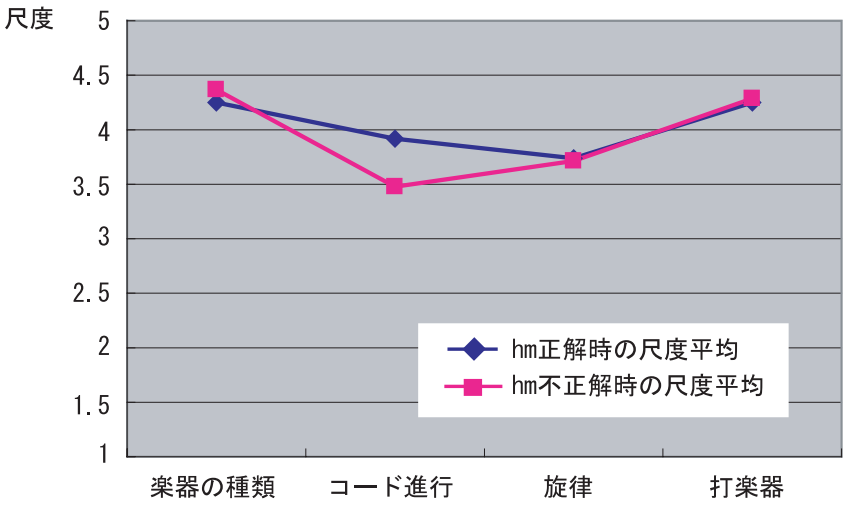


図 3.6: ヘヴィメタル判定要因の平均分布

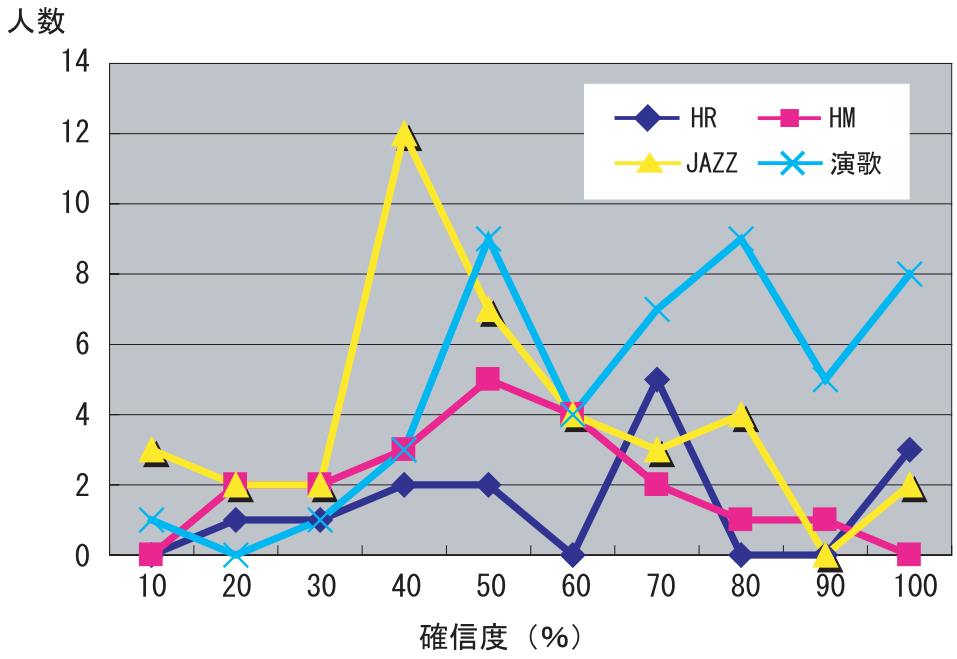


図 3.7: 正解時の確信度のヒストグラム

表 3.5: 分散分析結果

確信度

	平方和	自由度	平均平方	F 値	有意確率
ｸﾞﾙｰﾌﾟ間	11589.392	3	3863.131	8.169	.000
ｸﾞﾙｰﾌﾟ内	61000.699	129	472.874		
合計	72590.090	132			

3.4　本システムと被験者のジャンル推定比較

本節では、3.3.3 節で行った被験者による楽曲のジャンル推定結果と、本モジュールによるジャンルの比較を行う。本モジュールの方が推定率が優れていれば、将来的に楽曲のジャンルとしての特徴をフィードバックするためのジャンルの推定という意味で、本モジュールの優位性が示されることになる。

3.3.3 節より、被験者によるジャンルの推定率は 66.5%である。

ここで一つ考慮しなければならないことは、この被験者による推定率を何と比較するかということである。3.3.3 節で用いた楽曲 8 曲は、3.2 節で用いた本モジュールの推定実験に用いられたスタンダード MIDI ファイル形式の 120 曲中の 8 曲である。従って、先ほどの推定率を 3.2 節の学習したニューラルネットワークに推定させても、それはもともと 8 曲を学習しているので、当然ながら 100%である。本モジュールの学習し切ったニューラルネットワークは、人間でいうと、さまざまなジャンルの楽曲をたくさん聞いたことのあるポピュラー音楽の熟達者にあたる。従って学習したニューラルネットワークに学習曲を再認識させてその特徴をフィードバックさせるというプロセスは、ポピュラー音楽の熟達者にその人の知っている曲についてそのジャンルとしての特徴を述べてもらう事に等しい。そういう意味では、被験者による推定率と、学習したニューラルネットワークによる推定率と比較しても、学習支援システムとして優位性は一応示されたと言える。しかしながら、概してニューラルネットワークによるシステムは、汎用性を示すために、未知な入力に対する結果を求める実験が多く行われている [34]。従って、本システムも汎用性を調べる意味で、未知なる楽曲に対して、推定が有効であるかどうかを調査する必要がある。そこで本章では、3.3.3 節で用いた楽曲 8 曲を、120 曲の楽曲から 1 曲ずつ差し引いた 119 曲を学習させ、残りの 1 曲を未知曲として推定させる、という手続きの実験を行った。

本節では、3 章で説明した本モジュールを用いて、3.3.3 節の被験者によるジャンル推定実験で用いた、演歌、ジャズ、ハードロック、ヘヴィメタルの 4 ジャンル、合計 8 曲のスタンダード MIDI ファイルを、未知な楽曲としてニューラルネットワークへ認識させたジャンル推定実験について述べる。そしてその結果と、3.3.3 節の被験者実験による推定率との比較を行う。実験手続き

3.2 節で行ったジャンル推定実験で用いた楽曲 120 曲のスタンダード MIDI ファイルのうち、3.3.3 節の被験者によるジャンル推定実験で用いた楽曲を 1 曲ずつ未知曲として差し引き、差し引いた 119 曲でニューラルネットワークに学習させ、そして差し引いた 1 曲を認識させ、ジャンルを推定させる。これを 8 曲に対して同様に繰り返す。

その結果、本ニューラルネットワークによる推定率は、表 3.7 のようになった。これより、音楽知識が一般的な被験者によるジャンルの推定率よりも、本システムの方が推定率が高いことが分かる。これにより、ジャンルの推定という意味で

本システムの優位性が示唆されたと言える。この結果は、ニューラルネットワークが学習することにより、何らかのジャンルの特徴をつかんだことを示しており、その特徴が含まれていると思われる中間層を分析することにより、本研究で目標とする学習支援以外にも、作曲や編曲支援、曲検索システムなどさまざまな応用性が考えられる。また、曲印象などの感性情報を利用したそういった研究は実際に行われている [33][34]。

3.5　3章のまとめ

本章では、本システムと被験者のそれぞれに対して、同じスタンダード MIDI ファイルを用いてジャンルの同定及び推定実験を行った。

3.2 節で行った楽曲のジャンルへの同定実験では、演歌、ジャズ、ハードロック、ヘヴィメタルの 4 ジャンルで合計 120 曲のスタンダード MIDI ファイルの部分情報を学習させ、ジャンルを同定させることができた。なお、特徴量は、「楽器と音色、とリズムの分布」の 2 つを入力情報とした場合、と、「楽器と音色、と、リズムの分布、と音名の分布」の 3 つを入力情報とした場合のニューラルネットワークにおいてのみ同定できた。3.3 節で行った音楽経験が一般的な被験者 25 人に対して、演歌、ジャズ、ハードロック、ヘヴィメタルの 4 ジャンル合計 8 曲のスタンダード MIDI ファイルを聴取させてジャンルを推定させ、その要因を答えさせる、という手続きの推定実験を行った。その結果、全体の正解率は 66.5%の結果が得られた。

さらに、ジャンルを推定した 4 つの要因の分布を正解、不正解の 2 つに分け、判別分析を行ったが有意な結果は得られなかった。これより、ジャンル推定の際の 4 つの要因の分布、は直接正誤には結びつかない、もしくは音楽経験が一般的な被験者の内省には結びつかないことが示唆された。また、ジャンルを推定した要因と確信度の関係を分散分析と多重比較によって調べた結果、ジャズ—演歌、演歌—ヘヴィメタルに対して、有意確率 1 %、5 %で有意差が得られた。これにより、ジャンルを推定する際に、同じ正解した場合においても、ジャズと答えた場合と演歌と答えた場合、演歌と答えた場合とヘヴィメタルと答えた場合、において答えに対する自信が有意に異なっていたことが示唆された。

3.4 節では、3.3 節で得られた被験者による推定率と、本システムによる未知な楽曲を認識させた際の推定率とを比較し、本システムの有用性を示した。

続いて第 4 章では、学習させたニューラルネットワークの中間層の重みの分析を行う。

表 3.6: 多重比較

従属変数: 確信度
Tukey HSD

(I) ジャンル	(J) ジャンル	平均値の差 (I-J)	標準誤差	有意確率	95% 信頼区間	
					下限	上限
ジャズ	演歌	-21.8538*	4.710	.000	-33.9545	-9.7531
	ヘヴィメタル	-2.6410	5.981	.971	-18.0056	12.7236
	ハードロック	-10.7151	5.444	.200	-24.7013	3.2711
演歌	ジャズ	21.8538*	4.710	.000	9.7531	33.9545
	ヘヴィメタル	19.2128*	5.806	.005	4.2980	34.1275
	ハードロック	11.1387	5.251	.146	-2.3518	24.6292
ヘヴィメタル	ジャズ	2.6410	5.981	.971	-12.7236	18.0056
	演歌	-19.2128*	5.806	.005	-34.1275	-4.2980
	ハードロック	-8.0741	6.415	.589	-24.5555	8.4073
ハードロック	ジャズ	10.7151	5.444	.200	-3.2711	24.7013
	演歌	-11.1387	5.251	.146	-24.6292	2.3518
	ヘヴィメタル	8.0741	6.415	.589	-8.4073	24.5555

*. 平均の差は .05 で有意.

表 3.7: 全体のジャンル推定率

	被験者	＜4＞楽器と音色、リズム	＜7＞楽器、リズム、音名
推定率	66.5%	100.0%	100.0%

4章

学習したニューラルネットワークの中間層の分析

4.1　はじめに

本章では、3.2 節で行った、本システムによるジャンル推定実験で学習させたニューラルネットワークの中間層について分析を行う。　ニューラルネットワークの学習は、ノード間のリンクの重み付けの変化によって行われる。本研究で採用した 3 層のパーセプトロンの場合、中間層は入力層と出力層の両方から直接つながれている。

中間層の分析とはつまり、入力層と中間層、そして中間層と出力層の間のリンクの重み（weight）について調査することである。従来行われている手法としては、中間層の各ノードに対する発火促進と抑制のそれぞれを起こしやすい傾向を課題の専門家の協力により見出し、因子の解釈を得ようとするヒューリスティックな手法 [21][22] が主流であり、教育分野でも応用されている [20]。本研究でもその手法を用いる。

4.2　学習したニューラルネットワークの中間層の分析

始めに、中間層と出力層の間の辺の重みに着目すると、そのグラフは図 5-1〜5-5 のようになる。入力層と中間層の間の辺の重みに対して、＋ 10 以上の重みだけに着目すると、ノード 1〜5 は、それぞれ、次のジャンルに起因している。

ノード 1 ハードロック

ノード 2 ハードロック、ジャズ

ノード 3 ジャズ、ハードロック、演歌

ノード 4 ヘヴィメタル

ノード 5 演歌

続いて、この中間層の各ノードにおいて専門家によるネーミングの検討を行う。

各ノードのネーミング

ノード 1 図 4.1 よりこの因子は、ハードロックに対して正の重みを持っている。専門家によると、図 4.6 より、この中間層のノードは Distortion Guitar、Overdriven Guitar、などのハードな音色の入力ノードとの重みが大きな値となっている。また、抑制にあたる入力ノードを見ると、Fretless Bass、Trumpet、AltSax、Acoustic Bass 等、比較的静かな、落ち着いた雰囲気の音楽で扱う楽器が多く見られる。

以上の専門家の意見を踏まえて、ノード 1 は『ハードな音色因子』と名付けた。

ノード 2 図 4.2 より、ノード 1 に比べると微弱であるが、この因子もハードロックに対して促進の重みを持っている。図 4.7 を見ると、連結しているリンクの重みが大きな入力ノードはどちらもシンセサイザーの音色であるところの、Lead2(Sawtooth)、Lead8(Bass+Lead) の 2 つである。

従って、ノード 2 は『シンセサイザー音色因子』と名付けた。

ノード 3 図 4.3 より、ノード 3 はハードロック、ジャズ、演歌の 3 つに対して促進の重み、つまり正の値の重みを持っている。図 4.8 を見た専門家によれば、このノード 3 に対して促進の結合を持つ入力ノードは、Acoustic Grand Piano、Acoustic Bass、String Ensemble などのアコースティックな楽器が多い。その他の楽器（音色）においても、Electric Guitar(jazz)、Synth Bass2 など、比較的穏やかな音色である。

従って、ノード 3 は『ジャズ・アコースティック因子』と名付けた。

ノード 4 図 4.9 より、この因子における、結合された入力ノードは、促進、抑制共に、特徴量『リズムの分布』の値が数多く存在する。

従って、ノード 4 は、『リズム傾向因子』と名付けた。

ノード 5 図 4.10 より、この因子においても結合された入力ノードにおいて、促進、抑制共に『リズムの分布』から得た値が多く存在する。また、促進の結合となっている入力ノードを見ると、Electric Bass(Finger)、Distortion Guitar、Electric Bass(Pick) などのエレキギター、ベースといった楽器が見られる。

以上のことを踏まえて、ノード 5 は『エレキ楽器とリズム傾向の組み合わせ因子』と名付けた。

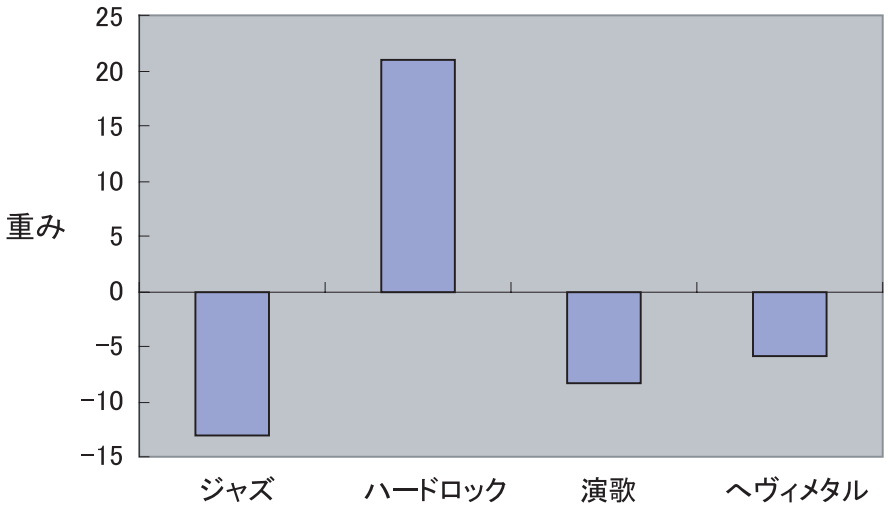


図 4.1: ノード 1 から各ジャンルへの重み

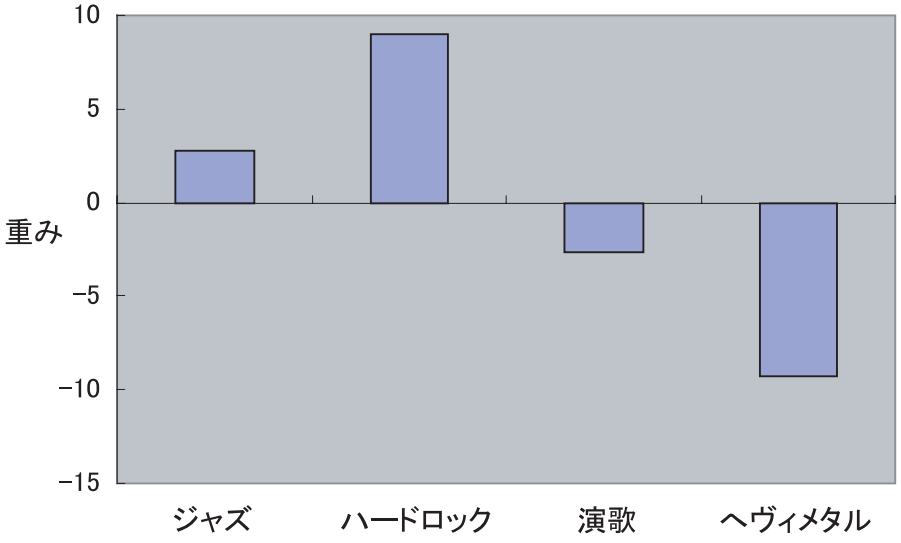


図 4.2: ノード 2 から各ジャンルへの重み

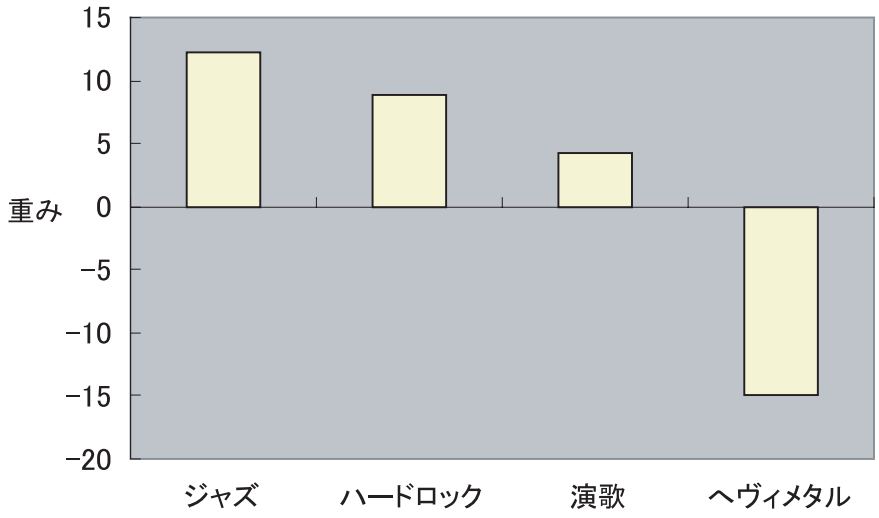


図 4.3: ノード 3 から各ジャンルへの重み

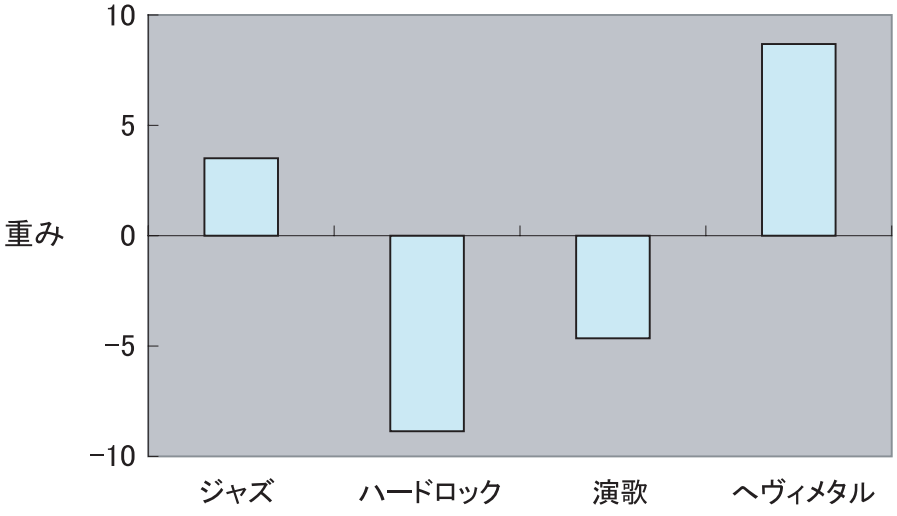


図 4.4: ノード 4 から各ジャンルへの重み

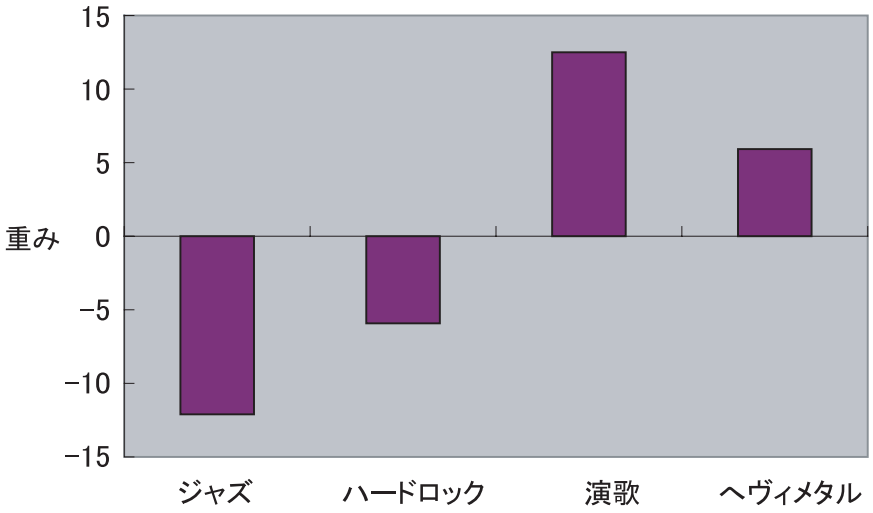


図 4.5: ノード 5 から各ジャンルへの重み

4.3　4章のまとめ

本章では、3.2 節で 120 曲学習させたニューラルネットワーク（『楽器と音色』、『リズムの分布』の 2 つの特徴量を入力とするネットワーク）に対して、中間層の特徴を、専門家の意見を交えて解析した。その結果、『ハードな音色因子』、『シンセサイザ音色因子』、『ジャズ・アコースティック因子』、『リズム傾向因子』、『エレキ楽器とリズム傾向因子』の 5 つの因子を抽出できた。

この結果より、本研究で構築した楽曲のジャンル推定システムを用いることにより、音楽教育のためのジャンル学習支援システムの構築することへの可能性が示されたと言える。

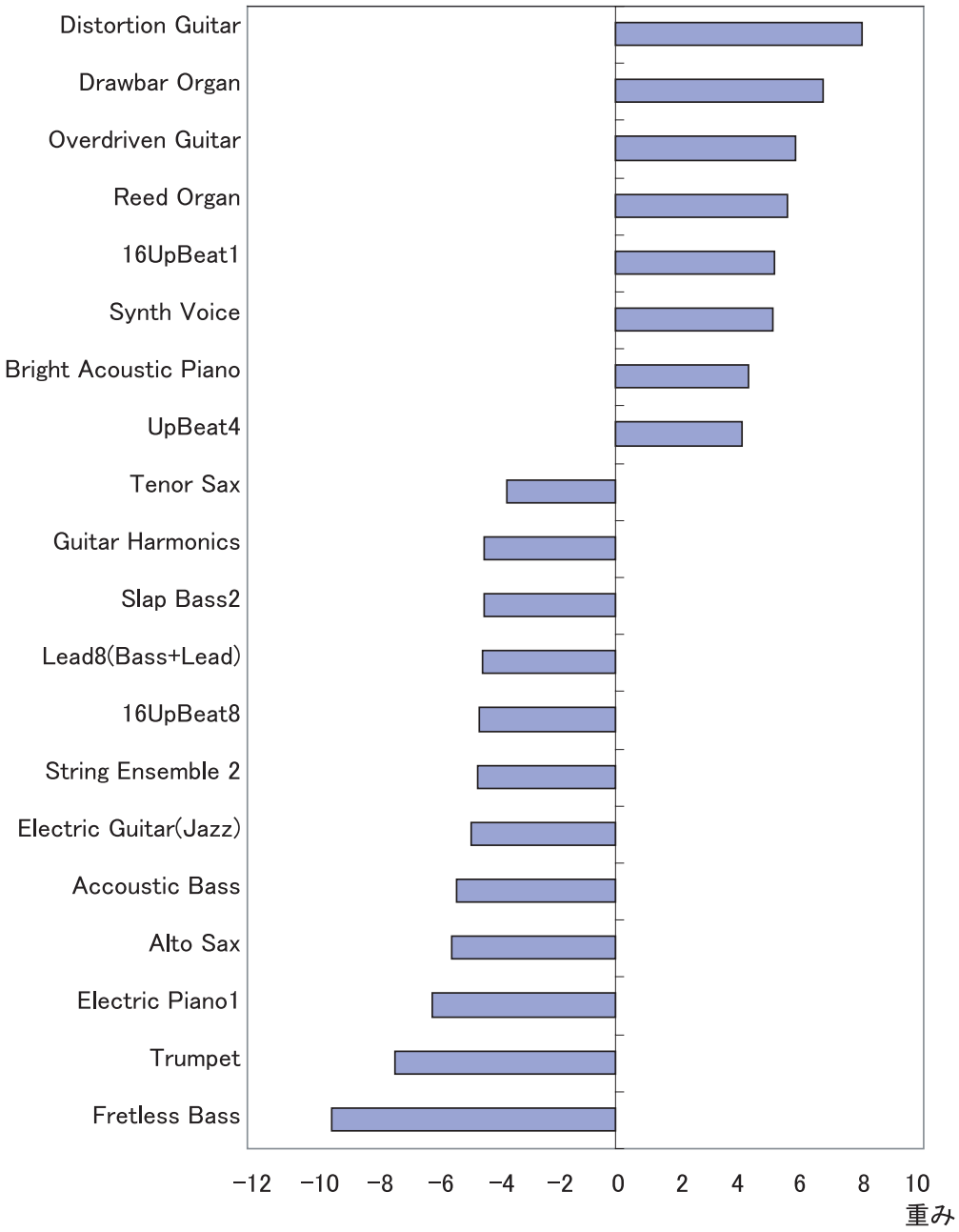


図 4.6: 入力層からノード 1 へのリンクの重みの大きさが大きなもの

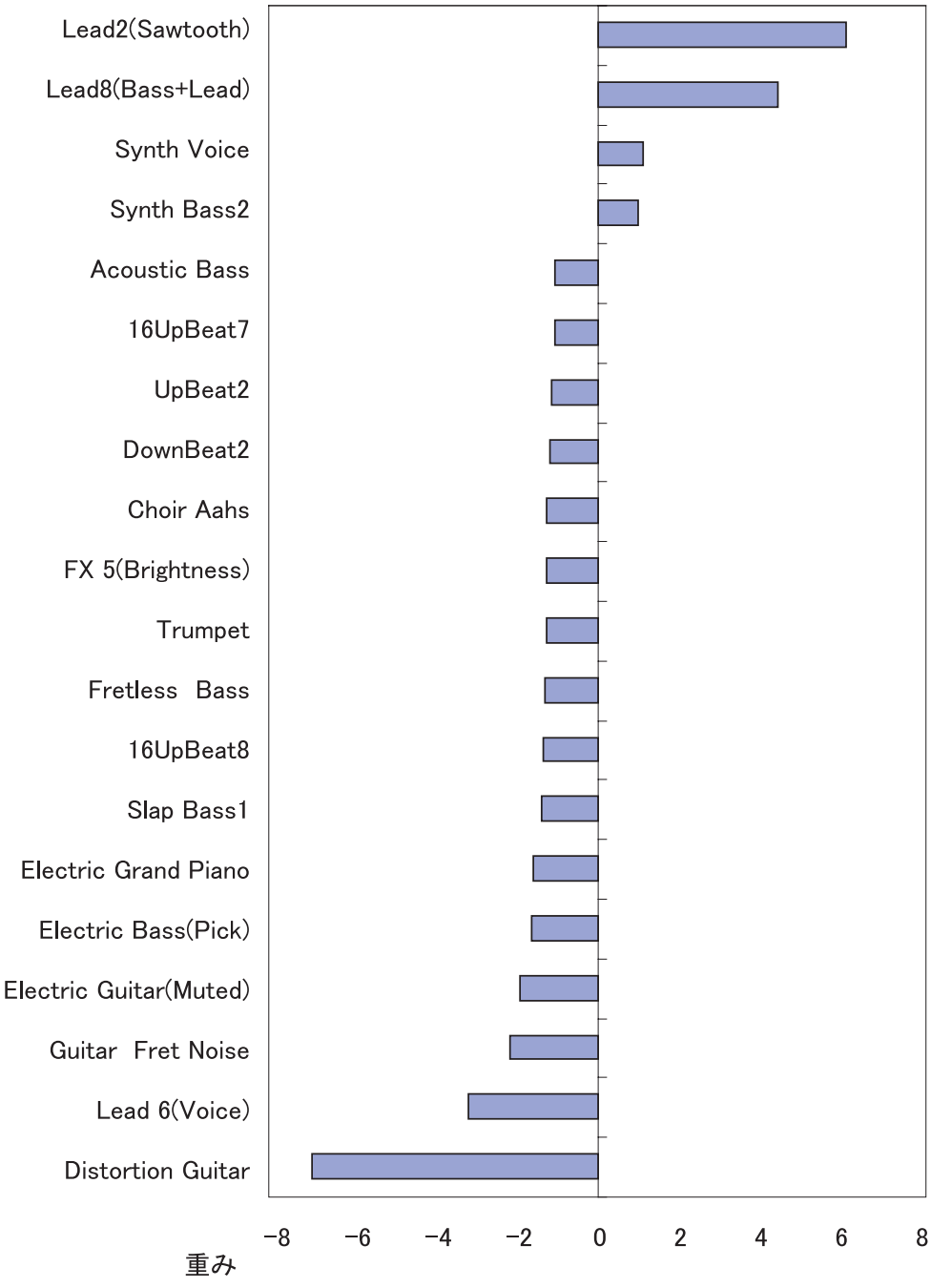


図 4.7: 入力層からノード 2 へのリンクの重みの大きさが大きなもの

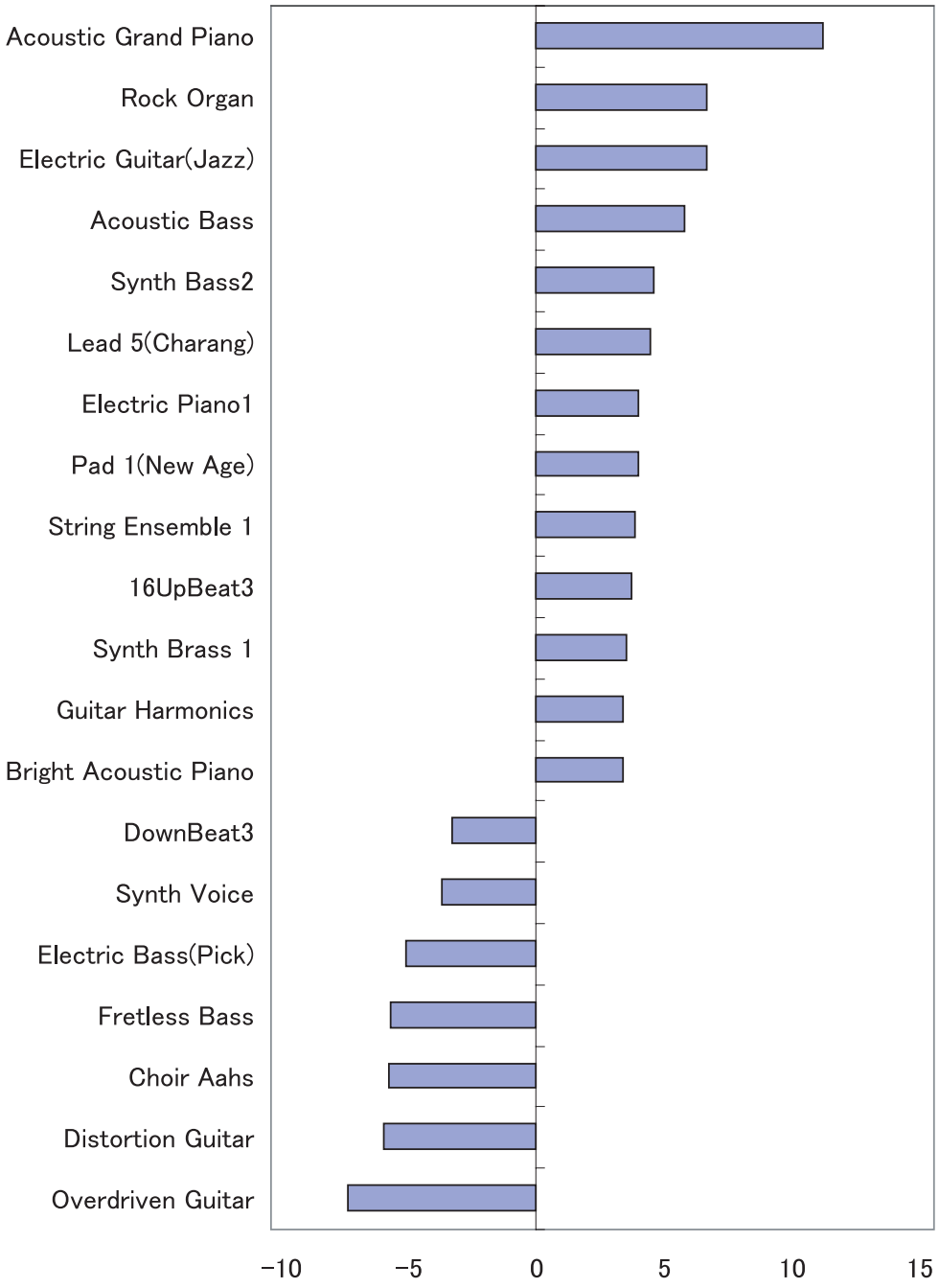


図 4.8: 入力層からノード 3 へのリンクの重みの大きさが大きなもの

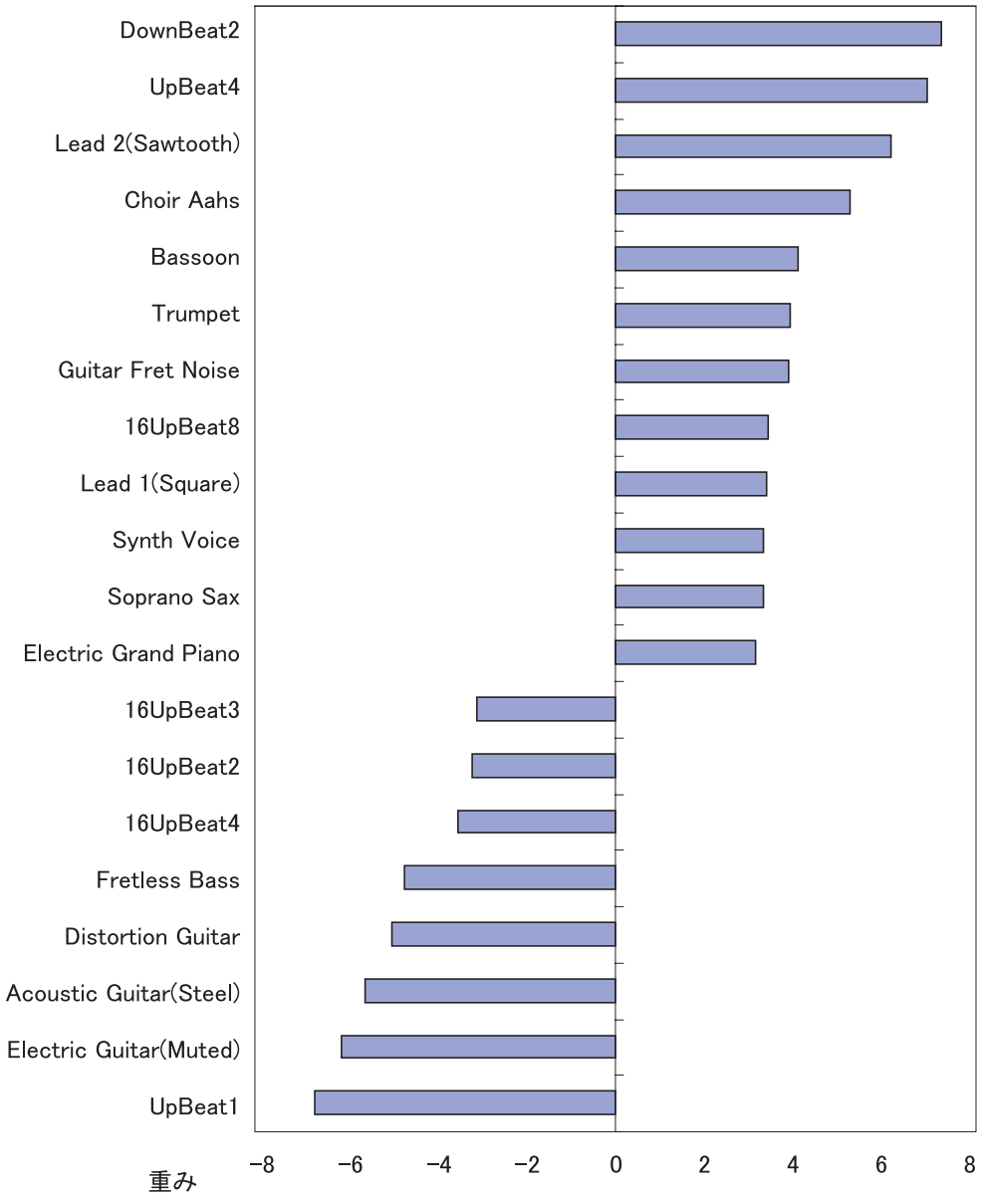


図 4.9: 入力層からノード 4 へのリンクの重みの大きさが大きなもの

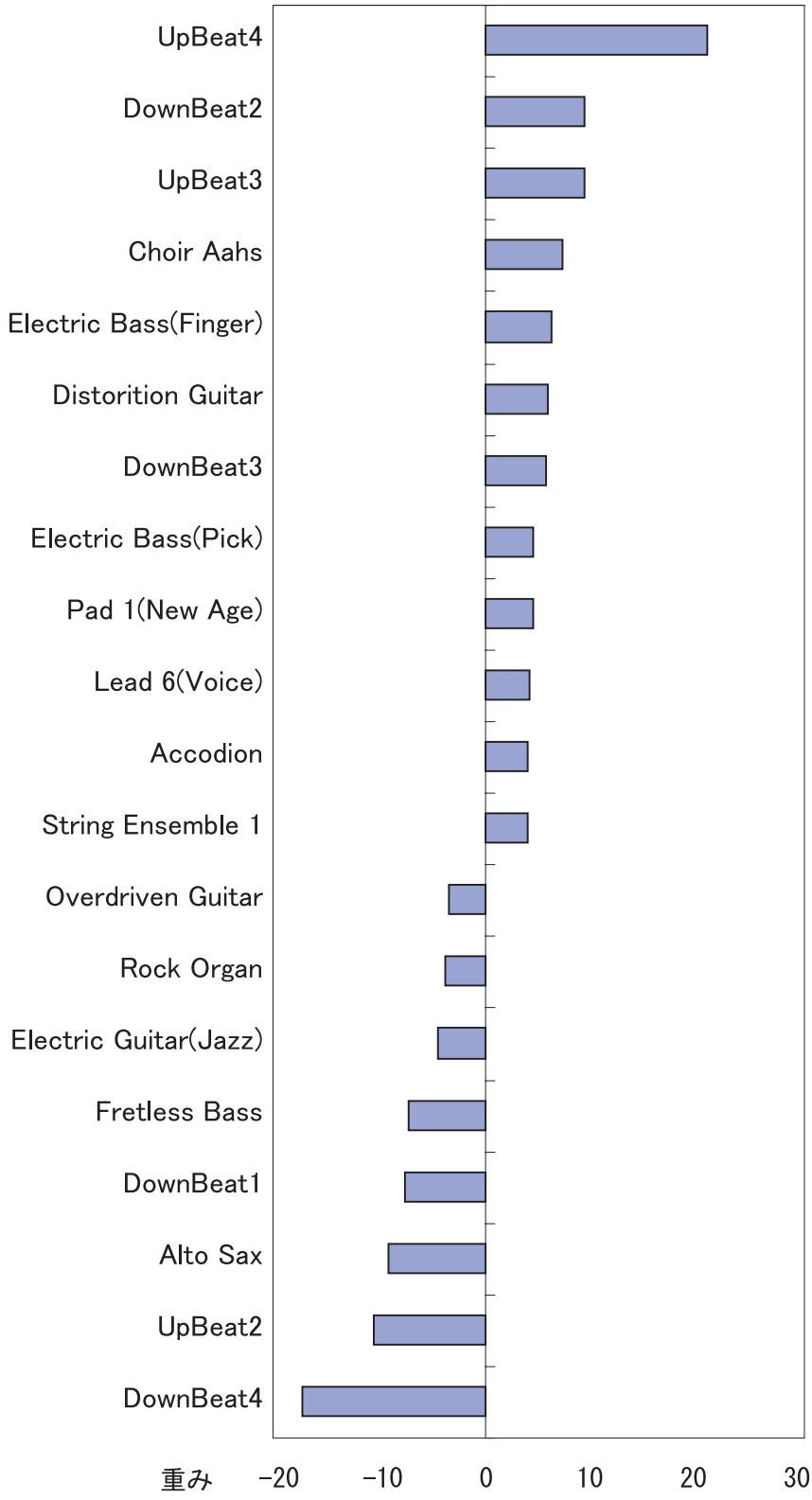


図 4.10: 入力層からノード 5 へのリンクの重みの大きさが大きなもの

5章
打拍フォームに対する教示の効果

5.1　はじめに

本章では、打楽器演奏における打拍フォームの重要性を示す為、及び打拍フォームに対する教示が打楽器未経験者の演奏に及ぼす効果を検証する目的で、打楽器未経験者に対して教示が与える効果を調べる検証実験を行なった。

5.2　打楽器経験者のフォーム教示による未経験者の演奏改善実験

実験は以下の被験者、手続きで行なわれた。

被験者

- （9.4節のフィードバックなしグループに含まれる）打楽器未経験者3名

実験手続き被験者は始めに1分間のリズムパターン3（図5.1）の打拍を2回行なう。その後、（週に8時間以上の練習経験を5年間以上持つ）打楽器（ドラム）経験者によって打拍フォームに関する教示を10分程度与える。そしてその後、同様にリズムパターン3の1分間打拍を8回行なう。尚、システムはいずれもフィードバックがない、演奏と手首の位置情報の測定のみを行なうタイプを用いた。

5.3　分析

前節の手続きによって得られたデータから、9.3節と同様に特徴量を抽出し、9.4節の実験における、統制群での結果と比較を行なった。どちらも2回目〜8回目の打拍データから得られた特徴量について比較を行なった。具体的には

- 今回の実験データにおいて、教示を行なった直後の4回（3〜6回目）と、後半の4回（7〜10回目）の平均値の比較

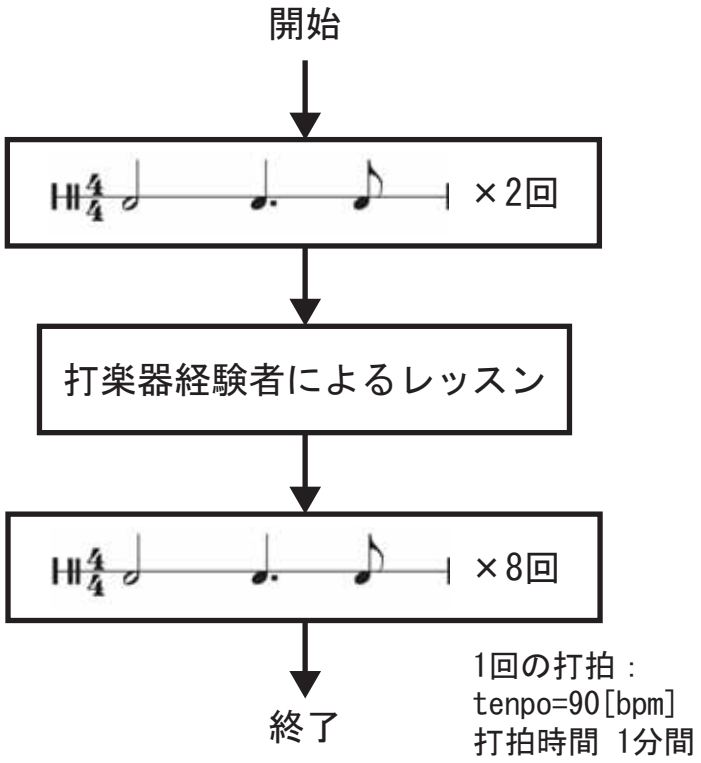


図 5.1: 実験の流れ

- 前回の結果と今回の結果における、各特徴量の平均値の比較

について分析を行なった。

5.3.1　特徴量の変化の検証

抽出した演奏データ及び手首位置情報による打拍フォームデータから得た各特徴量に対して、教示直後の 4 回と後半 4 回の平均値の比較を t 検定で行なった。

その結果、特徴量の中で振幅の不安定性 $SD(S)$ のみ後半 4 回の方が有意に値が小さい事が分かった（$t(22) = 2.306, p < .05$）。この原因は、教示が与えられた事により被験者が振幅がより安定したフォームへ適応していった為と考えられる。

5.3.2　教示前と教示後の各特徴量の平均の比較

教示を与えずにリズムパターン打拍を行なった 8 回（9.4 節）と、教示を与えた後に行なった今回の打拍 8 回に対して、各特徴量の平均を t 検定を用いて比較した。

その結果を図 5.2〜図 5.6 に示す。

- 音長の偏り $\overline{sh}$（$t(46) = 3.700, p < .01$）
- 偏りの不安定性 $SD(sh)$（$t(46) = 1.981, p < .10$）
- ストローク誤差 $\overline{E}$（$t(46) = 2.796, p < .01$）
- 手首の振り幅の不安定性 $SD(S)$（$t(46) = 2.490, p < .05$）
- リズム誤差 $\overline{T}$（$t(46) = 2.302, p < .05$）

の特徴量において、有意な違いを発見できた。

この結果より、打楽器未経験者へフォームに関する教示を与えた事により

- 安定したリズムパターンで演奏できた
- 安定した打拍フォームで演奏できた
- 安定した手首の振り幅で演奏できた

事が分かった。

5.4　5 章のまとめ

本章では、打拍フォームに対する教示が打楽器未経験者の演奏に及ぼす効果を検証する目的で、検証実験を行なった。

打楽器未経験者 3 名に対して、始めに 1 分間のリズムパターン打拍を 2 回行なわせ、その後に打楽器経験者による教示を 5 分間ほど与えた。続いて 1 分間のリズムパターン打拍を 8 回行なわせ、教示の効果を検証した。

その結果、フォームに関する教示を与えた後の 8 回の方が

- 安定したリズムパターンで演奏できた
- 安定した打拍フォームで演奏できた
- 安定した手首の振り幅で演奏できた

事が分かった。

この結果より、打楽器経験者による教示が未経験者の演奏を改善できる可能性のあることが分かった。

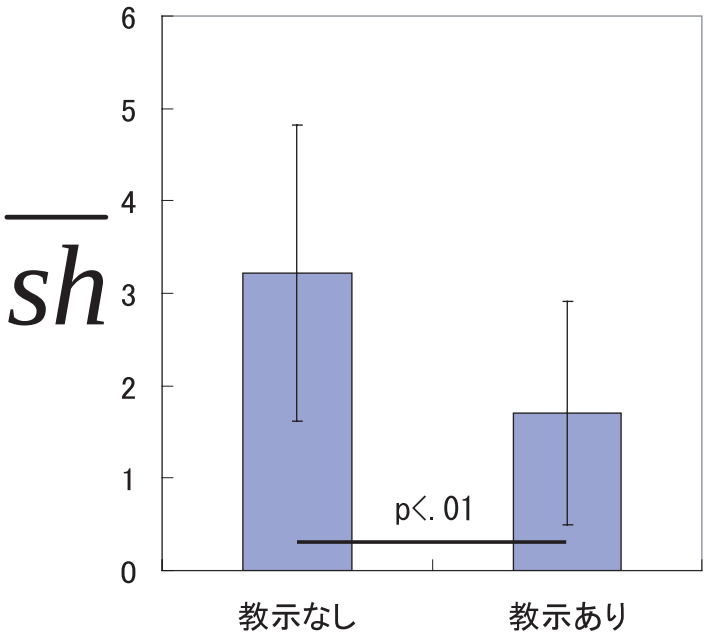


図 5.2: 教示あり、なし群間における音長の偏りの t 検定

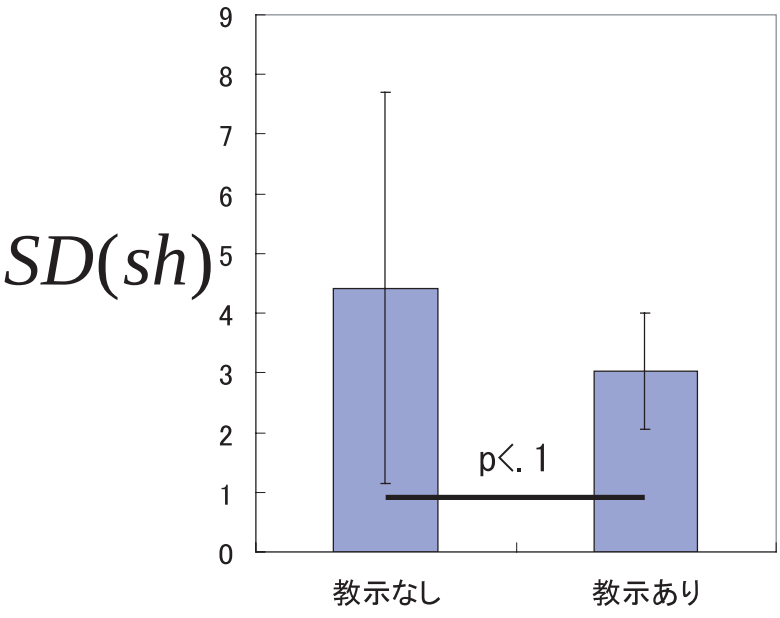


図 5.3: 教示あり、なし群間における音長の偏りの t 検定

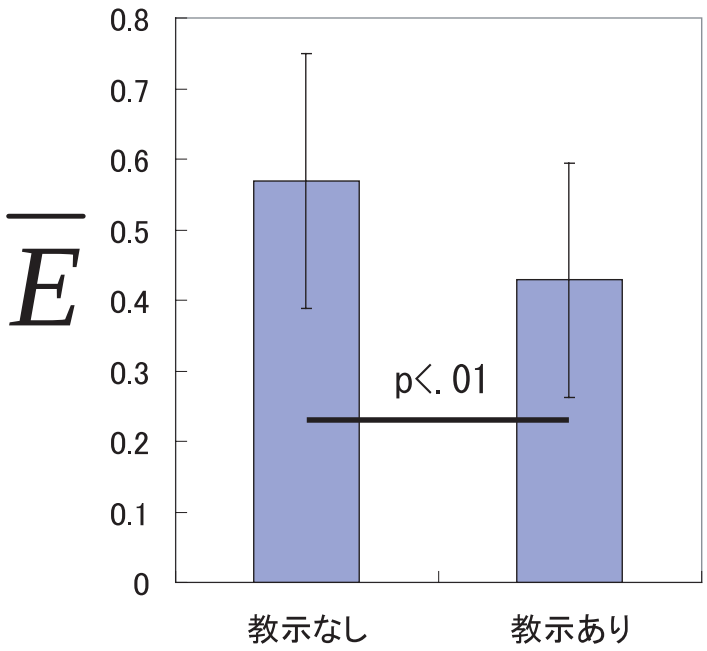


図 5.4: 教示あり、なし群間におけるストローク誤差の t 検定

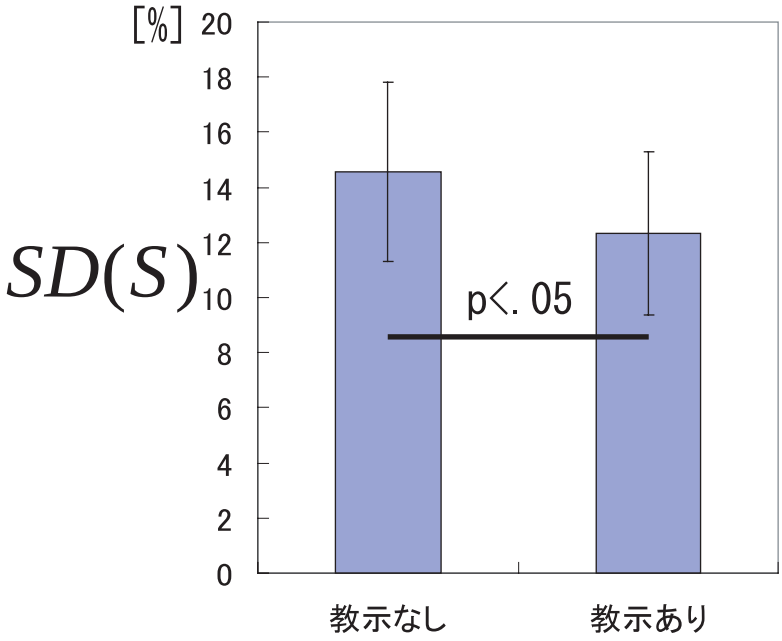


図 5.5: 教示あり、なし群間における振り幅の不安定性の t 検定

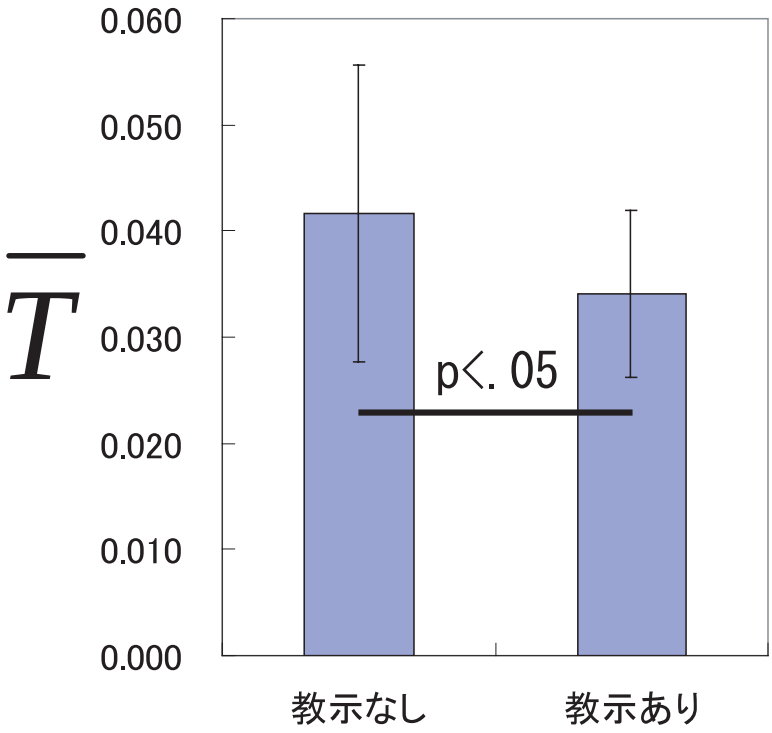


図 5.6: 教示あり、なし群間におけるリズム誤差の t 検定

6章

一定間隔打拍時の右手首位置情報の測定

6.1 はじめに

本章では、打楽器の演奏をフォーム改善の観点から試みるための前提として、打楽器経験者と未経験者のフォームの違いを調べた。

具体的には、簡易モーションキャプチャシステム [54] を用い、打楽器演奏時の腕の位置情報の取得を試みた。そして演奏時の手首の位置情報波形データから 3 種類の特徴量を抽出し、打楽器経験者と未経験者によって演奏の際の右腕のストローク波形に違いがあるのかを統計処理によって分析した。分析には、始めに特徴量ごとに、経験者と未経験者とのグループ間での t 検定を行ない、特徴量ごとの有意差を定量的に求めた。続いて、ステップワイズ法による判別分析を行ない、両グループ間での線形的な判別を行ない、正準判別関数係数を調べ、判別の際の特徴量の組合せについて検討し、両グループにどのような波形が有意に見られるかを検討した。そして、被験者の演奏の分布を可視化する目的で、得られた 3 種類の特徴量からクラスター分析を行ない、演奏の分布を観察した。

6.2 ストロークフォームデータの取得

6.2.1 目的

本実験の目的は、打楽器の経験者と未経験者の打拍時のストロークの違いを定量的に調べる事である。

6.2.2　測定装置と実験課題

図 6.1 に、本実験で用いたフォームデータの測定装置を示す。CCD カメラが被験者の右手の甲に付いているマーカの 3 次元位置情報を取得し、打拍している間の位置情報がコンピュータへ常に送られる。尚、[54] のモーションキャプチャシステムは、15sps(samples per second) のサンプリングレートを持つ 4 つのマーカで構成されているが、本研究ではそれを改造し、60sps のサンプリングレートのマーカを一つ用いている。

被験者は、一定の基準テンポで、スティックを用いてドラムパッドに等間隔に打拍する課題を行なわせ、その際の右手の甲に付けたマーカの位置情報を時系列波形として測定した。

本論文では基準テンポとして、90BPM（Beats Per Minute）、120BPM、180BPM の 3 種類のテンポを用いた。測定手順は以下である。まず基準テンポで等間隔なメトロノーム音を被験者に提示し、基準テンポを覚えさせた。その後、120 秒間、提示されたテンポの記憶を頼りに等間隔、等音量でスティックで打拍させた。この測定より、各被験者の各試行につきそれぞれ 120 秒間の打拍ストローク時の右手の甲の 3 次元位置情報を得た。それを各テンポに対して 3 回ずつ、つまり被験者一人あたり 9 回ずつ試行した。そして各テンポにおいて最後の試行の後に、自分の演奏に対する自己評価アンケートに回答させた。さらに全試行が終った後、楽器演奏経験、持っている CD の枚数、好きなジャンルなどの音楽経験アンケートに回答させた。被験者は、一般大学生及び大学院生の、1 週間に 8 時間で 5 年間以上の練習経験を持つ打楽器（ドラム）経験者 4 名、未経験者 10 名である。

6.2.3　測定データ

測定したストローク波形データは、

$$\vec{f}(t), t = 0, \Delta t, 2\Delta t, ..., 120\Delta t\,\mathrm{sec}, \Delta t = \frac{1}{60}\,\mathrm{sec}$$

で表される 3 次元座標ベクトルデータである。図 6.2、図 6.3 に、取得した波形データの例の中で、顕著な例を示す。縦軸はマーカの垂直座標、横軸は時間である。測定した波形は、大よそ (a)〜(d)4 種類の特徴的な波形が観察された。(a) は周期不安定型で、周期ごとの大きさが一定しない波形である。(b) は正弦波型で、左右対称で正弦波に比較的近い波形である。(c) はノコギリ波型で、左右非対称なノコギリ波に近い波形である。(d) は大振幅型で、振幅が大きい波形である。

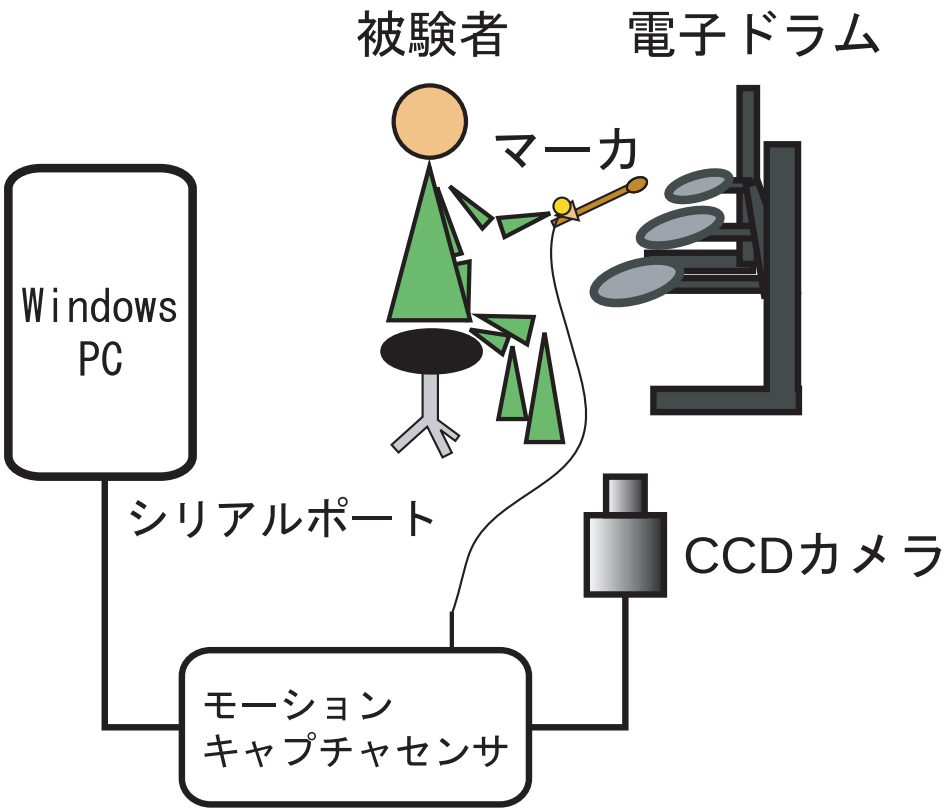
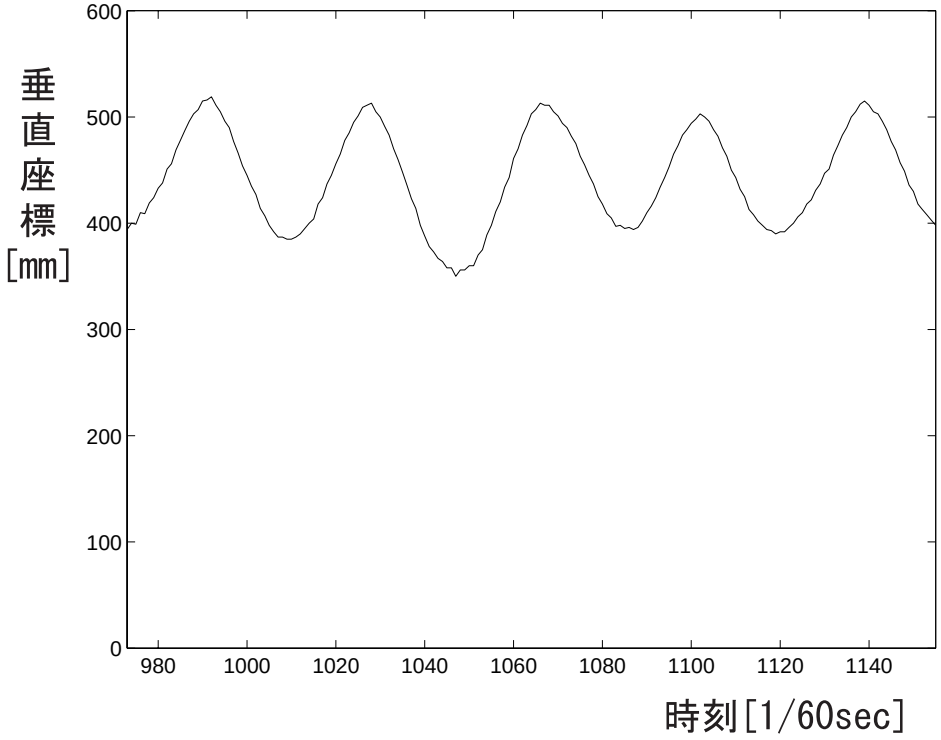
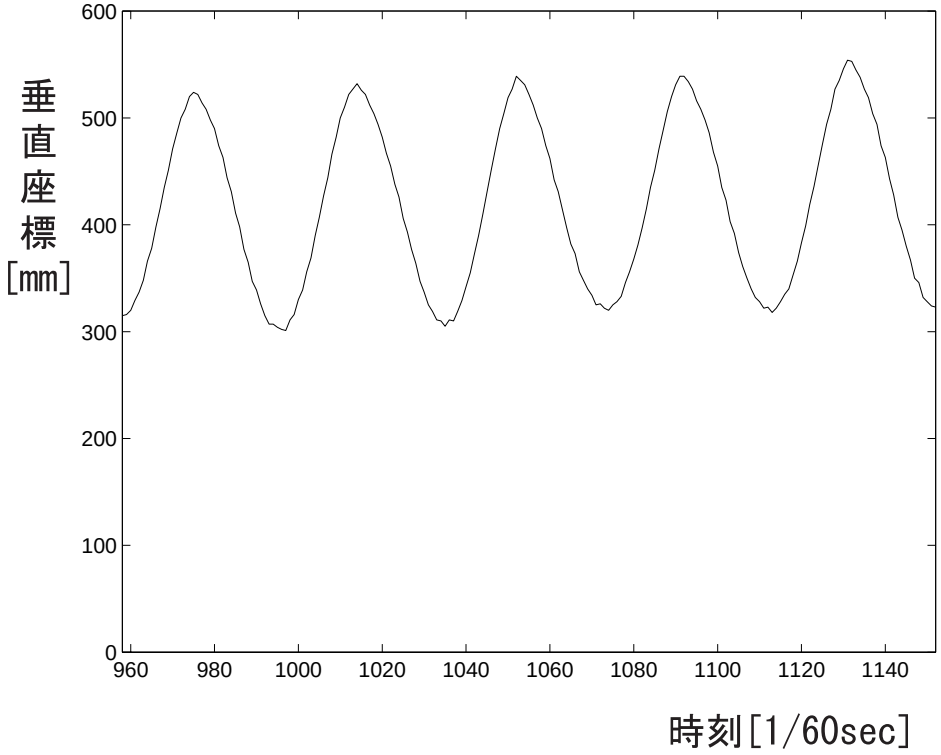


図 6.1: 実験環境

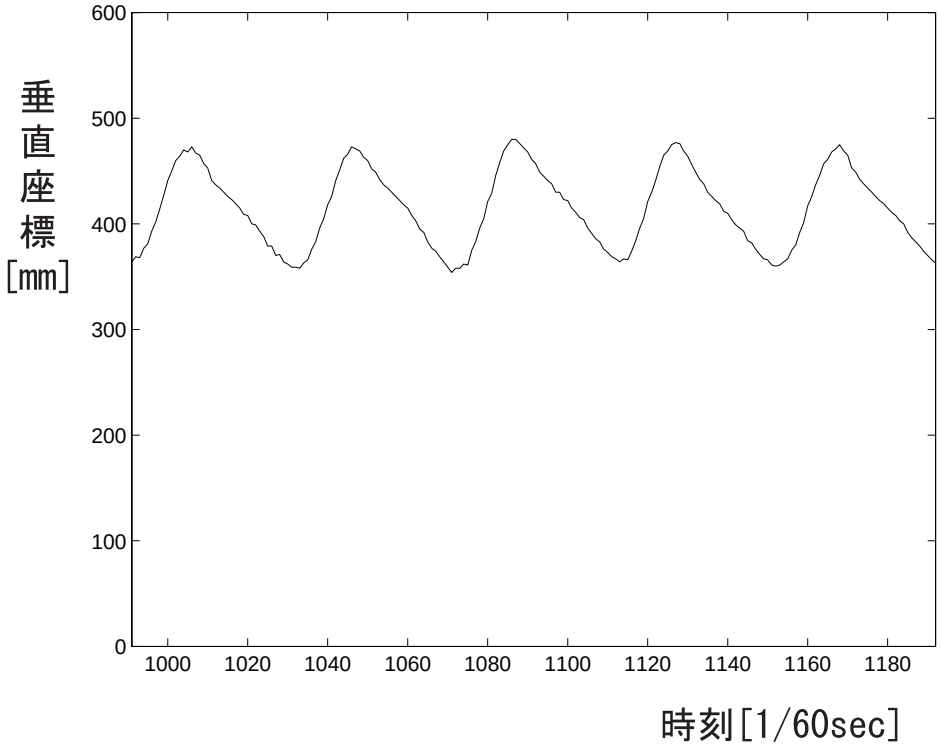


(a) 周期不安定型

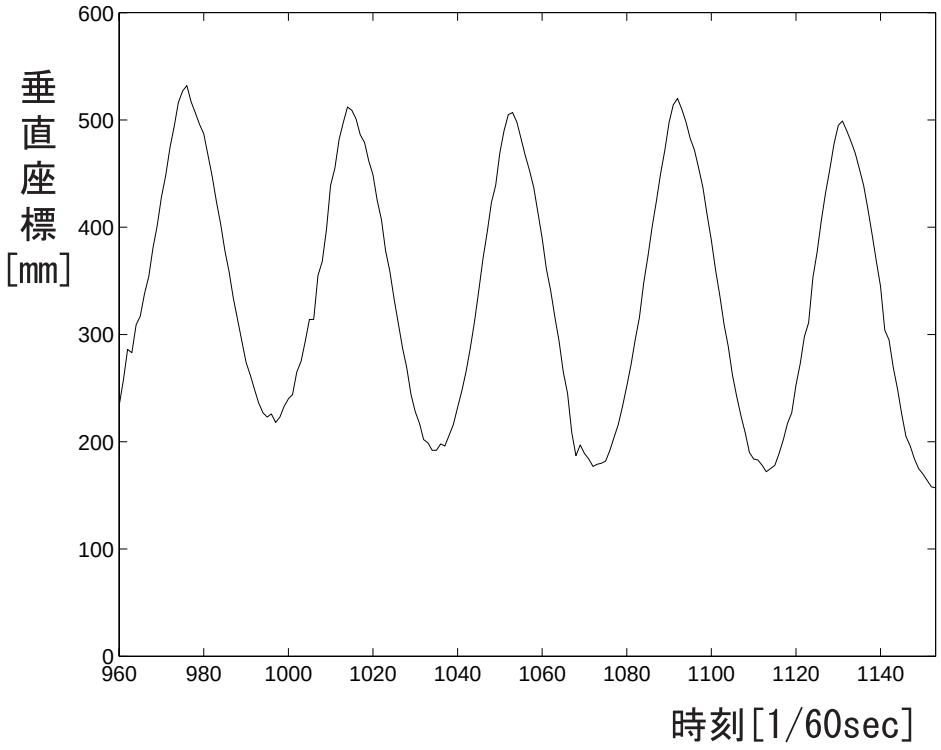


(b) 正弦波型

図 6.2: ストローク波形の例-1



(c) ノコギリ波型



(d) 大振幅型

図 6.3: ストローク波形の例-2

6.2.4　測定データからの支援の考察

前節の 4 種類の波形に対して学習支援を考える。システムが自動的に (a) の波形を認識することができれば、「ストロークフォームが一定していません。」等のフィードバックをユーザへ返すことができる。また、システムが自動的に (b)、(c) の波形を認識できれば、打楽器経験者のストローク波形を予め調べることにより、フィードバックが可能となる。同様に、システムが自動的に (d) の波形を認識できれば、つまり振幅を計算できれば、打楽器経験者のストローク波形の振幅に合うようなフィードバックをユーザへ提供することが可能となる。さらに、今回の実験では打拍時の音の強さは一定という条件だけであったが、「mf（メゾフォルテ）で叩く」などの音の強さの条件を付与した実験環境においては、振幅値がより重要なパラメータとなることが予想できる。

従って、測定データから学習支援を行なうには、上記のような波形を判別できること、もしくは定量的にその特徴を数値化することが必要となる。

そのために本論文では、

- 周期ごとのストロークフォームのずれ（誤差）
- 波の形
- 周期ごとの振幅の大きさ

の 3 種類の要素から特徴量を取得することを考えた。

6.3　特徴量の取得

6.3.1　打拍点の推定

前節において、周期ごとに値を計算するためには、周期の境目を特定しなければならない。しかし、測定したデータは被験者の手首の位置情報のみで、演奏情報は含まれていない。従って、ストローク 1 周期ごとに処理を行ないたい場合には、何らかのアルゴリズムで打拍点を推定し、決定する必要がある。

本論文では、打拍点を以下の方法で決定した。

打拍点推定アルゴリズム

1. $n-1$ 周期目の打拍点時刻 t_{n-1} に対し、

$$\begin{aligned} & t_{n-1} + (1-\alpha) \times \frac{60}{\text{tempo}} \\ \leq \ & t[\text{sec}] \\ \leq \ & t_{n-1} + (1+\alpha) \times \frac{60}{\text{tempo}} \end{aligned}$$

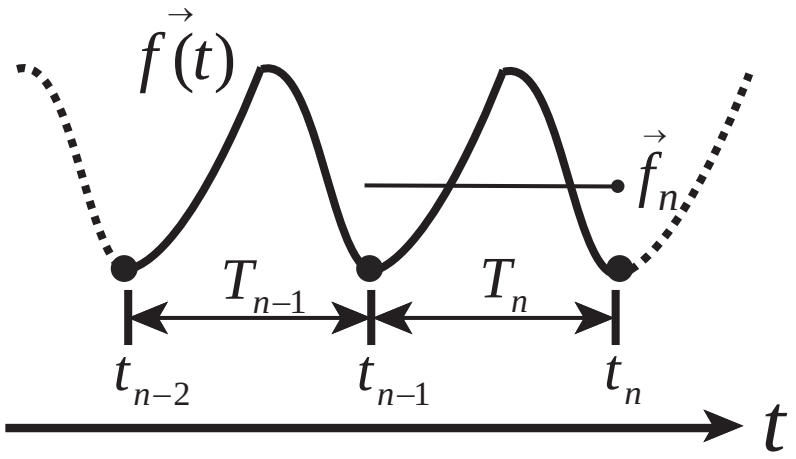


図 6.4: ストローク誤差の取得

$$(\text{tempo} = 90, 120, 180[\text{bpm}]\ \alpha = 0.4)$$

の時間範囲を、n 周期目の打拍点 t_n の候補とする。

2. 時間範囲 t の中で、最も垂直座標の小さな値を t_n と決定する。

6.3.2　ストローク誤差

本論文では、打拍の再現性を一評価指標とする。周期的な打拍（図 6.4）における繰り返しのストローク誤差 E_n を次式で定義する。

$$E_n = \frac{\frac{1}{D_n}\int_{t_{n-1}}^{t_n}\left|\vec{f}(t)-\vec{f}(t')\right|^2 dt}{\frac{1}{D_n}\int_{t_{n-1}}^{t_n}\left|\vec{f}(t)-\vec{f_n}\right|^2 dt}$$

ここで $\vec{f}(t)$ は複数マーカの座標を並べたベクトル、D_n は n 番目の打拍の周期で $D_n = t_n - t_{n-1}$、$\vec{f_n}$ は n 番目のフォームの平均値、つまり、$\vec{f_n} = \frac{1}{D_n}\int_{t_{n-1}}^{t_n}\vec{f}(t)dt$ である。

また、$t' = t_{n-2}\frac{t_n - t}{D_n} + t_{n-1}\frac{t - t_{n-1}}{D_n}$ であり、t_{n-2}、t_{n-1}、t_n は連続する打拍の時刻とする。このストローク誤差 E_n を求めることにより、周期ごとのストロークがどのくらいずれたかが定量的に分かる事が期待できる。特徴量として、被験者の一回の演奏におけるストローク誤差の平均値 $\overline{E}$ 及び標準偏差 $SD(E)$ を統計的分析に用いた。

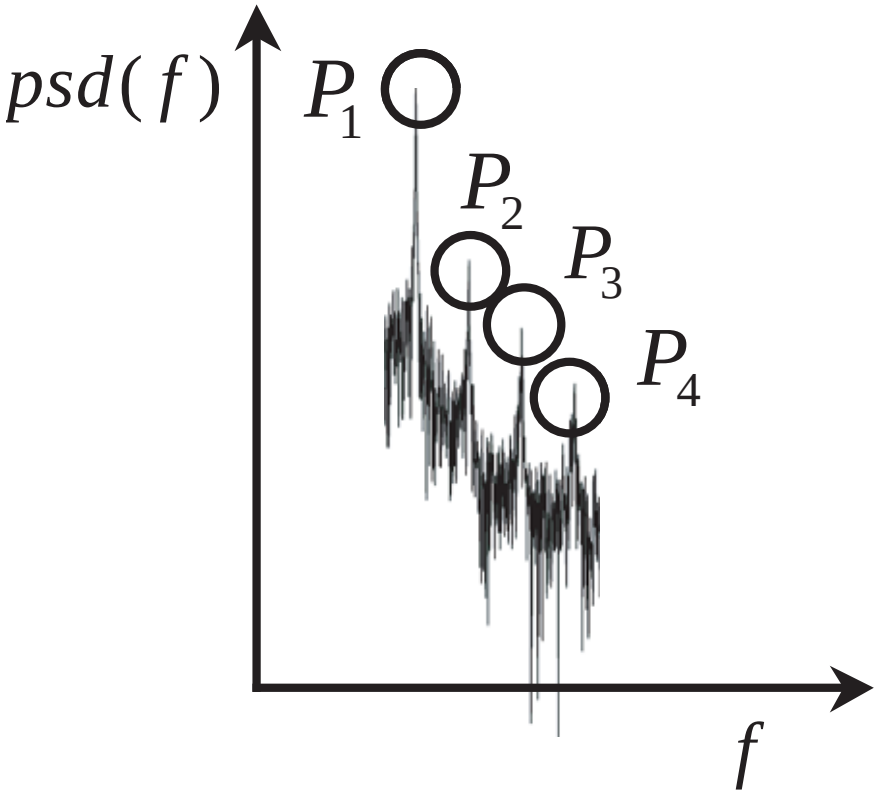


図 6.5: パワースペクトルピーク値の測定法

6.3.3　パワースペクトルピーク比

測定データから、ストロークにおける高周波数成分を含めた特徴量を抽出するために、パワースペクトルのピーク値を求めた。測定した 3 次元位置情報の垂直成分波形に対してパワースペクトル密度関数 $psd(f)$ を求め、そのピーク値 $P_1, P_2, ..., P_{10}$ を求めた（図 6.5）。さらに、その各ピーク値を 1[Hz] の一定幅で積分値 $\Delta P_1, \Delta P_2, ..., \Delta P_{10}$ を求めた。$\Delta P_n (n = 1, 2, ..., 10))$ は以下の式で定義できる。

$$\Delta P_n = \int_{f_n - 0.5[Hz]}^{f_n + 0.5[Hz]} psd(f) df$$

その比 $\frac{\Delta P_2}{\Delta P_1}, \frac{\Delta P_3}{\Delta P_1}, ..., \frac{\Delta P_{10}}{\Delta P_1}$ の 9 つの値を特徴量として後の統計的分析に用いた。

この値を求めることにより、2 章 4 節で述べた波の形を特徴量として取得できると考えられる。

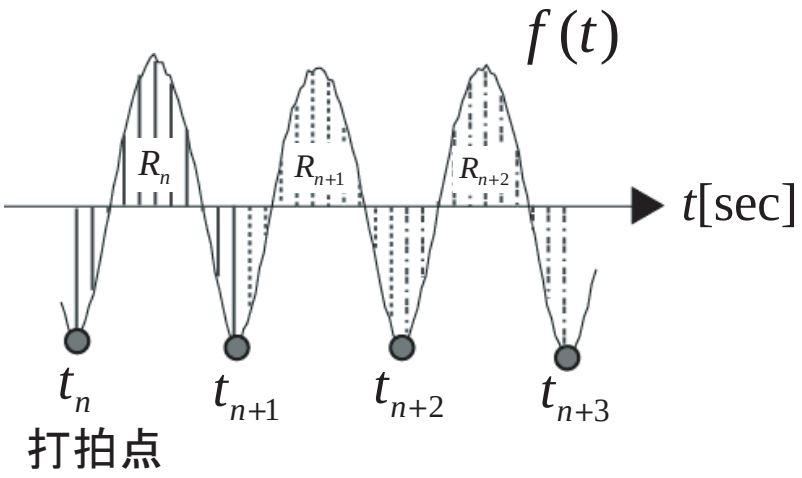


図 6.6: 振幅 RMS 値の測定法

6.3.4　振幅 RMS 値

振幅の RMS（Root Mean Square：2 乗平均）値とは、測定した 3 次元位置情報の垂直成分波形に対して、遮断周波数が 0.1[Hz] の HPF(High Pass Filter) を通し、周期ごとに振幅の 2 乗平均値を求めたものである。ストローク波形における、n 周期目の振幅の RMS 値 R_n は、以下の式で定義できる。

$$R_n = \sqrt{\frac{1}{T_n}\int_{t_{n-1}}^{t_n} f_y^2(t)dt},\ n = 1, 2, ...$$

振幅の RMS 値を求めることにより、被験者のストロークの大きさが求められる。特徴量として、RMS 値の平均値 $\overline{R}$、標準偏差 $SD(R)$ を統計的分析に用いた。

6.4　t 検定による分析

測定した被験者のフォームデータから得られた各特徴量 $\frac{\Delta P_2}{\Delta P_1}, \frac{\Delta P_3}{\Delta P_1}, ..., \frac{\Delta P_{10}}{\Delta P_1}, \overline{E}, SD(E), \overline{R}, SD(R)$ に対して、経験者、未経験者のグループ間における t 検定を行なった。t 検定は 3 種類のテンポごとに別々に行なった。その理由は、テンポが異なると演奏におけるストロークの動作も異なると仮定しているからである。

6.4.1　90bpm における t 検定

被験者合計 14 名の 90bpm のテンポによる演奏フォーム波形データより抽出した、 合計 40 サンプルの特徴量において、経験者、未経験者のグループ間におけ

る t 検定を行なった。その結果、$\frac{\Delta P_2}{\Delta P_1}$ の特徴量において、経験者と未経験者で有意な差が見られた（$t(38) = -3.352, p < 0.01$）。これにより、打楽器経験者の方が未経験者よりも基本周波数のピーク値と比較して、基本周波数に対して 2 倍の高調波の割合が有意に大きくなっていることが分かる。

また、$\overline{R}, SD(R)$ の特徴量においても、経験者と未経験者では有意差が見られた（$\overline{R}$：$t(38) = 3.649, p < 0.01$、$SD(R)$：$t(38) = 3.447, p < 0.01$）。これより、打楽器経験者は未経験者に比べて、リズムキープすることを最重要目的に打拍した場合、有意にストロークの振り幅が小さく、また振り幅の分布が狭いことが分かった。顕著な例として未経験者のストローク波形（図 6.7）と経験者の波形（図 6.8）を比較すると、図 6.7 の方は観察すると波形が左右対称で、sin 波に近い波形であるのに対し、図 6.8 の方は若干、左右非対称な形で 2 倍高調波を多く含むノコギリ波に近い波形であることが分かる。

6.4.2　120bpm における t 検定

前節と同様に、被験者計 14 名の 120bpm のテンポによる演奏フォーム波形より抽出した計 40 サンプルの特徴量において、経験者、未経験者のグループ間における t 検定を行なった。その結果、$\frac{\Delta P_3}{\Delta P_1}$ の特徴量において、経験者と未経験者で有意な差が見られた（$t(38) = 2.112, p < 0.05$）。これより、打楽器経験者の方が未経験者よりも基本周波数に対して 3 倍の高調波の割合が有意に小さくなっていることが分かる。さらに、$\overline{E}, SD(E), \overline{R}, SD(R)$ の特徴量においても、経験者と未経験者では有意差が見られた（$\overline{E}$：$t(38) = 2.749, p < 0.01$、$SD(E)$：$t(38) = 2.029, p < 0.10$、$\overline{R}$：$t(38) = 3.182, p < 0.01$、$SD(R)$：$t(38) = 4.306, p < 0.01$）。これより、120bpm の演奏において打楽器経験者は、未経験者に比べて有意にストローク誤差及びその標準偏差が小さく、また有意にストロークの振り幅が小さく、その分布も狭いことが分かった。従って、打楽器経験者の方がコンパクトで、正確なストロークフォームであったことが言える。顕著な例として未経験者のストローク波形図 6.9 と経験者の波形図 6.10 を比較すると、図 6.9 の方は観察すると波形が左右対称で、3 倍高調波を多く含む三角波に近い波形であるのに対し、図 6.10 の方は若干、左右非対称な形でノコギリ波に近い波形であることが分かる。

6.4.3　180bpm における t 検定

さらに 前節と同様に、被験者計 14 名の 180bpm のテンポによる演奏フォーム波形より抽出した計 42 サンプルの特徴量において、経験者、未経験者のグループ間に

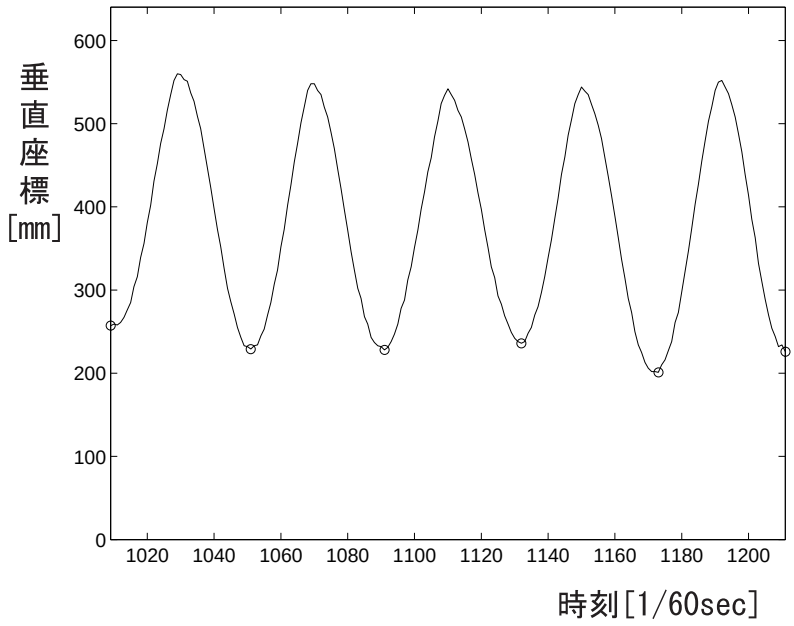


図 6.7: 90bpm における打楽器未経験者のストローク波形の例

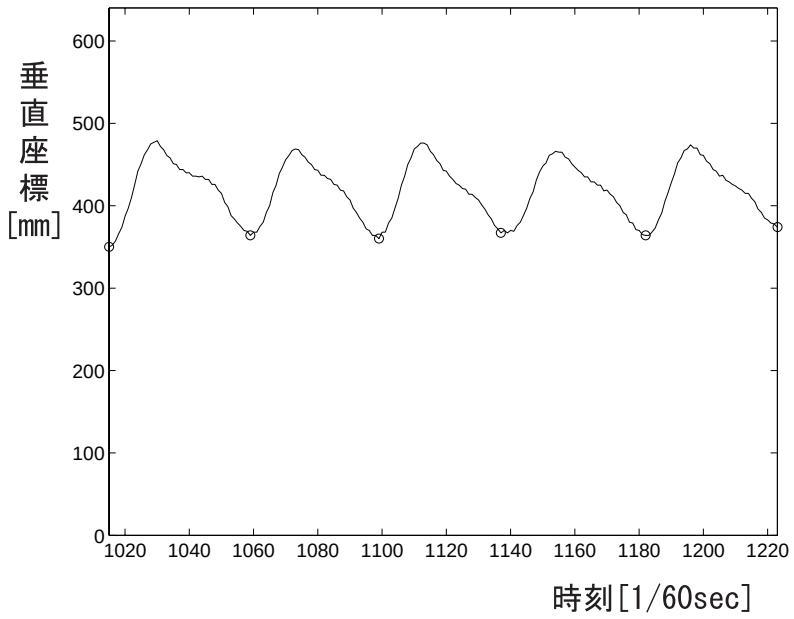


図 6.8: 90bpm における打楽器経験者のストローク波形の例

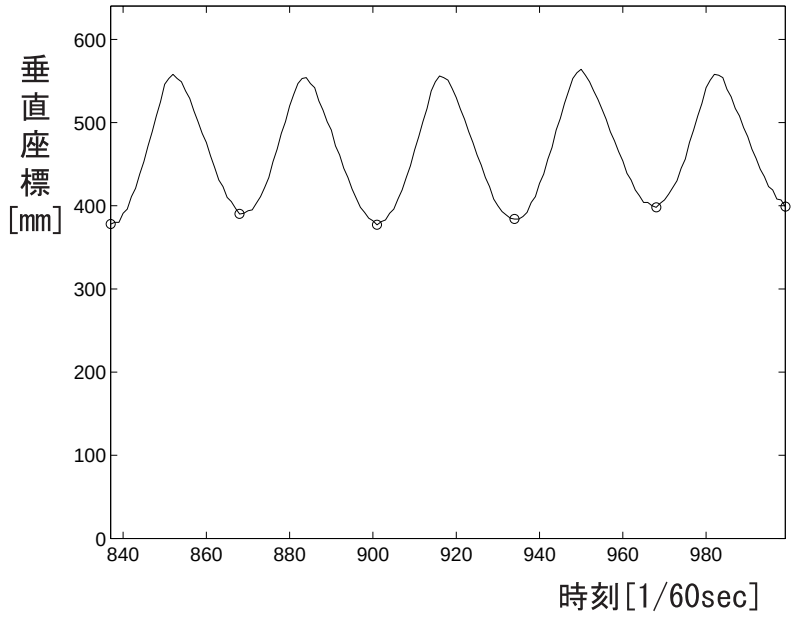


図 6.9: 120bpm における打楽器未経験者のストローク波形の例

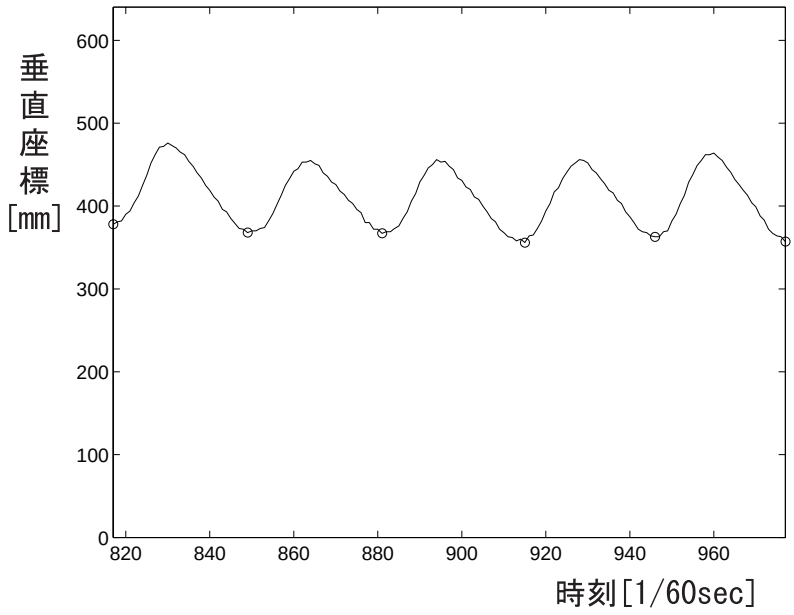


図 6.10: 120bpm における打楽器経験者のストローク波形の例

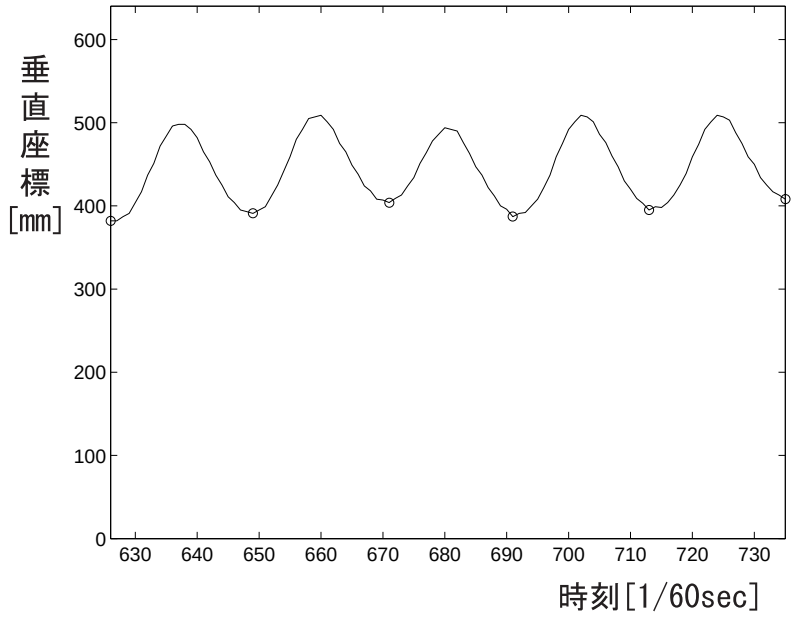


図 6.11: 180bpm における打楽器未経験者のストローク波形の例

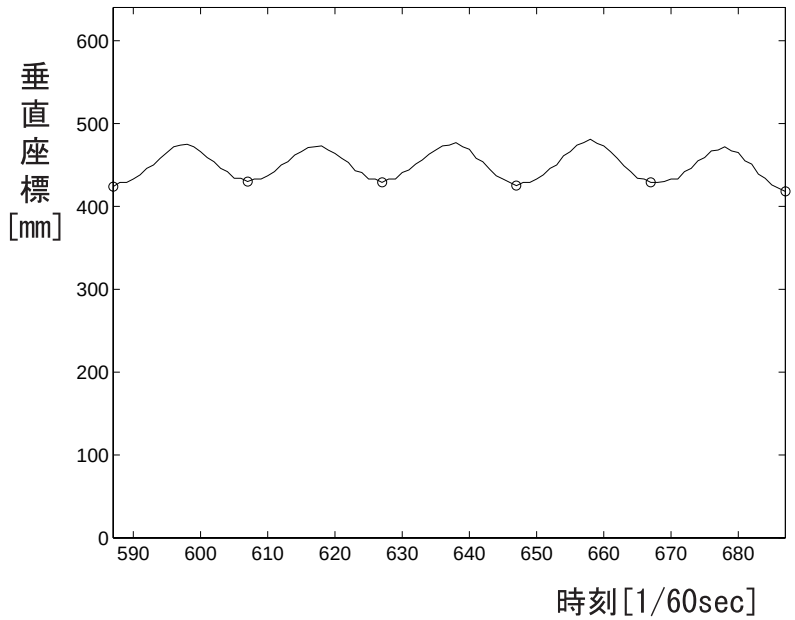


図 6.12: 180bpm における打楽器経験者のストローク波形の例

表 6.1: 判別率集計表 - 90bpm

		予測グループ		
※単位はすべて [人]		未経験者	経験者	合計
元のデータ	未経験者	28	2	30
	経験者	2	8	10

90.0% が正しく分類

表 6.2: 正準判別関数係数 - 90bpm

特徴量	係数
$\frac{\Delta P_2}{\Delta P_1}$	-2.488
$\frac{\Delta P_4}{\Delta P_1}$	2.950
$\overline{R}$	0.830

おける t 検定を行なった。その結果、フォーム波形データより得られたパワースペクトル密度関数から抽出した特徴量に関して、多少偏りはあったものの、有意な差は見当たらなかった。しかし、$\overline{R}, SD(R)$ の特徴量においては経験者と未経験者で有意な差が見られた（$\overline{R}$:$t(40) = 4.039, p < 0.001$、$SD(R)$:$t(40) = 3.537, p < 0.01$）。これより、180bpm の演奏において打楽器経験者は、有意にストロークの振り幅が小さく、その分布も狭いことが分かった。従って、打楽器経験者の方がコンパクトなストロークフォームで演奏していたことが言える（図 6.11、図 6.12）。

6.5　判別分析による解析

6.5.1　90bpm における判別分析

前節の t 検定の結果を基に、打楽器経験者か未経験者かを目的変数、特徴量を独立変数とした判別分析を行なった。分析にはステップワイズ法を用い、2 つのグループ間のマハラノビスの距離が最大となるような独立変数を順次、選定し判別させた。その結果、90bpm のテンポにおいて、必要な特徴量は $\frac{\Delta P_2}{\Delta P_1}, \frac{\Delta P_4}{\Delta P_1}, \overline{R}$ の 3 種類の特徴量が選定され、90% の判別率が得られた（表 6.1、表 6.2）。この結果より、90bpm のテンポにおいては、経験者か未経験者かを分類する主な要因は $\frac{\Delta P_2}{\Delta P_1}, \frac{\Delta P_4}{\Delta P_1}, \overline{R}$ の特徴量であり、これを用いることにより大よその分類が可能であることが分かった。

表 6.3: 判別率集計表 - 120bpm

※単位はすべて [人]		予測グループ 未経験者	予測グループ 経験者	合計
元のデータ	未経験者	29	1	30
	経験者	1	9	10

95.0% が正しく分類

表 6.4: 正準判別関数係数 - 120bpm

特徴量	係数
$\frac{\Delta P_2}{\Delta P_1}$	-2.389
$\frac{\Delta P_3}{\Delta P_1}$	4.170
$\frac{\Delta P_4}{\Delta P_1}$	-2.516
$\frac{\Delta P_5}{\Delta P_1}$	1.170
$SD(R)$	0.916

6.5.2　120bpm における判別分析

同様に、120bpm テンポにおいても判別分析をステップワイズ法で行なった。その結果、120bpm のテンポにおいて、必要な特徴量は $\frac{\Delta P_2}{\Delta P_1}$,$\frac{\Delta P_3}{\Delta P_1}$,$\frac{\Delta P_4}{\Delta P_1}$,$\frac{\Delta P_5}{\Delta P_1}$,$SD(R)$ の 4 種類の特徴量が選定され、95.0% の判別率が得られた（表 6.3、表 6.4)。この結果より、120bpm のテンポにおいては、経験者か未経験者かを分類する主な要因は $\frac{\Delta P_2}{\Delta P_1}$,$\frac{\Delta P_3}{\Delta P_1}$,$\frac{\Delta P_4}{\Delta P_1}$,$\frac{\Delta P_5}{\Delta P_1}$,$SD(R)$ の特徴量であり、これを用いることにより大よその分類が可能であることが分かった。また表 6.4 を見ると、この特徴量に関して、$\frac{\Delta P_2}{\Delta P_1}$ や $\frac{\Delta P_4}{\Delta P_1}$ などの基本周波数の偶数倍成分のピーク値が大きいほど経験者側、$\frac{\Delta P_3}{\Delta P_1}$ や $\frac{\Delta P_5}{\Delta P_1}$ などの基本周波数の奇数倍成分のピーク値が大きいほど未経験者側に判別される結果になった。これより、打楽器経験者のストロークフォーム波形はノコギリ波などの左右非対称な波形である傾向があり、それに対して未経験者のストロークフォーム波形は、三角波などの左右対称な傾向があると言える。また、ストロークの振幅の標準偏差 $SD(R)$ も、打楽器未経験者の方が、演奏全体を通して、広い分布を持つ傾向があることが分かった。

表 6.5: 判別率集計表 - 180bpm

		予測グループ		
※単位はすべて [人]		未経験者	経験者	合計
元のデータ	未経験者	24	6	30
	経験者	3	9	12

78.6% が正しく分類

表 6.6: 正準判別関数係数 - 180bpm

特徴量	係数
$\frac{\Delta P_3}{\Delta P_1}$	0.947
$\overline{R}$	1.131

6.5.3　180bpm における判別分析

180bpm テンポにおいても判別分析をステップワイズ法で行なった。その結果、180bpm のテンポにおいて、必要な特徴量は $\frac{\Delta P_3}{\Delta P_1}$,$\overline{R}$ の 2 種類の特徴量が選定され、78.6% の判別率が得られた（表 6.5、表 6.6)。

この判別率が他の 2 種類のテンポよりも小さい原因としては、テンポが速いため、被験者のストロークフォームの自由度が少なくなり、経験者と未経験者の違いが現れにくくなったためと推察できる。

6.6　クラスター分析による解析

今回の実験によるストローク波形がどのようなグループを構成しているのかを観察するために、テンポ別にクラスター分析を行なった。図 6.13 は 90bpm の演奏の特徴量におけるクラスター分析の結果（デンドログラム）である。クラスター化には Ward 法、クラスター間の距離には平方ユークリッド距離を用いた。この結果を見ると、大まかに 5 種類のクラスターに分類できる。このクラスターを、この演奏における散布図で表現したのが図 6.14 である。また、経験者、未経験者のどちらかであるかを目的変数とした判別分析の結果の因子得点を y 軸に取った散布図が図 6.15 である。これを見ると、クラスター 1 やクラスター 5 のように、経験者の演奏が存在しないクラスターも存在する。また、大よそ演奏者ごとにクラスターが分類されている。これより、クラスター分析を用いることによって、打

楽器経験者及び未経験者という分類だけでなく、演奏スタイル、もしくは演奏者の分類への応用が期待できる。

さらに図 6.15 を見ると、各クラスターにおいても経験者と未経験者が存在している。つまり、ストロークフォームの一部では同じような演奏スタイルでも、その熟練には差があり、向上の余地があることが推測できる。

6.7　6 章のまとめ

5 章以降では、打楽器の演奏支援を、フォーム改善の観点から目指している。フォームの改善により、間接的に演奏者の演奏能力を向上させることを目的としている。以下に 6 章の結果をまとめる。

(1) 簡易モーションキャプチャシステムを用い実験を行ない、被験者の打楽器演奏時の腕の位置情報を取得した。

(2) 得られた演奏時の手首の位置の時間波形から、「ストローク誤差」、「パワースペクトルピーク比」、「振幅 RMS 値」の 3 種類の特徴量を抽出した。

(3) 打楽器経験者と未経験者によって演奏の際の右腕のストローク波形に違いがあるのかを t 検定によって分析した結果、3 種類のテンポごとに特徴量において、経験者と未経験者間で有意な差が見られた。

(4) 被験者から得られた特徴量を独立変数とした、ステップワイズ法による判別分析を行ない、経験者グループと未経験者グループ間の線形的な判別を行なった結果、3 種類のテンポにおいて、90.0%、95.0%、78.6% の判別率が得られた。

(5) 被験者から得られた特徴量よりクラスター分析を行なった結果、5 つのクラスターを得た。このクラスターは同じ被験者はほぼ同じクラスターに属していた。

上記の結果 (3) より、打楽器経験者及び未経験者の、ストローク波形の傾向が発見できた。本研究で用いた特徴量を用いることにより、打拍を行なったユーザへ対して、「フォームがずれました。」、「もう少しスナップを効かせて叩きましょう。」、「もう少し腕の振りを小さくしましょう。」等のフィードバックを行える可能性が示されたと言える。

また結果 (4) より、本研究で用いた特徴量より被験者の大よその分類が可能であることが示された。今回は「打楽器経験者」、「未経験者」という分類であったが、今回の結果を基にさらに被験者を増やし、波形的な特徴を持つグループに対する分類を行なうことにより、ユーザの演奏の特徴をつかむことが可能となる。

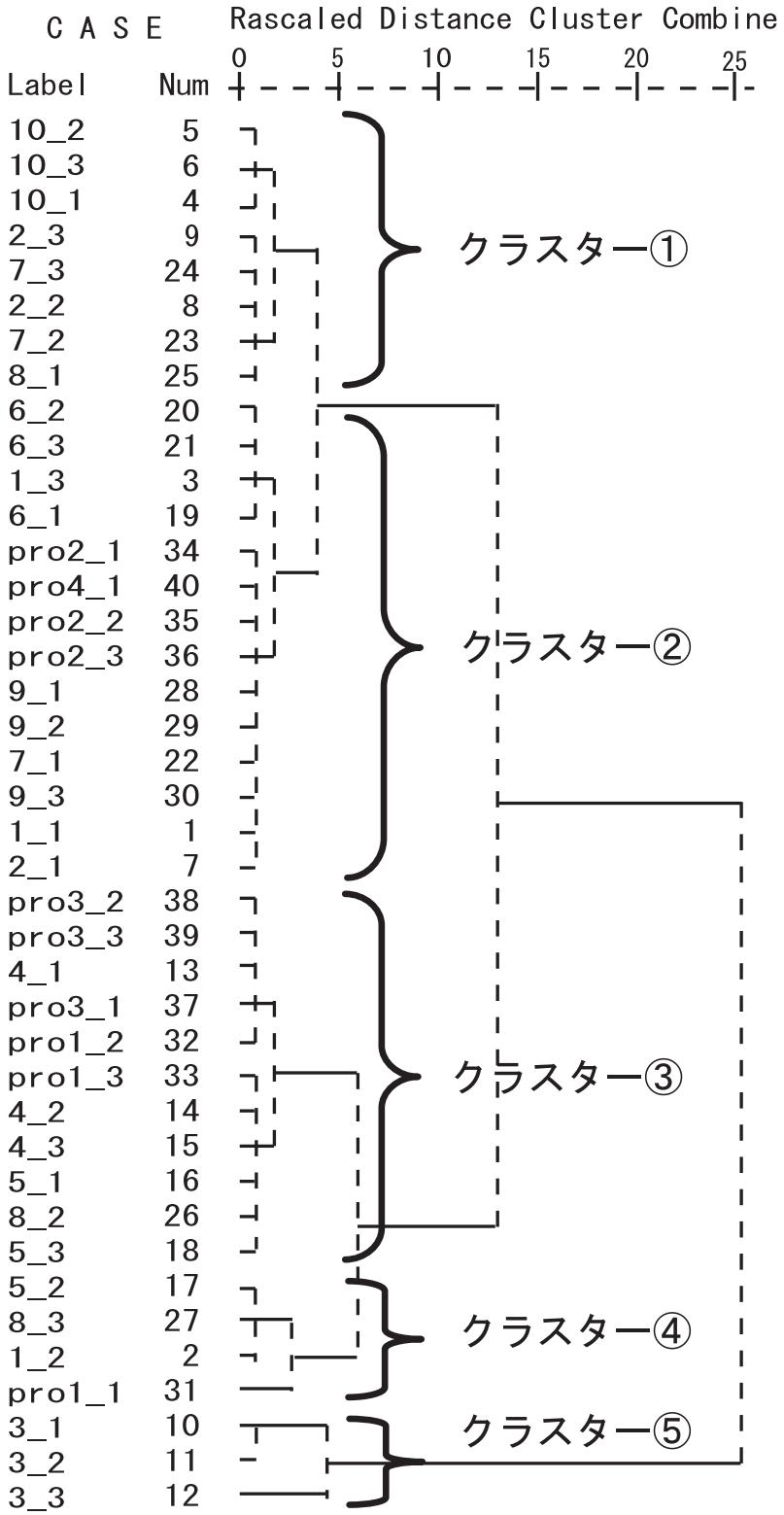


図 6.13: 90bpm の演奏におけるデンドログラム

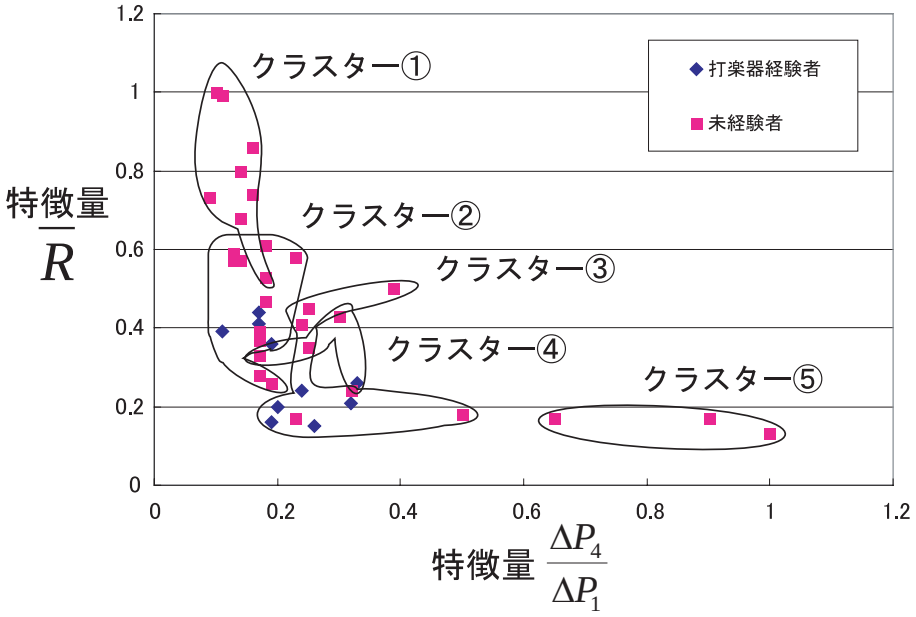


図 6.14: 90bpm の演奏における散布図-(1)

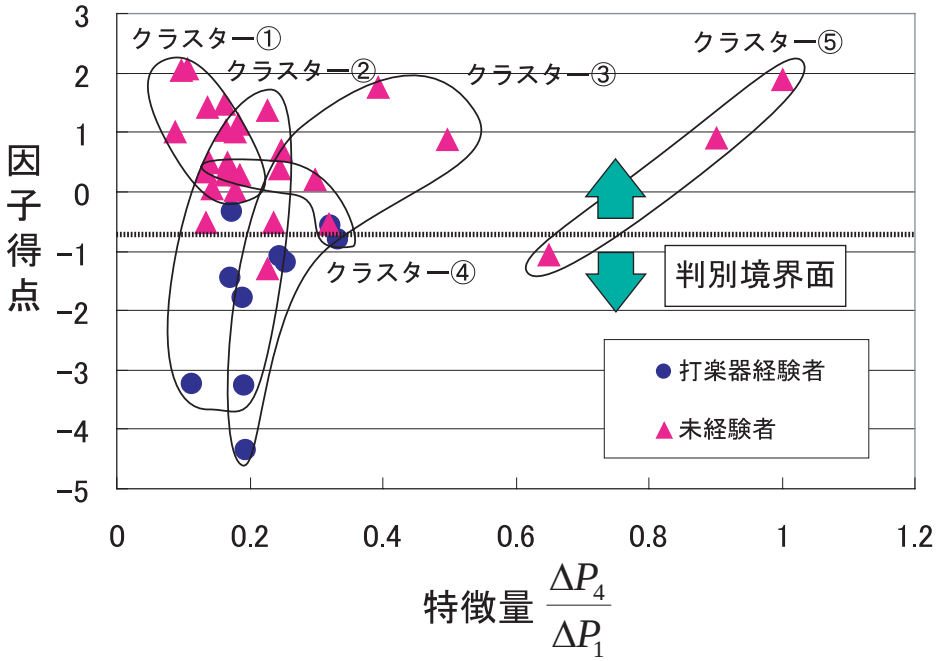


図 6.15: 90bpm の演奏における散布図-(2)

以上より、結果 (2) で得られた特徴量の有効性が示されたと言える。

次章では、本章では取り上げなかったリズムとフォームの同時測定及びそれらの関連性について述べる。

7章

一定間隔打拍時の右手首位置情報及び演奏情報の同時測定

7.1　はじめに

本章では、打楽器演奏時の手首の位置情報と、打拍時刻、打拍の強さ等の演奏情報の同時取得を行なった。そして前章と同様に、取得データからストローク誤差、振幅 RMS 値、リズム誤差、ベロシティ（打拍の強さ）の 4 種類の特徴量を抽出し、打楽器経験者と未経験者によって演奏の際の右腕のストローク波形及び演奏に違いがあるのかを統計処理を用いて分析した。処理には、経験者と未経験者とのグループ間での t 検定を行ない、特徴量ごとの有意差を定量的に求めた。

さらに、ストローク波形と演奏に関連があるかどうかを調べるために、ストローク誤差とリズム誤差、振幅 RMS 値とベロシティの各平均値に関して、相関係数を求めて散布図を調べた。

7.2　フォームデータと演奏データの同時測定

7.2.1　目的

前章の分析結果より、一定のテンポで、且つ音量も一定にして打拍させる実験課題においては、音量の絶対的な値を明確に提示しているわけではないので、各被験者の演奏する音量はそれぞれが叩きやすい音量になってしまっていたと考えられる。

また、前章の実験では演奏データは取得しなかったので、ストロークと演奏との関連性が不明である。

本実験の目的は、テンポ且つ音量において明示的に値を定めた提示音を各被験者に平等に聴取させた場合、ストローク及び演奏が打楽器経験者と未経験者でどのように違うのかを明らかにすること、さらに、ストロークの際の右手首位置情

報と、演奏情報を同時に取得する事によって、その関連性を明らかにすることである。

7.2.2　測定装置と実験課題

図 7.1 に、本実験で用いた、ストロークデータと演奏データの同時測定装置を示す。被験者がドラムパッドを打拍すると、演奏が MIDI データとなり、MIDI 音源からシリアルポートを通じてコンピュータへデータが送られる。さらに、CCD カメラを通じてモーションキャプチャセンサが被験者の右手の甲に付いているマーカの 3 次元位置情報を取得し、打拍している間の位置情報がシリアルポートからコンピュータへ 1/60[sec] おきに送られるが、この点は前章と同様である。

この一連の流れにより、ストロークデータと演奏 MIDI データの同時測定が可能となる。尚、開発及び実験には以下の環境を用いた

実験環境

OS　Windows2000 Professional

ソフトウェア開発環境　Visual C++ 6.0 Standard Edition

MIDI 音源　YAMAHA Drum Trigger Module DTXPRESS

MIDI 音源ドライバ　YAMAHA CBX-Driver V2.00 for Windows XP/2000/NT4.0

モーションキャプチャセンサ　KRI DigitEye3D

実験手順は以下の通りである。

被験者に一定の基準テンポ、一定の音の強さの手本となる打拍音を聞かせて記憶させた。その後、何も聞かない状態で 120 秒間、提示されたテンポの記憶を頼りにスティックを用いて手本通りに打拍させた。この測定より、各被験者の各試行につきそれぞれ 120 秒間の打拍ストローク時の右手の甲の 3 次元位置情報と、打拍時刻、打拍の強さといった演奏情報を同時に得た。基準テンポは 90BPM（Beats Per Minutes）、120BPM、180BPM の 3 種類のテンポを用いた。また、音の強さはベロシティが 41, 63, 117 の値の 3 段階（ベロシティは 0〜127 の範囲を取る）を用いた。従って $3 \times 3 =$ 合計 9 通りの手本を用いた。提示した順番は以下の通り。

提示順番

1. 90bpm Velocity=41 のスネアドラム音を提示

2. 90bpm Velocity=63

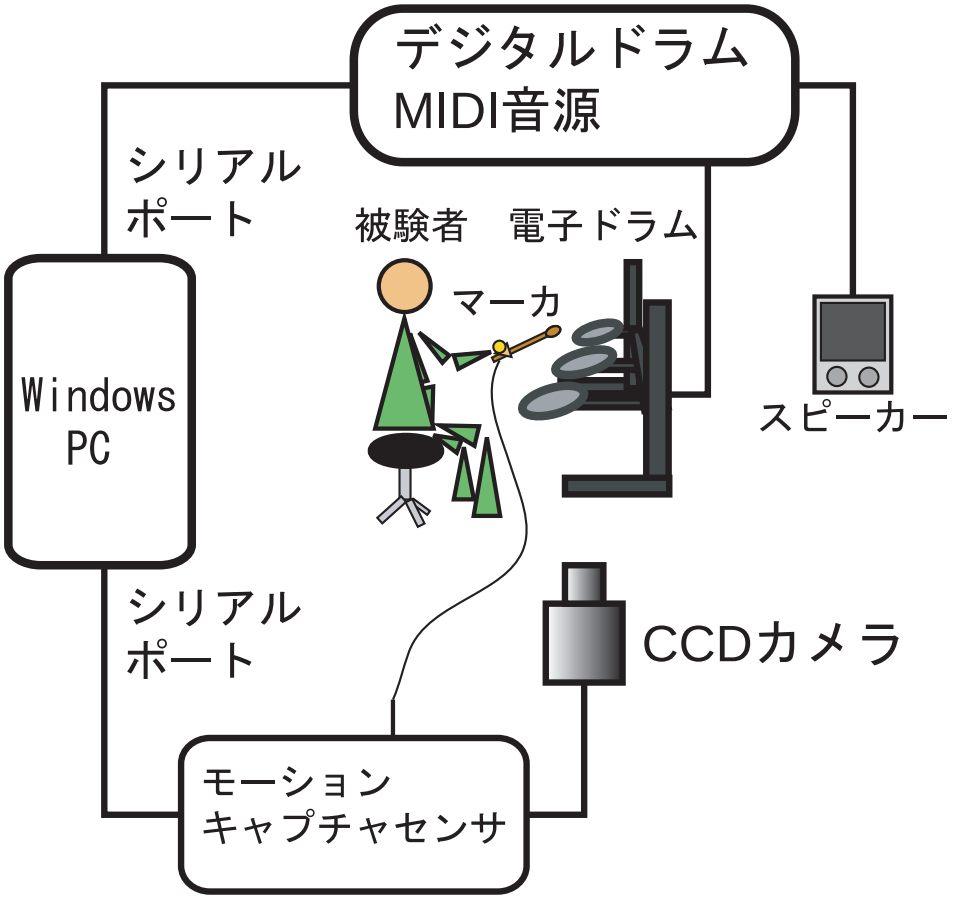

デジタルドラム
MIDI音源
シリアル
ポート
被験者　電子ドラム
マーカ
Windows
PC
スピーカー
シリアル
ポート
CCDカメラ
モーション
キャプチャセンサ

図 7.1: 測定環境

3. 90bpm Velocity=117

4. 120bpm Velocity=41

5. 120bpm Velocity=63

6. 120bpm Velocity=117

7. 180bpm Velocity=41

8. 180bpm Velocity=63

9. 180bpm Velocity=117

その各提示音に対して 1 回ずつ、従って被験者一人あたり 9 回ずつ試行した。被験者は、一般大学生及び大学院生の、1 週間に 8 時間で 5 年間以上の練習経験を持つ打楽器（ドラム）経験者 4 名、未経験者 10 名である。

7.2.3　測定データ

測定したデータは、以下の 3 種類である（図 7.2.3）。

- 手首位置情報 $\{\vec{f}(t)\}$, $t = 0, \Delta t, 2\Delta t, ..., 120$ sec, $\Delta t = \frac{1}{60}$ sec
- 打拍時刻系列 $\{t_1, t_2, ...\}$
- ベロシティ（打拍時の打鍵の強さ）$\{V_1, V_2, ...\}$

7.3　準備

特徴量を抽出し分析をする前に、各データのカスタマイズを行なう。具体的には打拍の際に一定ごとに生じると考えられるインパルス的変動点（米川ら,2001[53]）の削除を行なった。

7.3.1　インパルス的変動点の削除

米川らの手法に基づいて、測定した打拍時刻系列 $\{t_1, t_2, ...\}$ からそのインパルス的変動点を求め、削除した。削除したのは打拍時刻系列だけでなく、その時刻間における右手首位置情報、及びその時刻における打拍の強さ（ベロシティ）も同時に削除した。

米川らによれば、インパルス的変動点の除去は以下のように行なわれる（米川ら, 2001）。以下、引用を示す。

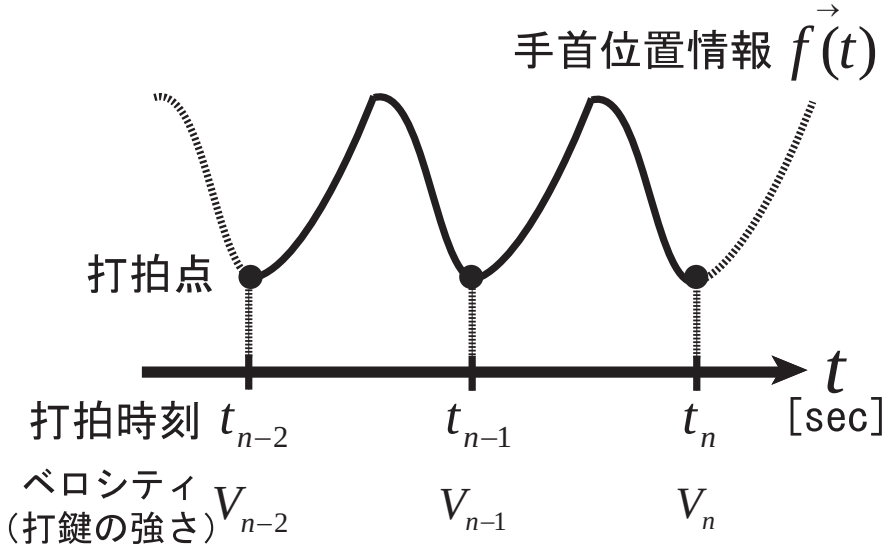


図 7.2: 測定したデータ

測定した打拍時刻系列 $\{t_1, t_2, ...\}$ から求められる、瞬時テンポ $\nu(t_i)$ は以下で定義される。

$$\nu(t_i) = \frac{60}{t_i - t_{i-1}},\ i = 2, 3, ...$$

瞬時テンポの系列 $\{\nu_i\}$ に含まれる長さ n+1 のある系列 $\{x_1, x_2, \ldots, x_{n+1}\}$ を考える (図 7.3)．いま，x_{n+1} について直前の n 点の平均 $\bar{x}$ と不偏分散 s^2 を求める．x_{n+1} が 区間 $[\bar{x} - cs, \bar{x} + cs]$ $(c > 0)$ に入らない場合，x_{n+1} をインパルス的変動とみなし，系列 $\{x_i\}$ から除去する．

尚、米川らによれば、2 分間の打拍テストにおいては、$c = 3, n = 15$ が最適であるとの知見が得られている [53]。本研究でもこの値を用いて除去を行なった。

7.4　特徴量の抽出

インパルス変動点を除去した $\vec{f(t)}$,$\{t_1, t_2, ...\}$,$\{V_1, V_2, ...\}$ において、以下の 4 種類の特徴量を抽出した。

- ストローク誤差（6.3.2 節）
- 振幅 RMS 値（6.3.4 節）
- リズム誤差

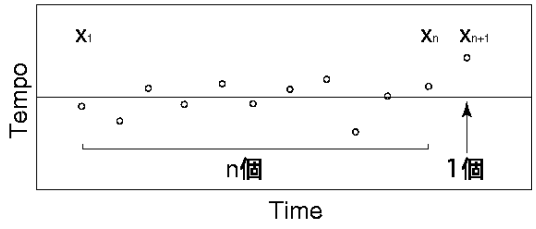


図 7.3: インパルス的変動の除去（[53] から引用）

- ベロシティ（打拍の強さ）

ストローク誤差、振幅 RMS 値に関してはそれぞれ 6.3.2 節、6.3.4 節で定義したアルゴリズムをそのまま適用した。それぞれ一人が一回 2 分間の演奏における平均と標準偏差 $\overline{E}, SD(E), \overline{R}, SD(R)$ を用いた。

また、インパルス変動点の除去によるカスタマイズを加えてない $f(\vec{t})$ より、以下の特徴量を抽出した。

- パワースペクトル密度関数の 1 次式近似 $y = ax + b$ の係数 a,b

リズム誤差、ベロシティ及びパワースペクトル密度関数の 1 次式近似に関しては次節で述べる。

7.4.1　リズム誤差

リズム誤差とは、ある周期と一つ前の周期の時間の差を数値化したものである。打拍時刻 t_{n-2}, t_{n-1}, t_n に対して、n 周期目のリズム誤差 T_n を以下に定義する 7.2.3。

$$T_n = \frac{|(t_n - t_{n-1}) - (t_{n-1} - t_{n-2})|}{(t_n - t_{n-2})/2},\ n = 3, 4, ...$$

本研究では、一人の一回の 2 分間の演奏における平均値 $\overline{T}$ 及び標準偏差 $SD(T)$ を特徴量として用いた。

7.4.2　ベロシティ

ベロシティの値はそのまま用いた。一人の一回の演奏における平均値 $\overline{V}$ 及び標準偏差 $SD(V)$ を特徴量として用いた。

7.4.3　パワースペクトル密度関数の 1 次式近似

ここでは、ストロークフォームに関する特徴量として、パワースペクトル密度（psd）関数の 1 次近似式 $y = af + b$ の係数 a,b に関して説明する。この特徴量は、6.3.3 節のパワースペクトル密度関数のピーク比とは異なり、ある周波数範囲における 1 次近似式である。

この 1 次近似式は次のアルゴリズムで抽出される。

psd 関数の 1 次近似アルゴリズム

1. 抽出したストロークフォームデータ $\{f(t)\}$ に対して、psd 関数 $\{f, \mathrm{psd}(f)\}(0 \leq f \leq 30)$ を求める

2. psd 関数の $0 \leq f \leq 8$[Hz] の範囲で、1 次近似式 $y = a_0 f + b_0$ を求める

3. $y = a_0 f + b_0$ に対して、$y = 0$ となる $f = f_{\mathrm{limit}}$[Hz] を求める

4. psd 関数の $0 \leq f \leq f_{\mathrm{limit}}$[Hz] の範囲で、一次近似式 $y = af + b$ を求める

5. 抽出した a,b が求める値

具体例を図 7.4 に示す。これは 90[bpm] のテンポにおける打楽器未経験者 N.U の演奏時の手首の垂直位置情報のパワースペクトル密度関数に対して、一次近似式を求めたものである。この一次近似式の係数より、演奏を行なった被験者の腕の振りがどの周波数帯を多く含んでいるかの割合が分かる。この係数 a,b を特徴量として、次節以降の分析に用いた。

7.5　分析

7.5.1　各特徴量の t 検定

前節で定義した特徴量 $\overline{E}$,$SD(E)$,$\overline{R}$,$SD(R)$,$\overline{T}$,
$SD(T)$,$\overline{V}$,$SD(V)$, 係数 a,b を用いて、打楽器経験者・未経験者のグループ間の t 検定を行なった。その結果、$\overline{R}$,$SD(R)$,$\overline{T}$, $SD(V)$, 係数 a,b に関して、有意水準 1%の有意差が見られた。その様子を図 7.5〜図 7.10 に示す。

具体的には、$\overline{R}$ は打楽器経験者の方が、未経験者よりもその値が有意に小さい事が分かった（$t(124) = 3.331, p < 0.01$）。$SD(R)$ もまた打楽器経験者の方が未経験者よりもその値が有意に小さい事が分かった（$t(124) = 3.349, p < 0.01$）。さらに、リズム誤差の平均、$\overline{T}$ も打楽器経験者の方が有意に値が小さい事が分かった（$t(124) = 4.087, p < 0.001$）。続いて、ベロシティの不安定性 $SD(V)$ に関しても打

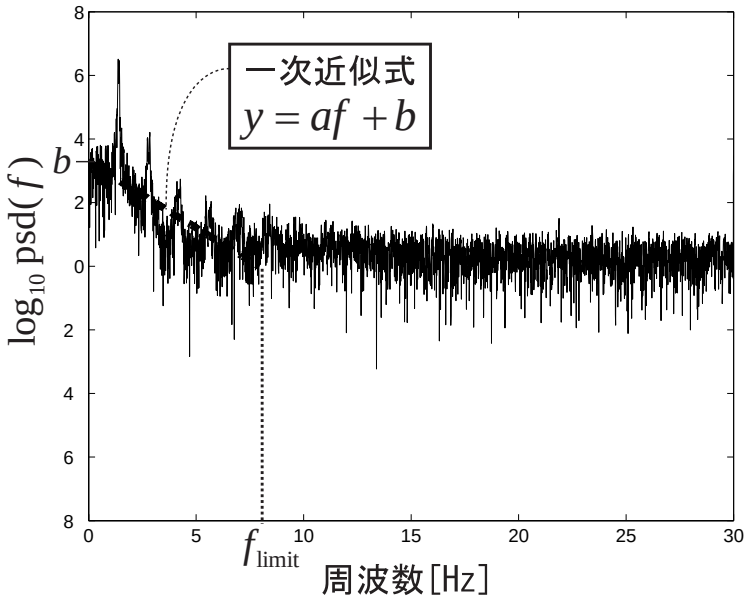


図 7.4: psd 関数の 1 次近似式

楽器経験者の方が有意に小さいという結果が得られた（$t(124) = 3.336, p < 0.01$）。係数 a に関しては打楽器経験者の方が値が小さい、すなわち絶対値が大きく、傾きが大きい事が分かった（$t(124) = -2.934, p < 0.01$）。係数 b、つまり y 切片に関しては打楽器経験者の方が値が有意に大きい事が分かった（$t(124) = 4.736, p < 0.001$）。

この結果より、打楽器経験者を未経験者と比較すると、

- 手首の振り幅が小さい傾向がある
- 手首の振り幅が一定である
- より小さいリズムのずれで演奏できる
- 一定の音量で演奏できる
- 手首の振り方に高周波成分を多く含む、すなわちスナップの効いた振り方をしている

という傾向があることが分かった。

7.5.2　ストローク誤差とリズム誤差の関連性

前節より、ストローク誤差に関しては打楽器経験者と未経験者間で有意な差が見られなかった。しかし、特に打楽器未経験者において、リズムがぶれる原因の

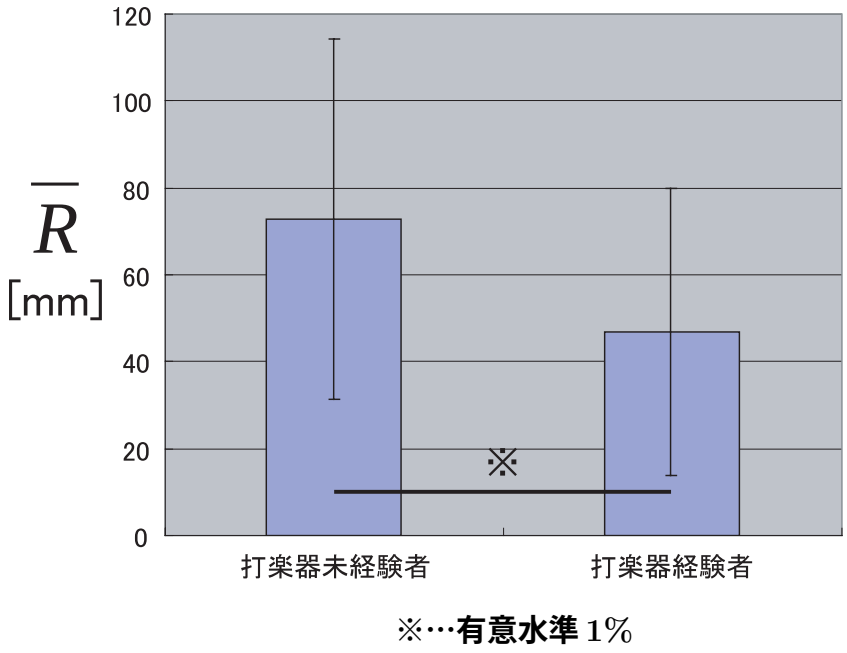


図 7.5: 打楽器経験者、未経験者における振幅 RMS 値平均の t 検定

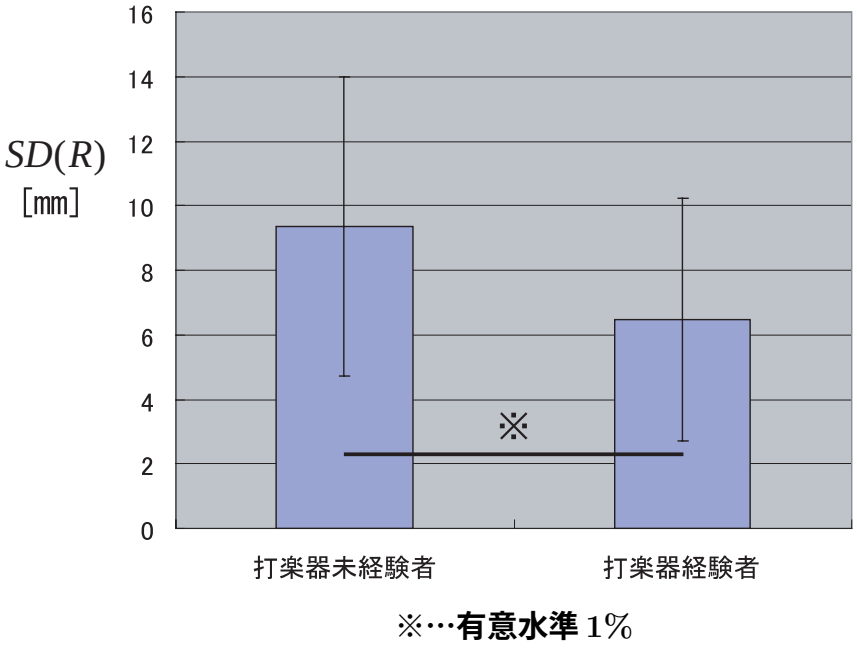


図 7.6: 打楽器経験者、未経験者における振幅 RMS 値標準偏差の t 検定

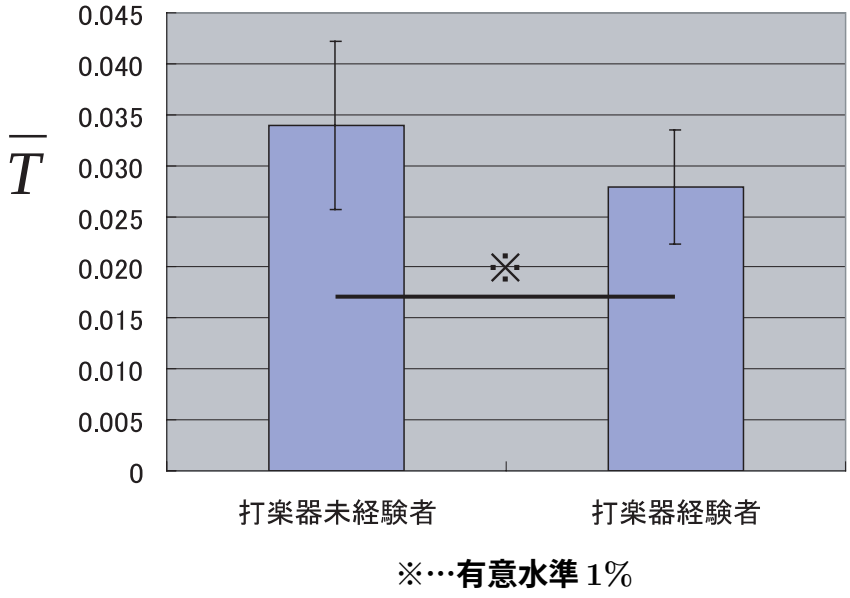


図 7.7: 打楽器経験者、未経験者におけるリズム誤差平均値の t 検定

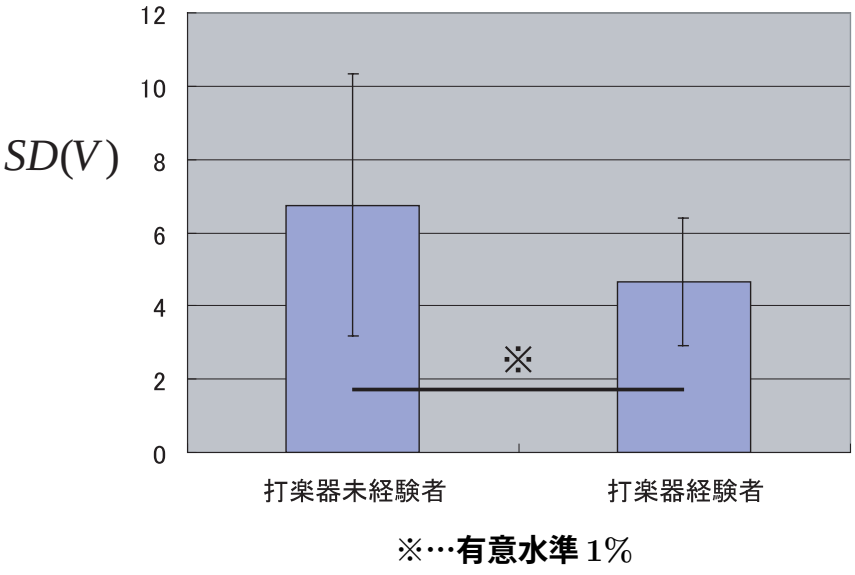


図 7.8: 打楽器経験者、未経験者におけるベロシティ標準偏差の t 検定

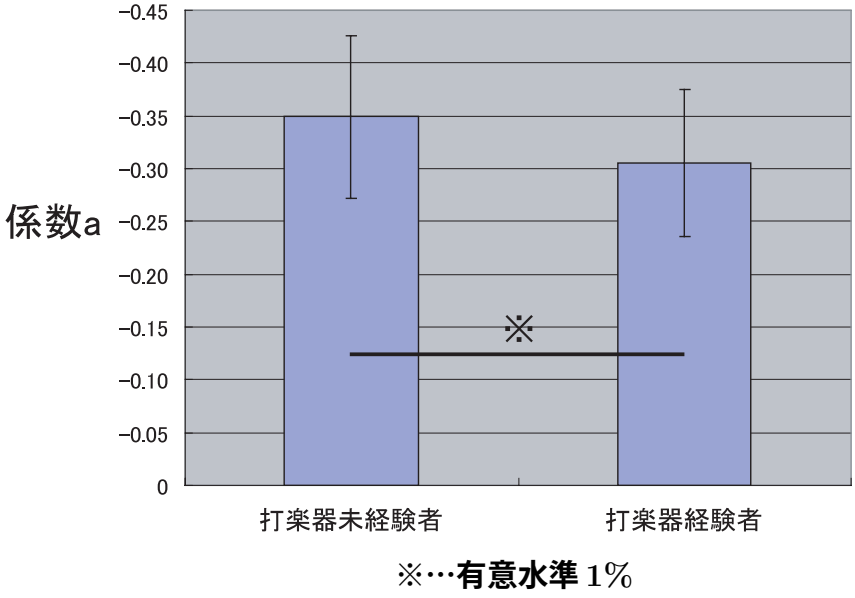


図 7.9: 打楽器経験者、未経験者における一次近似式の係数 a

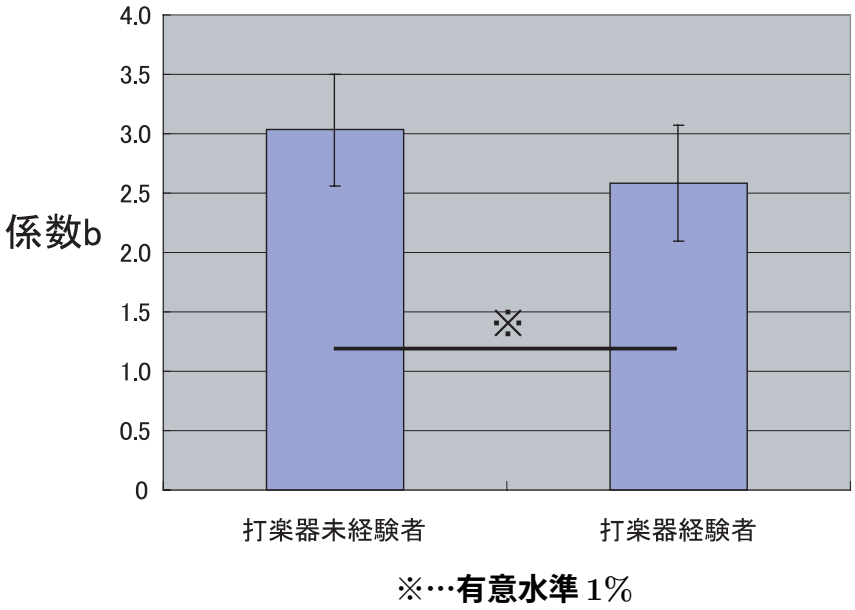


図 7.10: 打楽器経験者、未経験者における一次近似式の係数 b

表 7.1: 提示テンポ・音量による二元配置分散分析

特徴量	未経験者		経験者	
	テンポ	音量	テンポ	音量
$\overline{R}$	**	*	-	***
$SD(R)$	*	*	-	***
$\overline{V}$	-	***	-	***
$SD(V)$	-	***	-	***
a	***	***	-	-
b	***	***	**	**

* 有意水準 $p < .05$
*** 有意水準 $p < .01$
**** 有意水準 $p < .001$
- 有意傾向なし $p > .1$

一つとしてはストロークがぶれる事が考えられる。本節では、リズムとストロークフォームの関連性を調べるために、リズム誤差とストローク誤差の散布図を求め、その相関を求めた。

リズム誤差 $\overline{T}$ とストローク誤差 $\overline{E}$ の散布図は図 7.11 のようになった。また、それらの相関係数は全体で 0.464、打楽器未経験者は 0.627、打楽器未経験者においては 0.504 であった（いずれも $p < .01$ で有意）。この結果より、打楽器の経験者、未経験者を問わずリズム誤差とフォーム誤差は正の相関を持ち、片方が大きければもう片方が小さい可能性が示された。従って、片方を支援すればもう片方の要素が小さくなる可能性が期待できる。

7.5.3　打拍提示音、提示テンポにおける手首の振幅、演奏音量の違い

7.5.1 節の分析より、$\overline{R}$,$SD(R)$,$SD(V)$,a,b において、打楽器経験者と未経験者間で有意差が確認された。ここでは、手本として提示されたテンポ及び音量の違いにより打楽器経験者及び未経験者の手の振り、演奏音量がそれぞれ違うのかどうかを調べた。具体的には、提示されたテンポの 3 グループ、及び音量の 3 グループ間における、$\overline{R}$,$SD(R)$,$\overline{V}$,$SD(V)$,a,b の二元配置の分散分析を行なった。

表 7.1 に分散分析結果を示す。この結果より、打楽器経験者及び未経験者は共に、提示された音の音量に対して、それぞれ異なる振幅 RMS、音量、周波数の割

表 7.2: 打楽器未経験者の分散分析結果

		平方和	自由度	平均平方	F 値	有意確率
mean_R	ｸﾞﾙｰﾌﾟ間	13180.868	2	6590.434	4.095	.020
	ｸﾞﾙｰﾌﾟ内	140001.058	87	1609.208		
	合計	153181.926	89			
SD(R)	ｸﾞﾙｰﾌﾟ間	178.907	2	89.454	4.505	.014
	ｸﾞﾙｰﾌﾟ内	1727.471	87	19.856		
	合計	1906.379	89			
mean_V	ｸﾞﾙｰﾌﾟ間	71218.486	2	35609.243	414.548	.000
	ｸﾞﾙｰﾌﾟ内	7473.215	87	85.899		
	合計	78691.701	89			
SD(V)	ｸﾞﾙｰﾌﾟ間	322.182	2	161.091	16.993	.000
	ｸﾞﾙｰﾌﾟ内	824.752	87	9.480		
	合計	1146.934	89			
a	ｸﾞﾙｰﾌﾟ間	.133	2	6.631E-02	14.463	.000
	ｸﾞﾙｰﾌﾟ内	.399	87	4.585E-03		
	合計	.532	89			
b	ｸﾞﾙｰﾌﾟ間	3.271	2	1.636	8.643	.000
	ｸﾞﾙｰﾌﾟ内	16.466	87	.189		
	合計	19.737	89			

表 7.3: 打楽器未経験者の多重比較

Tukey HSD

従属変数	(I) VELOCITY	(J) VELOCITY	平均値の差 (I-J)	標準誤差	有意確率	95% 信頼区間 下限	95% 信頼区間 上限
mean_R	41	63	4.48555513	10.357630	.902	-20.21211	29.183224
		117	-23.13346	10.357630	.071	-47.83112	1.56421237
	63	41	-4.485555	10.357630	.902	-29.18322	20.212113
		117	-27.61901*	10.357630	.025	-52.31668	-2.921343
	117	41	23.133456	10.357630	.071	-1.564212	47.831124
		63	27.619011*	10.357630	.025	2.92134277	52.316680
SD(R)	41	63	-1.950886	1.15053584	.213	-4.694328	.79255510
		117	-3.443419*	1.15053584	.010	-6.186860	-.69997707
	63	41	1.95088640	1.15053584	.213	-.79255510	4.69432790
		117	-1.492532	1.15053584	.401	-4.235974	1.25090933
	117	41	3.44341857*	1.15053584	.010	.69997707	6.18686006
		63	1.49253217	1.15053584	.401	-1.250909	4.23597366
mean_V	41	63	-20.96402*	2.39303186	.000	-26.67018	-15.25786
		117	-67.32659*	2.39303186	.000	-73.03276	-61.62043
	63	41	20.964023*	2.39303186	.000	15.257862	26.670184
		117	-46.36257*	2.39303186	.000	-52.06873	-40.65641
	117	41	67.326594*	2.39303186	.000	61.620433	73.032756
		63	46.362572*	2.39303186	.000	40.656410	52.068733
SD(V)	41	63	-1.918180*	.79498035	.047	-3.813803	-2.26E-02
		117	-4.612793*	.79498035	.000	-6.508416	-2.717170
	63	41	1.91817973*	.79498035	.047	2.256E-02	3.81380264
		117	-2.694613*	.79498035	.003	-4.590236	-.79899016
	117	41	4.61279280*	.79498035	.000	2.71716989	6.50841571
		63	2.69461307*	.79498035	.003	.79899016	4.59023597
a	41	63	-2.58E-02	1.748E-02	.308	-6.75E-02	1.589E-02
		117	-9.12E-02*	1.748E-02	.000	-.13289771	-4.95E-02
	63	41	2.580E-02	1.748E-02	.308	-1.59E-02	6.749E-02
		117	-6.54E-02*	1.748E-02	.001	-.10709844	-2.37E-02
	117	41	9.121E-02*	1.748E-02	.000	4.952E-02	.13289771
		63	6.541E-02*	1.748E-02	.001	2.372E-02	.10709844
b	41	63	-.19809087	.11232736	.188	-.46593435	6.975E-02
		117	-.46529650*	.11232736	.000	-.73313999	-.19745301
	63	41	.19809087	.11232736	.188	-6.98E-02	.46593435
		117	-.26720563	.11232736	.051	-.53504912	6.379E-04
	117	41	.46529650*	.11232736	.000	.19745301	.73313999
		63	.26720563	.11232736	.051	-6.38E-04	.53504912

*. 平均の差は .05 で有意

表 7.4: 打楽器経験者の分散分析結果

		平方和	自由度	平均平方	F 値	有意確率
mean_R	ｸﾞﾙｰﾌﾟ間	16974.785	2	8487.393	13.245	.000
	ｸﾞﾙｰﾌﾟ内	21146.409	33	640.800		
	合計	38121.194	35			
sd_R	ｸﾞﾙｰﾌﾟ間	277.388	2	138.694	21.274	.000
	ｸﾞﾙｰﾌﾟ内	215.138	33	6.519		
	合計	492.526	35			
mean_V	ｸﾞﾙｰﾌﾟ間	34151.738	2	17075.869	407.780	.000
	ｸﾞﾙｰﾌﾟ内	1381.882	33	41.875		
	合計	35533.620	35			
sd_V	ｸﾞﾙｰﾌﾟ間	49.063	2	24.531	13.845	.000
	ｸﾞﾙｰﾌﾟ内	58.471	33	1.772		
	合計	107.534	35			
a	ｸﾞﾙｰﾌﾟ間	8.329E-03	2	4.165E-03	.859	.433
	ｸﾞﾙｰﾌﾟ内	.160	33	4.849E-03		
	合計	.168	35			
b	ｸﾞﾙｰﾌﾟ間	4.231	2	2.116	16.991	.000
	ｸﾞﾙｰﾌﾟ内	4.109	33	.125		
	合計	8.340	35			

表 7.5: 打楽器経験者の多重比較

Tukey HSD

従属変数	(I) VELOCITY	(J) VELOCITY	平均値の差 (I-J)	標準誤差	有意確率	95% 信頼区間 下限	95% 信頼区間 上限
mean_R	41	63	-8.177553	10.334411	.711	-33.53609	17.180981
		117	-49.60464*	10.334411	.000	-74.96317	-24.24611
	63	41	8.17755258	10.334411	.711	-17.18098	33.536086
		117	-41.42709*	10.334411	.001	-66.78562	-16.06855
	117	41	49.604640*	10.334411	.000	24.246107	74.963174
		63	41.427088*	10.334411	.001	16.068554	66.785621
SD(R)	41	63	-.38440442	1.04237969	.928	-2.942191	2.17338247
		117	-6.071203*	1.04237969	.000	-8.628989	-3.513416
	63	41	.38440442	1.04237969	.928	-2.173382	2.94219130
		117	-5.686798*	1.04237969	.000	-8.244585	-3.129011
	117	41	6.07120250*	1.04237969	.000	3.51341562	8.62898938
		63	5.68679808*	1.04237969	.000	3.12901120	8.24458497
mean_V	41	63	-25.94740*	2.64181747	.000	-32.42988	-19.46492
		117	-74.32528*	2.64181747	.000	-80.80776	-67.84280
	63	41	25.947399*	2.64181747	.000	19.464919	32.429880
		117	-48.37788*	2.64181747	.000	-54.86036	-41.89540
	117	41	74.325282*	2.64181747	.000	67.842801	80.807763
		63	48.377883*	2.64181747	.000	41.895402	54.860363
SD(V)	41	63	-1.418253*	.54342413	.035	-2.751704	-8.48E-02
		117	-2.859540*	.54342413	.000	-4.192992	-1.526088
	63	41	1.41825250*	.54342413	.035	8.480E-02	2.75170434
		117	-1.441288*	.54342413	.032	-2.774739	-.10783574
	117	41	2.85954008*	.54342413	.000	1.52608824	4.19299193
		63	1.44128758*	.54342413	.032	.10783574	2.77473943
a	41	63	6.827E-03	2.843E-02	.969	-6.29E-02	7.658E-02
		117	-2.83E-02	2.843E-02	.585	-9.81E-02	4.145E-02
	63	41	-6.83E-03	2.843E-02	.969	-7.66E-02	6.293E-02
		117	-3.51E-02	2.843E-02	.441	-.10489088	3.462E-02
	117	41	2.831E-02	2.843E-02	.585	-4.14E-02	9.806E-02
		63	3.513E-02	2.843E-02	.441	-3.46E-02	.10489088
b	41	63	-.36032633*	.14405512	.045	-.71380819	-6.84E-03
		117	-.83706558*	.14405512	.000	-1.190547	-.48358373
	63	41	.36032633*	.14405512	.045	6.844E-03	.71380819
		117	-.47673925*	.14405512	.006	-.83022110	-.12325740
	117	41	.83706558*	.14405512	.000	.48358373	1.19054744
		63	.47673925*	.14405512	.006	.12325740	.83022110

*. 平均の差は .05 で有意

合（係数 a, b）で演奏していた可能性が示された。これより、音量を変えて演奏する際には、打拍フォームも変えて演奏している可能性があることが分かった。

表 7.2 と表 7.3 に打楽器未経験者の、そして表 7.4 と表 7.5 に打楽器経験者のにおける、提示音量グループ（Velocity の値＝ $41, 63, 117$）間の各特徴量の分散分析及び多重比較の結果を示す。これより、提示打拍音の音量の違いによって、打楽器経験者、未経験者がともに異なる振幅、及び音量で演奏していたことが分かる。特に音量 $\overline{V}$ に関しては、多重比較の結果を見ると 3 種類の提示音（Velocity=41,63,117：以下、提示音が小、中、大と呼ぶ）のすべての場合において有意水準 1% の有意差が得られた。この事より、打楽器経験者、未経験者共に提示音を記憶した上、音量に気を配った上で打拍を行なった事が分かる。手首の振幅 RMS 値に関しても、

- 提示音が小 $\Leftrightarrow$ 提示音が大
- 提示音が中 $\Leftrightarrow$ 提示音が大

の組合せにおいて、有意に振幅の値が異なる結果が得られた（いずれも有意水準 1%）。しかし、打楽器経験者と未経験者では振幅 RMS の平均値 $\overline{R}$ 及び標準偏差 $SD(R)$ の値が違うため、両グループ間にはその大きさに差があると考えられる。

しかし、提示されたテンポに関しては、打楽器未経験者が異なる振幅 RMS、振幅 RMS の不安定性、一次近似式の傾きであったのに対し、経験者の方はいずれの特徴量においても有意な差は確認できなかった。一次近似式の傾き a は、打拍フォーム波形において、低周波数に対する高周波数の割合を示していると言える。つまり、この値が異なる事は、打拍フォーム波形において、スナップなどの急な動作を含む割合が異なる事に関連があると言える。

このことより、打楽器未経験者は打拍フォームがテンポに影響されやすいのに対し、打楽器経験者はテンポが変わっても打拍フォームには影響しない可能性が示されたと言える。この理由は、打楽器経験者は打拍フォームの軌道が安定している事と、さまざまなテンポで練習した経験がある事が考えられる。

7.5.4　手首の振幅と打拍の強さの関連性

前節では各提示テンポ及び音量における、$\overline{R}$,$SD(R)$,$\overline{R}$,$SD(V)$,a,b の違いを求めたが、ここでは、手の振りと演奏した音の強さの直接的な関連性を調べるために、振幅 RMS 値の平均 $\overline{R}$ とベロシティの平均 $\overline{V}$ との相関を調べた。

図 7.12 が $\overline{R}$ と $\overline{V}$ の散布図である。そして、打楽器経験者群と打楽器未経験者群における、$\overline{R}$ と $\overline{V}$ の相関係数を求めた。その結果、打楽器未経験者群においては 0.276 であったのに対し、打楽器経験者群においては 0.565 であり、やや正の相関が見られた（いずれも有意水準 $p < .01$）。この結果より、打楽器経験者の方が手の振りと打拍の強さの相関性がより強いのではないかと考えられる。

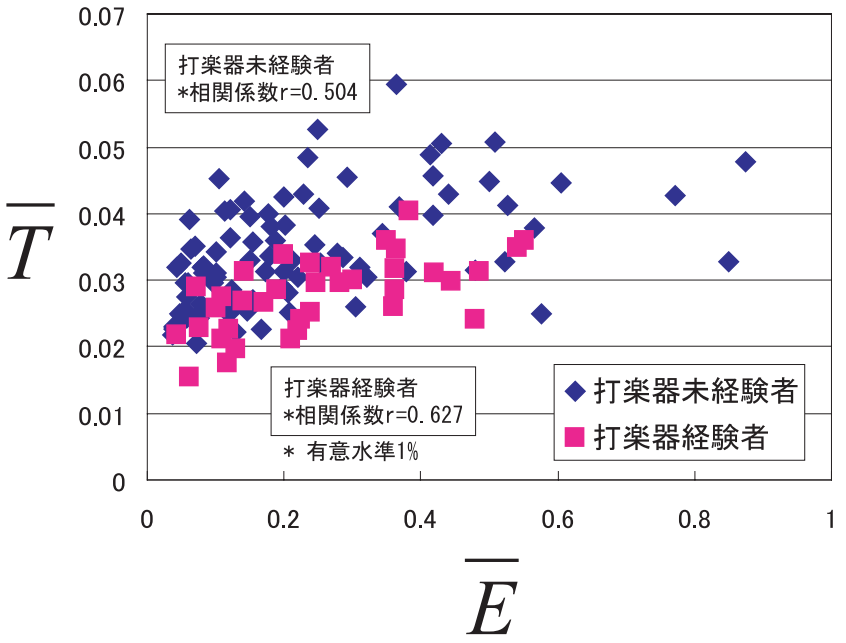


図 7.11: $\overline{E}$,$\overline{T}$ の散布図

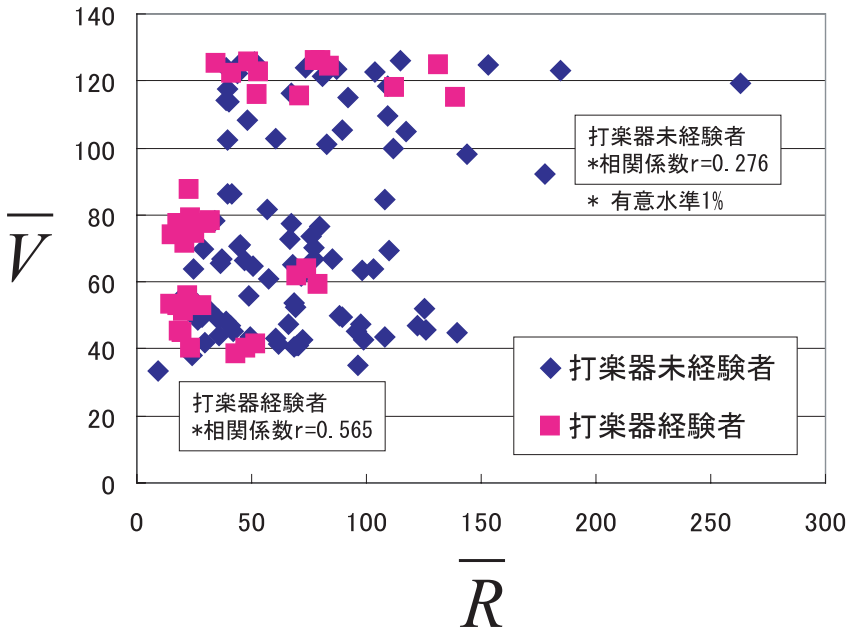


図 7.12: $\overline{R}$,$\overline{V}$ の散布図

7.6　7章のまとめ

本章では、打楽器演奏時の手首の位置情報と、打拍時刻、打拍の強さ等の演奏情報の同時取得を行なった。そして前章と同様に、取得データからストローク誤差、振幅 RMS 値、リズム誤差、ベロシティ（打拍の強さ）、パワースペクトル密度関数の 1 次近似式の係数の 5 種類の特徴量を抽出し、打楽器経験者と未経験者によって演奏の際の右腕のストローク波形及び演奏に違いがあるのかを統計処理を用いて分析した。処理には、経験者と未経験者とのグループ間での t 検定を行ない、特徴量ごとの有意差を定量的に求めた。

その結果、$\overline{R}$,$SD(R)$,$\overline{T}$,$SD(V)$,a,b に関して、有意水準 1%の有意差が見られた。この結果より、打楽器経験者を未経験者と比較すると、

- 手首の振り幅が小さい傾向がある
- 手首の振り幅が一定である
- より小さいリズムのずれで演奏できる
- 一定の音量で演奏できる
- 手首の振り方に高周波成分を多く含む、すなわちスナップの効いた振り方をしている

という傾向があることが分かった。

さらに、ストローク波形と演奏に関連があるかどうかを調べるために、ストローク誤差とリズム誤差に関して散布図を調べ、相関係数を求めた。その結果、打楽器経験者、未経験者共に軽い正の相関が見られた。このことより、フォームを改善すればリズムの短期的なずれが改善される可能性が期待できる。

さらに、提示テンポ及び音量の違いによって被験者の手の振り及び演奏音量がどう変化するかを調べるために、振幅 RMS 値とベロシティの平均、標準偏差、psd 関数の一次近似式の係数に対して二元配置の分散分析を行なった。その結果、打楽器未経験者においては提示音量だけでなく、提示テンポが異なる場合にも振幅 RMS 値 $\overline{R}, SD(R)$ や一次近似式の係数 a が異なるのに対し、打楽器経験者においては提示音量にのみ振幅 RMS 値 $\overline{R}, SD(R)$ や係数 a が有意に異なる結果が得られた。この事より、打楽器未経験者は提示された音量だけでなくテンポが異なる場合にも打拍フォームに影響を受けるのに対し、経験者は提示テンポが異なっても打拍フォームには影響を受けないことが分かった。

続いて、振幅 RMS 値とベロシティの各平均値に関して、相関係数を求めて散布図を調べた。

その結果、打楽器未経験者群よりも、打楽器経験者群においてやや強い正の相関が見られた。この結果より、打楽器経験者の方が手の振りと打拍の強さの相関性がより強いのではないかと考えられる。

8章

ジャンル推定能力と打楽器演奏能力の関連性の検証

8.1 はじめに

本章では、打楽器演奏能力と、ジャンル推定能力の関連性について検討した。打楽器演奏能力とジャンル推定能力の相関性を仮に示せれば、打楽器演奏の支援だけでなく、ジャンルを学習支援することも打楽器学習において有効である事が示唆されると考えられる。

始めに、前章で行なった実験の被験者の一部を対象に、打楽器パートのみのジャンル推定実験を行なった。そして打楽器経験者と未経験者において、打楽器演奏能力とジャンル推定正解率とを比較した。

8.2 打楽器パートのジャンル推定実験

打楽器演奏能力とジャンル推定能力の相関性を調べるために、打楽器パートに限定したジャンル推定実験を行なった。

8.2.1 方法

実験は以下の手続きで行なわれた。

被験者 7.2 節の実験をした大学生及び大学院生 8 名。その内、打楽器経験者 3 名、未経験者 5 名。

実験環境 実験は以下の環境で行なわれた。

- 実験マシンの OS: WindowsXP HomeEdition
- 使用したアプリケーション

  - WindowsMediaPlayer（MIDI 再生用）
  - YAMAHA CBX-Driver V2.00 for Windows XP/2000/NT4.0（MIDI 音源ドライバ）
- 使用した MIDI ファイル
  - 市販のジャンル別 MIDI 素材集 [56] より打楽器パートのみを抽出した、ハードロック、プログレッシブ・ロック、P ファンク、ヘヴィメタル、ジャズ、フュージョン、演歌の 7 ジャンル計 20 曲（各演奏時間:約 7〜30 秒）
- 周辺装置:
  - YAMAHA MU2000（MIDI 音源）
  - SONY Stereo Headphones MDR-CD2000（密閉式ヘッドフォン）

実験手続き 被験者には、実験者が予め打楽器パートのみかつイフェクト、イコライジングなどの処理をなくした General Midi 対応の SMF（Standard MIDI File）形式のファイルを 20 曲聴取させ、その打楽器パートがどのジャンルに最もふさわしいかを、ハウス、テクノに素材基の 7 ジャンルを加えた 9 ジャンルからの選択式設問で回答させた。さらに、選択肢には ”分からない ”の項目も設けた。各 SMF ファイルの演奏時間は、約 7〜30 秒程度である。1 つの SMF ファイルにつき聴取回数は 3 回までとした。さらに、回答は設問 1 から順番に回答させた。これは、被験者が回答する前にすべてのファイルを予め聴取し、その後で似てるかどうかでグルーピングしてから回答する方法を避けるためである。

8.2.2　結果

表 8.1: 打楽器パートのジャンル推定率

被験者種別	打楽器未経験者	打楽器経験者	合計
人数	5	3	8
平均推定率	18.0%*	63.3%*	35.0%
標準偏差	9.1	7.6	24.8

* $p < .001$ で有意

推定結果を表 8.1 に示す。被験者すべての推定率は 35.0%であった。そして打楽器未経験者の平均推定率が 18.0%だったのに対し、打楽器経験者は 63.3%であっ

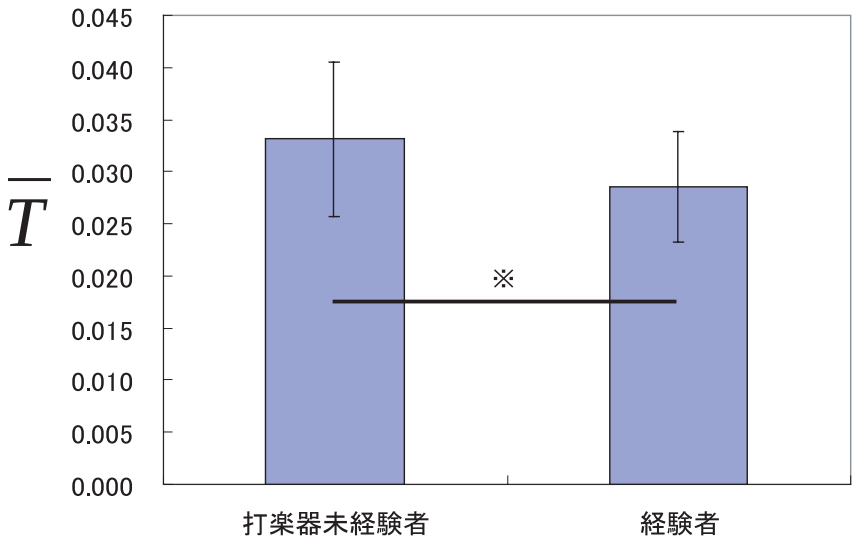


図 8.1: リズム誤差 $\overline{T}$ の比較

た（$t(6) = -7.195, p < .001$）。この結果より、打楽器経験者の方がジャンルごとの打楽器フレーズについて良く知っている、もしくは CD などを聴取する際に打楽器パートに対してより神経を注いで聴取している可能性が示唆された。その反対に、打楽器未経験者は打楽器パートのみを聴取してもそのジャンルが分からない、もしくはイメージの沸かない可能性が示された。さらに、打楽器フレーズとジャンルの関連性を学習する事により、打楽器未経験者に対してより打楽器への興味を向上させることが期待できる。

8.3　打楽器演奏能力との関連性の検証

前節の被験者における、打楽器演奏能力とジャンル推定能力の関連性をここで示す。

打楽器演奏能力として、7.4.1 節で定義したリズム誤差に関して今回の被験者 8 人に対して打楽器経験者 ⇔ 未経験者間に有意差が存在するかどうかを t 検定で調べた。

その結果を図 8.1 に示す。今回の被験者 8 人に対しても、そのリズム誤差の平均値 $\overline{T}$ に対して有意に打楽器経験者の方が小さな値を取る事が分かった（$t(70) = 2.799, p < .01$）。前章の結果をふまえると今回の被験者 8 人においては、打楽器経験者の 3 人は少なくとも打楽器パートのジャンル推定率とリズム安定性において有意に能力が高い事が分かった。しかし、リズム誤差とジャンル推定率に対して

相関係数を求めた所、有意な値は得られなかった。

8.4　8 章のまとめ

本章では、打楽器演奏能力と、ジャンル推定能力の関連性について検討した。

前章で行なった実験の被験者の一部を対象に、打楽器パートのみのジャンル推定実験を行なった。そして打楽器経験者と未経験者において、打楽器演奏能力とジャンル推定正解率とを比較した。

その結果、被験者 8 人に対して、打楽器経験者 3 人と未経験者 5 人では、打楽器未経験者の平均推定率が 18.0%だったのに対し、打楽器経験者は 65.0%であり、有意な差が確認できた（$t(6) = -7.195, p < .001$）。さらに、リズム誤差の平均値 $\overline{T}$ において、有意に打楽器経験者の方が小さな値を取る事が分かった（それぞれ $t(70) = 2.799, p < .01$）。

これより、打楽器経験者と未経験者では、打楽器パートのジャンル推定能力及びリズム安定性に有意差があることが分かる。従って、打楽器未経験者にジャンルについて学習させる事の有効性が期待できる。

9章

打楽器を対象とした学習支援システムの開発と評価

9.1 はじめに

本章では、打楽器に関する学習支援システムを開発し、その評価を行なった。

今まで、楽器演奏の学習支援に関するさまざまな研究が行なわれている。リズムの学習支援に関する従来の研究としては、西井らの実演音とクリック音との聴覚的及び視覚的ずれをフィードバックする研究 [59]、米川らのドラムパッドを用いたリズム音楽基礎能力評価法に関する研究がある [53]。

これらの研究はいずれもユーザの演奏したリズムに関する情報のみをフィードバックしており、その叩き方に関する情報はフィードバックされていない。

本論文では、米川らのシステムのリズム支援機能に対して、ストロークフォーム支援機能、打拍安定性支援機能を付与したシステムを開発し、その評価を行なった。

9.2 システム構成

打楽器学習支援システムの装置構成を図 9.1 に示す。装置の構成は以前の実験と同様であるが、ディスプレイ及びスピーカによる視覚的、聴覚的フィードバックを行なえる点が異なる。

システム構成

OS Windows2000 Professional

ソフトウェア開発環境 Visual C++ 6.0 Standard Edition

MIDI 音源 YAMAHA Drum Trigger Module DTXPRESS

MIDI 音源ドライバ YAMAHA CBX-Driver V2.00 for Windows XP/2000/NT4.0

モーションキャプチャセンサ KRI DigitEye3D

本アプリケーションは VisualC++6.0 のダイアログベースアプリケーションとして作成した。メインとなるクラスの中で、各ボタンがクリックされた際の処理、演奏／ストロークフォームデータの記録、演奏データのクラスタリングなどが行なわれている。

9.3　システムの流れ

図 9.2 に打楽器学習支援システムの流れを、図 9.3 にインターフェイスを示す。

被験者は始めに名前を入力する。続いてテンポを 90[bpm] と 120[bpm] のいずれかから選択し、さらに練習したいリズムパターンを選ぶ。ここで好みによって、「アクセント付与」ボタンをクリックする事によりアクセント記号付きのリズムパターンを練習する事も可能である。そして演奏時間を秒単位で入力し、「開始」ボタンをクリックして打拍を行なう。演奏時間は一番最初の打拍をした瞬間からカウントされ、演奏と手の位置情報の測定が始まる。指定した演奏時間が過ぎると「分析しています」と音声で伝えられ、分析が始まる。

分析が終るとその結果がグラフと音声でフィードバックされる。具体的には、

- 1 小節あたりの音長の偏り
- 1 小節あたりの音長の不安定性（標準偏差）
- 各音符あたりのストロークフォームのずれ
  $\overline{E_k}$, k：音符数
- 各音符あたりの手の振り幅
  $\overline{S_k}$
- 各音符あたりの手の振り幅の不安定性（標準偏差）
  $SD(S_k)$
- 各音符の打拍の強さ（ベロシティ）
  $\overline{V_k}$
- 各音符の打拍の強さの不安定性（標準偏差）
  $SD(V_k)$

がフィードバックされる。

また、フィードバックには用いられていないが、演奏分析の為に以下の特徴量を抽出している。

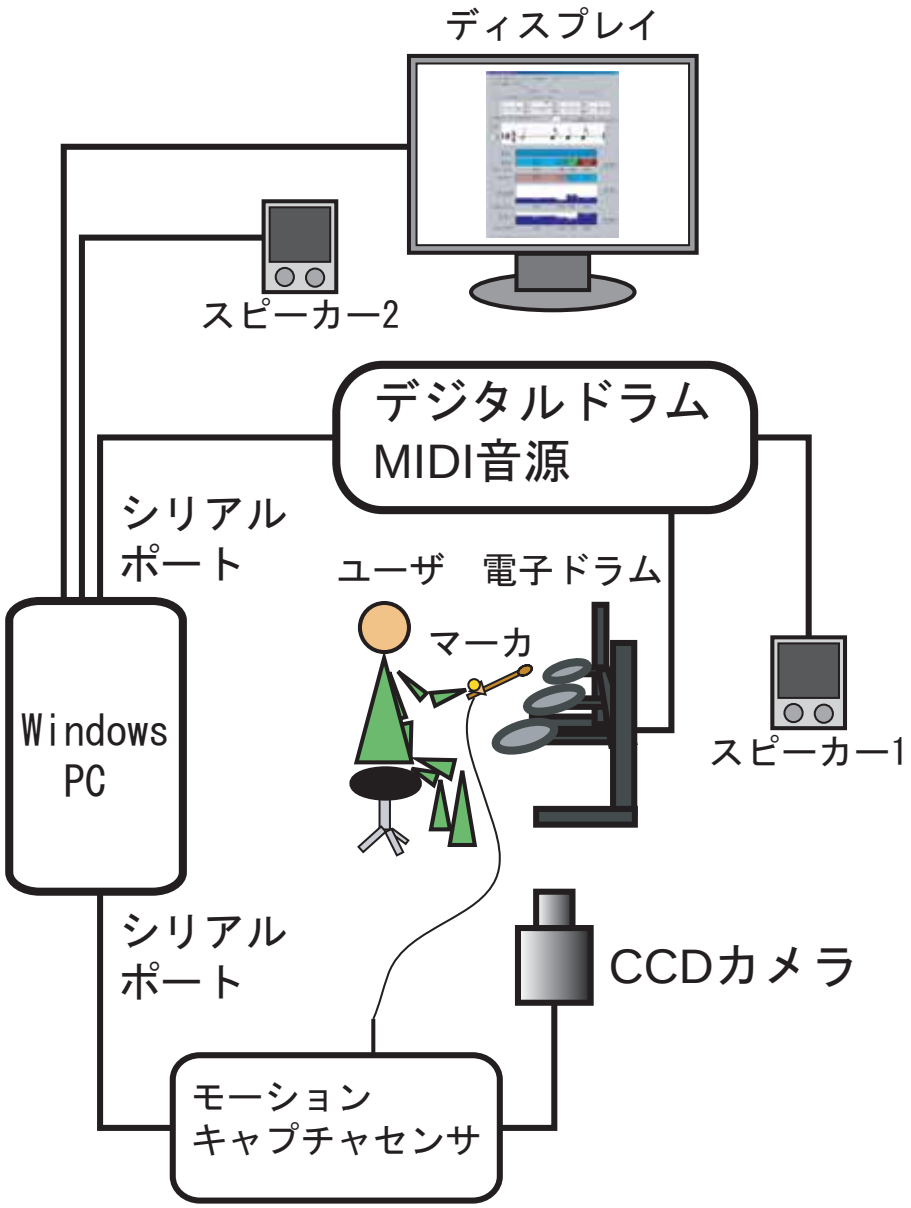


図 9.1: 打楽器学習支援システムの装置構成図

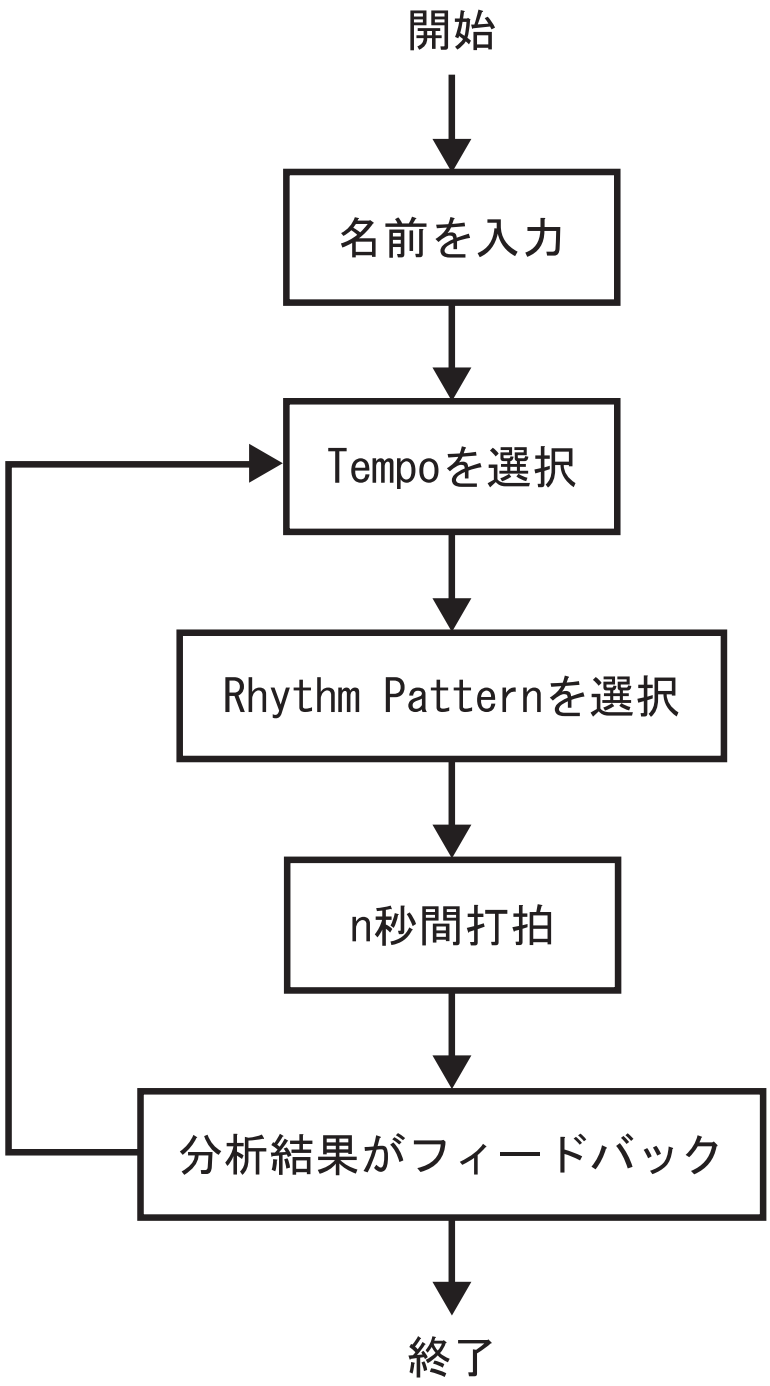

開始
名前を入力
Tempoを選択
Rhythm Patternを選択
n秒間打拍
分析結果がフィードバック
終了

図 9.2: システムの流れ

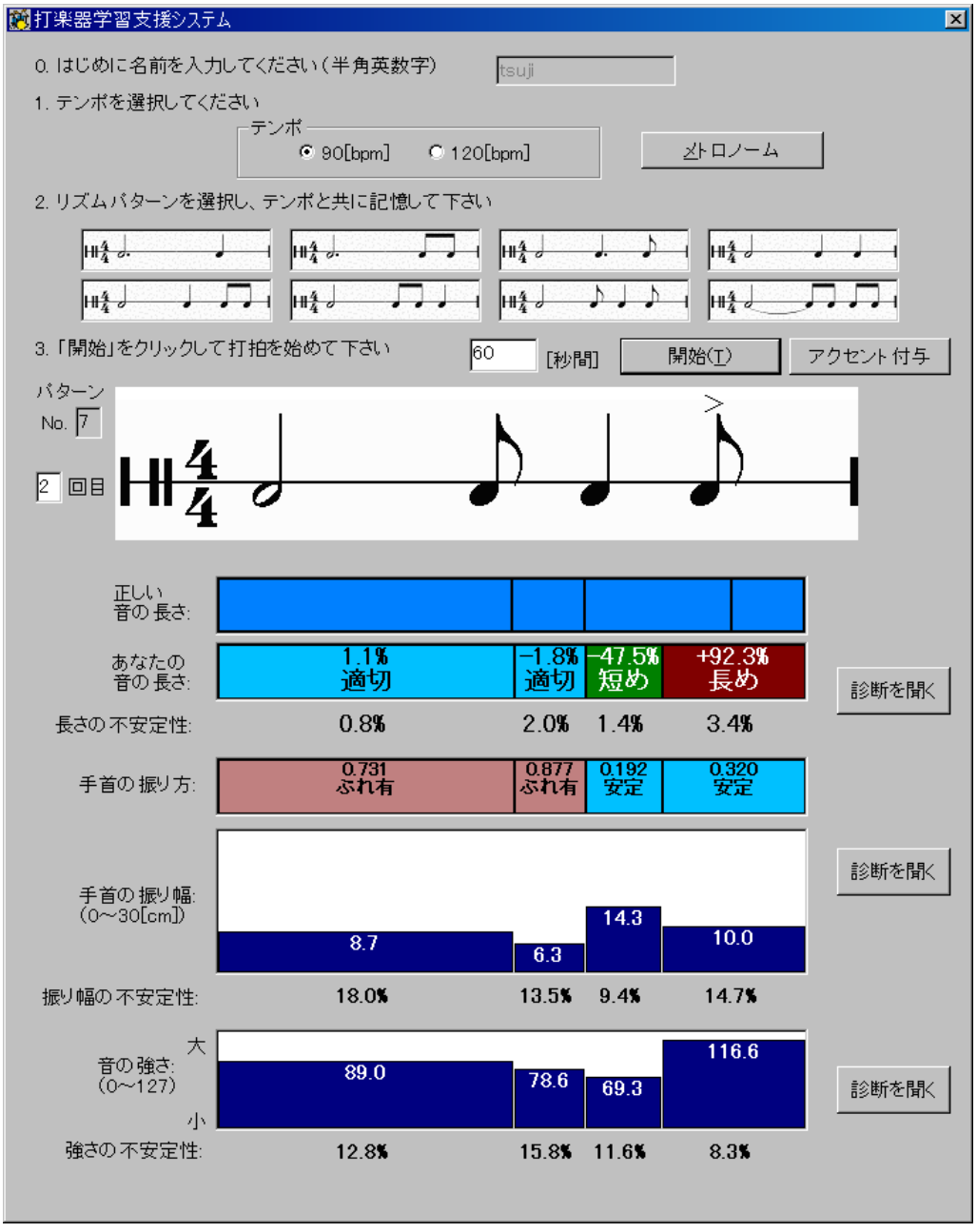


図 9.3: インターフェィス

- 各音符あたりのリズム誤差
  $\overline{T_k}$
- 各音符あたりの振幅 RMS
  $\overline{R_k}$
- 各音符あたりの振幅 RMS の不安定性（標準偏差）
  $SD(R_k)$
- 打拍フォームのパワースペクトル密度関数の 1 次式近似
  $y = af + b$

音長の偏り及び音長の不安定性の抽出においては、先行研究である米川らのアルゴリズムをそのまま採用している [53]。drumSupport クラスの cluster_main 関数でクラスター分析を行ない、その後に偏りを計算している。提示は音符毎に % 単位で表示される。正しい音符の長さを 100% とした時の、ユーザの音符の長さの平均を%で表示する。

具体的には、各音符に対してその音長を音符の 1 小節あたりの個数次元のベクトルにし、それをクラスター分析することにより正しいリズムパターンと同じクラスターとみなされた小節のみをピックアップしている。これにより、打ち損じの演奏データの削除を行なっている。打ち損じの演奏及びその際のストロークフォームデータを削除した後、音長の偏り、音長の不安定性、ストローク誤差、手の振り幅、振り幅の不安定性、打拍の強さ、強さの不安定性を計算する。

ストロークフォームのずれ $\overline{E_k}$ は 6.3.2 節で定義したストローク誤差を、小節単位に拡張した値を用いて計算している。n 小節目の k 番目の音符の際のストロークと、$n-1$ 小節目の k 番目の音符の際のストロークを比較したストローク誤差が E_k^n と定義される。具体的には、E_k^n は以下の式で定義される。

$$E_k^n = \frac{\frac{1}{D_n^k}\int_{t_k^n}^{t_{k+1}^n}\left|\vec{f_k^n}(t) - \vec{f_k^{n-1}}(t')\right|^2 dt}{\frac{1}{D_n^k}\int_{t_k^n}^{t_{k+1}^n}\left|\vec{f_k^n}(t) - \vec{f_k^n}\right|^2 dt}$$
$$(n = 2...N,\ k = 1...3)$$

ここで、N はユーザの演奏の全小節数、$\vec{f_k^n}(t)$ は n 小節 k 番目の音符の際のストロークにおけるマーカの座標を、時刻に関して補間を行なったベクトル関数、t_k^n は n 小節 k 番目の音符を打拍した時刻である。ただし、用いたリズムパターンにおける 1 小節の音符数を $k_{\max}$ とすると、$t_{k_{\max}+1}^n = t_1^{n+1}$。$D_k^n$ は n 小節 k 番目の音符の長さで $D_k^n = t_{k+1}^n - t_k^n$、$\vec{f_k^n}$ は n 小節 k 番目の音符の際のフォームストロークの振幅の平均値、つまり、$\vec{f_k^n} = \frac{1}{D_k^n}\int_{t_k^n}^{t_{k+1}^n}\vec{f_k^n}(t)dt$ である。

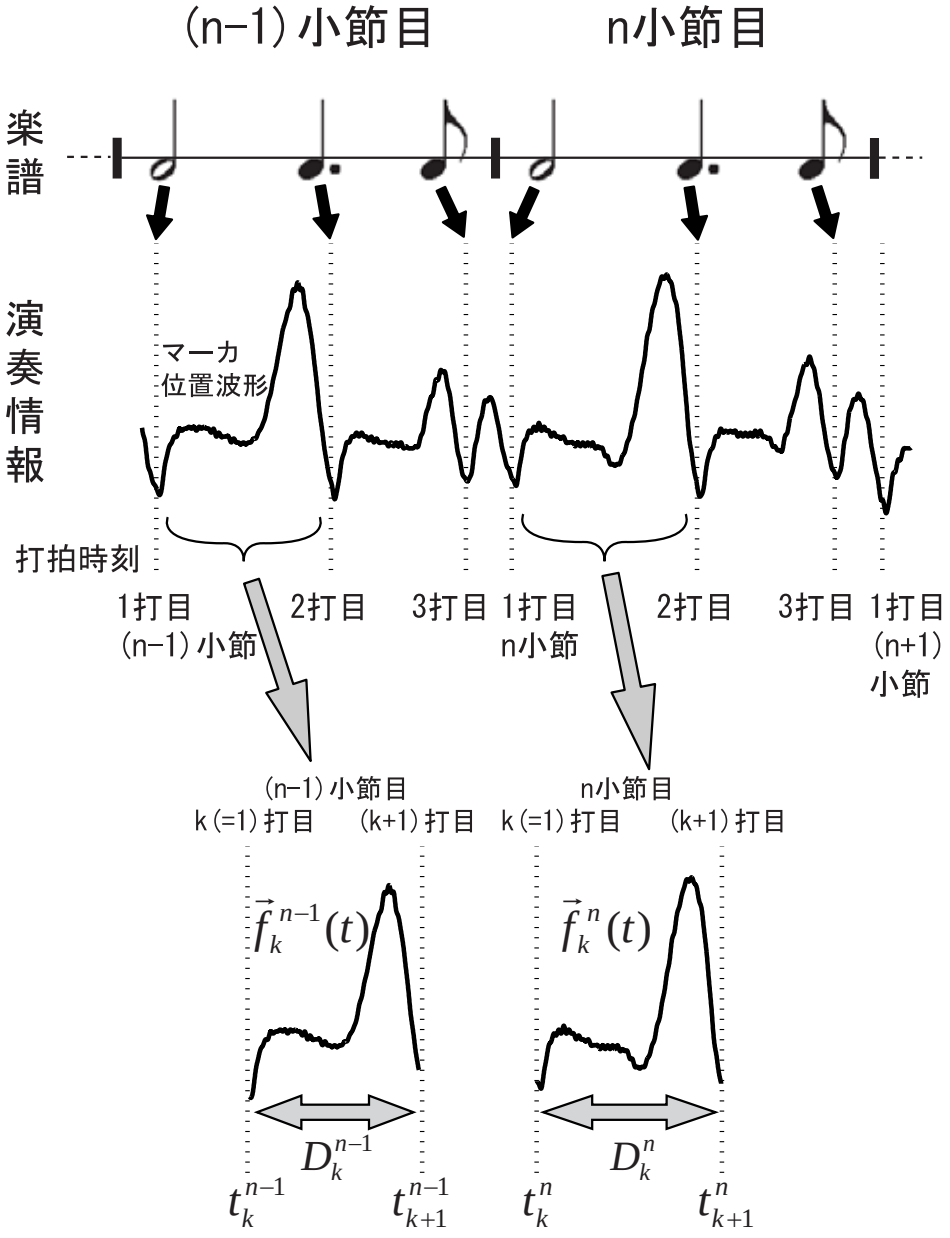

(n−1)小節目
n小節目
楽譜
演奏情報
マーカ
位置波形
打拍時刻
1打目
(n−1)小節
2打目
3打目
1打目
n小節
2打目
3打目
1打目
(n+1)
小節
(n−1)小節目
k(=1)打目
(k+1)打目
n小節目
k(=1)打目
(k+1)打目
$\vec{f}_k^{\,n-1}(t)$
$\vec{f}_k^{\,n}(t)$
D_k^{n-1}
D_k^{n}
t_k^{n-1}
t_{k+1}^{n-1}
t_k^{n}
t_{k+1}^{n}

図 9.4: 特徴量の計算

また、t' は t が $t_k^n \to t_{k+1}^n$ まで動く際に $t_k^{n-1} \to t_{k+1}^{n-1}$ まで動く値であるから、

$$t' = t_k^{k-1} + \frac{t - t_k^n}{t_{k+1}^n - t_k^n} \times (t_{k+1}^{n-1} - t_k^{n-1})$$

と定義できる。この E_k^n を小節数で割った平均値 $\overline{E_k}\left(= \frac{1}{N}\sum_n E_k^n\right)$ を "ストロークのずれ "としてフィードバックしている。この値 $\overline{E_k}$ は、小節の中の k 番目の音符の際のストロークがどのくらいずれたかを表している。

各音符あたりの手の振り幅 $\overline{S_k^{\max}}$ は、次の定義で計算している。n 小節目 k 番目の音符におけるストロークの振幅 S_k^n は以下で定義できる。

$$S_k^n = \max(f_y(t)|_{t=t_k^n \ldots t_{k+1}^n}) - \min(f_y(t)|_{t=t_k^n \ldots t_{k+1}^n})$$
$$(n = 1...N,\ k = 1...3)$$

この S_k^n を小節 n で平均を取った値 $\overline{S_k}(= \frac{1}{N}\sum_n S_k^n)$ は、k 番目の音符における振幅の平均を示している。S_k^n の標準偏差 $SD(S_k)$ は、k 番目の音符における振幅の不安定性を示していると言える。この両者の値をフィードバックとして用いており、不安定性が大きい場合は「振り幅が不安定です」とフィードバックされる。

各音符の打拍の強さ（ベロシティ）$\overline{V_k}$ は各打拍におけるベロシティの値をそのまま用いている。すなわち、n 小節 k 番目のベロシティの値を V_k^n とすると、$\overline{V_k} = \frac{1}{N}\sum_n V_k^n$ である。この k 番目の音符のベロシティ平均 $\overline{V_k}$ と標準偏差 $SD(V_k)$ をフィードバックとして用いている。標準偏差 $SD(V_k)$ は k 番目の音符におけるベロシティの不安定性を示していると言える。不安定性が大きい場合は「音の強さが不安定です」とフィードバックされる。

また、我々が [?] で定義した振幅 RMS 値を小節単位に拡張した値を分析に用いている。すなわち、n 小節目の k 番目の音符の際の振幅 RMS 値 R_k^n は、以下の式で定義される。

$$R_k^n = \sqrt{\frac{1}{D_k^n}\int_{t_k^n}^{t_{k+1}^n} f_y^2(t)dt}$$
$$(n = 1...N,\ k = 1...3)$$

ここで $f_y(t)$ は、$\vec{f_k^n}(t)$ の y 成分のみを抽出した関数である。R_k^n のすべての小節 n における平均値 $\overline{R_k}\left(= \frac{1}{N}\sum_n R_k^n\right)$ 及び標準偏差 $SD(R_k)$ を分析に用いた。

同様に、7.4.1 節で定義したリズム誤差も小節単位に拡張した値を分析に用いている。n 小節 k 番目の音符におけるリズム誤差 T_k^n の値は

$$T_k^n = \frac{\left|(t_{k+1}^n - t_k^n) - (t_{k+1}^{n-1} - t_k^{n-1})\right|}{\frac{(t_{k+1}^n - t_k^n) + (t_{k+1}^{n-1} - t_k^{n-1})}{2}}$$

$$(n = 2...N,\ k = 1...3)$$

と表す事ができる。このすべての小節における平均値 $\overline{T_k}\left(= \frac{1}{N}\sum_n T_k^n\right)$ は、k 番目の音符において全体的にどのくらい短期的なリズムがずれたかを表している。この値は後の分析に用いている。

さらに、7.4.3 節で求めたパワースペクトル密度関数の 1 次式近似 $y = af + b$ も分析の為に抽出した。抽出アルゴリズムは 7.4.3 節と同様である。傾き a と y 切片 b の値を分析に用いた。

9.4　評価実験

本システムを評価するために、被験者実験を行なった。本システムは、大まかに分割すると、“リズムのフィードバック”、“フォームのフィードバック”、“ベロシティのフィードバック” の 3 種類のフィードバックから構成されている。各フィードバックが、打楽器の特別な経験のないユーザに対してどのように作用するのかを調査するために、“リズムフィードバック群”、“フォームフィードバック群”、“ベロシティフィードバック群”、“統制群” の 4 グループで実験を行なった。

9.4.1　方法

実験は次の手続きで行なわれた。

被験者は、始めに楽器経験、持っている CD の枚数などを聞くアンケートに回答させ、打楽器の継続的な練習経験がないことを確認した。その後、打楽器学習支援システムを用いて 1 分間のリズムパターン打拍を 10 回行なわせた。10 回のうち、始めの 2 回は練習で、後の 8 回が本番であると被験者には伝えた。打拍するリズムパターンは、始めの 2 回がパターン 6（図 9.5)、残りの 8 回はパターン 3（図??）を打拍させた。また始めの 2 回では、リズム、フォーム、ベロシティのすべての演奏及びフォームの情報をフィードバックするフルバージョンの打楽器学習システムを用いた。始めの 2 回が終った後、4 つのグループに分け、それぞれ違う種類のフィードバックを返す本システムを用いて 8 回の打拍をさせた。4 つのグループに分ける段階では、演奏結果データの音長の偏りのデータを確認し、リズム能力がグループ間で有意に異ならないようにした。4 つのグループとは、” リズムフィードバック群’、’ フォームフィードバック群”’、” ベロシティフィードバック群”、” 統制群”’ の 4 つである。リズムフィードバック群では、音長の偏り、不安定性の項目のみ、フォームフィードバック群では、ストローク誤差、手首の振り幅、振り幅の不安定性の項目のみ、ベロシティフィードバック群では、打拍の強さ（ベロシティ)、強さの不安定性の項目のみ本システムによりフィードバック

される。統制群ではフィードバックを行なわない本システムを用いた。

10 回の演奏が終った後、本システムの主観評価をアンケートにより行なった。

被験者は打楽器の継続的な練習経験のない大学生・大学院生 24 人である。

また、打楽器未経験者のデータと比較するために、打楽器経験者に対しても同様の実験を行なった。被験者は平均して 1 週間に 5 時間以上の練習を 6 年以上している打楽器経験者 3 名（そのうち一人は打楽器のプロ演奏家）である。手続きは以下である。

始めに楽器経験アンケートに回答させた。その後、フルバージョンの本システムを用いてリズムパターン 6 （図 9.5）の 1 分間打拍を 2 回、練習として行なわせた。その後、アクセント付のリズムパターン 3（図 9.6）の 1 分間打拍を 8 回行なわせ、主観評価アンケートに記入させた。さらにその後、リズムフィードバックバージョンのシステムを用いて 1 分間打拍を 8 回、フォームフィードバックバージョンのシステムを用いて 1 分間打拍を 8 回、ベロシティフィードバックバージョンのシステムを用いて 1 分間打拍を 8 回、フィードバックしないバージョンのシステムを用いて 1 分間打拍を 8 回、合計 32 回の打拍を行なわせた。打拍したリズムパターンはすべてアクセントなしのパターン 3 である（図 9.7）。また、システムを使っていく順番は 3 人でそれぞれ異なるようにした。

9.4.2　結果と考察

各フィードバックグループにおける特徴量の変化の検証

始めに、音長の偏り、偏りの不安定性（標準偏差）、ストローク誤差の平均 $\overline{E}$, 振幅 RMS 値の平均 $\overline{R}$、RMS 値の標準偏差 $SD(R)$、手の振り幅の最大値の平均 $\overline{S}$、手の振り幅の最大値の不安定性（標準偏差）$SD(S)$、リズム誤差の平均 $\overline{T}$、打拍の強さの平均 $\overline{V}$、強さの不安定性 $SD(V)$ の各特徴量に対して、24 名の被験者の前半 4 回及び後半 4 回の打拍施行における平均値を求めた。そして t 検定によりその平均値を 4 つの群において比較した。

その結果、フィードバックなし群、フォームフィードバック群、ベロシティフィードバック群においては、前半 4 回と後半 4 回の上記の特徴量の平均値において有意な改善傾向は確認できなかった（$p > .10$）が、リズムフィードバック群においては、

- 音長の偏りの不安定性（$t(46) = 1.742, p < .10$）
- 振幅 RMS 値の平均 $\overline{R}$（$t(46) = 2.035, p < .05$）
- RMS 値の標準偏差 $SD(R)$（$t(46) = 2.536, p < .05$）
- 手の振り幅の最大値の平均 $\overline{S}$（$t(46) = 2.253, p < .05$）

図 9.5: リズムパターン 6

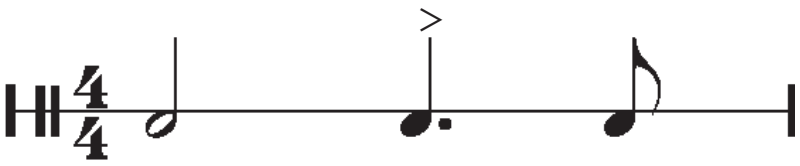


図 9.6: リズムパターン 3-アクセント付

図 9.7: リズムパターン 3-アクセントなし

- 手の振り幅の最大値の不安定性 $SD(S)$（$t(46) = 2.727, p < .01$）
- リズム誤差の平均 $\overline{T}$（$t(46) = 1.736, p < .10$）
- 打拍の強さの不安定性 $SD(V)$（$t(46) = 1.935, p < .10$）

の特徴量において、前半 4 回、後半 4 回の平均値間で有意差及び有意傾向が確認できた（図 9.8）。これより、ユーザの演奏情報から音長の偏りをフィードバックすると、短期的な音符の長さのずれ（リズム誤差）が小さくなり、分布も小さくなり（音長の偏りの不安定性）、打拍の強さも一定に近くなり（打拍の強さの不安定性）、ストローク自体は小さく、コンパクトになる傾向があることが分かった。

4 つの群を比較して考えると、演奏したリズムの情報をフィードバックする事は、ストロークフォームや打拍した強さの情報をフィードバックするよりも、打楽器未経験者の演奏に即座に影響しやすい傾向があると考えられる。

反対に、ストロークフォームや打拍の強さの情報は、打楽器未経験者に直接「フォームがずれました」、「打拍の強さが不安定です」とフィードバックを与えても、どのように改善したら良いか分からない、もしくはすぐには制御・改善でき

ない可能性があることが分かった。本システムの長期的な使用に関しては今後の課題である。

打楽器経験及びフィードバックグループ間の特徴量の違いの検証

さらに、打楽器経験者及び未経験者グループと、フィードバックグループ間において、各特徴量を従属変数とした二元配置の分散分析を行なった。その結果、フィードバックグループ間では有意な違いが確認できなかったが、打楽器経験者及び未経験者間では、音長の不安定性（有意水準 $p < .01$）、$\overline{T}$（$p < .01$）、$\overline{V}$（$p < .001$）、一次近似式の係数 a（$p < .05$）、b（$p < .05$）において有意な違いが確認できた。このことより、リズムパターンの打拍においても打楽器経験者は、未経験者と比べて

- 安定した音長で演奏できる
- 短期的なリズム誤差が小さい
- 打拍フォームの高周波数成分の割合が大きい

可能性が示された。

特徴量間の相関の検証

続いて、各グループにおける、ストローク誤差の平均 $\overline{E}$ の特徴量とリズム関連の特徴量間の相関について調べた。また、振幅 RMS 値の平均 $\overline{R}$ と振幅の不安定性（標準偏差）$sd(S)$ とベロシティの不安定性（標準偏差）$sd(V)$ の特徴量間についても相関を調べた。

その結果を図 9.9 と図 9.10 に示す。

この結果を見ると、打楽器経験者は腕の振幅と打拍の強さとの関わりが大きく、また逆に打楽器未経験者の方が打拍フォーム誤差と演奏のずれ（音長の偏り、音長の不安定性、リズム誤差 $\overline{T}$）に関わりが強い結果が得られた。打楽器経験者の方が打拍の強さを制御する際に振り幅を意識している可能性があると言える。この点は 7.5.4 節の結果と一致している。

ストローク誤差とリズム誤差の関係については 7.5.2 節の結果とは一致せず、どちらも関連性が薄い結果となった。この理由は、7.5.2 節におけるリズム誤差 $\overline{T}$ とストローク誤差 $\overline{E}$ の定義が一定打拍における値だったのに対し、今回の定義は小節単位に拡張した値であるので、相関が出にくかったと考えられる。

9.5　ユーザによるシステムの主観評価

打楽器経験者による主観評価

リズム学習支援システムを使用した打楽器の経験が充分にある 3 人の被験者に対して、システムに対する評価を SD 法を用いて行なった。各質問項目に対して

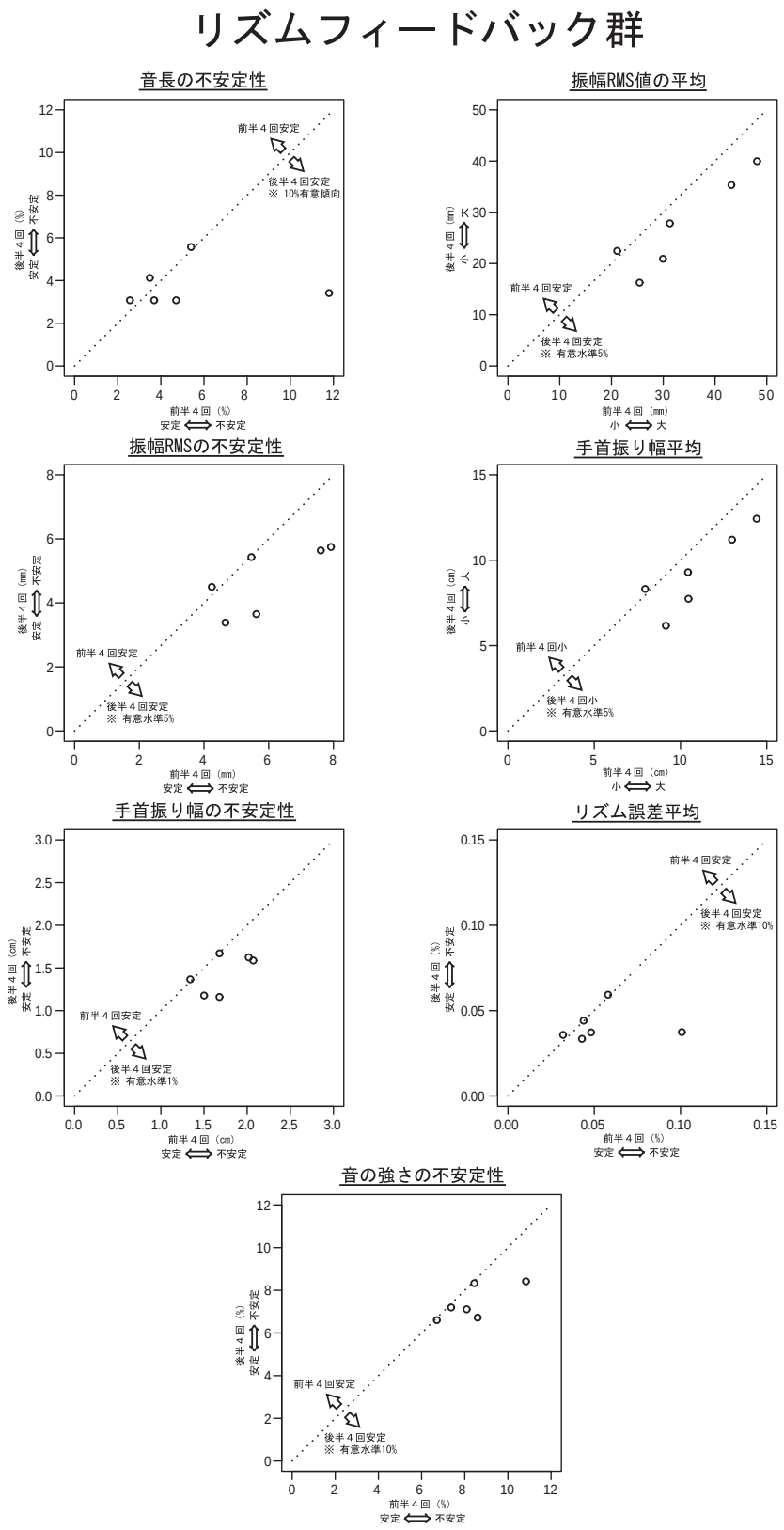


図 9.8: 各特徴量の前半 4 回と後半 4 回の平均値の比較

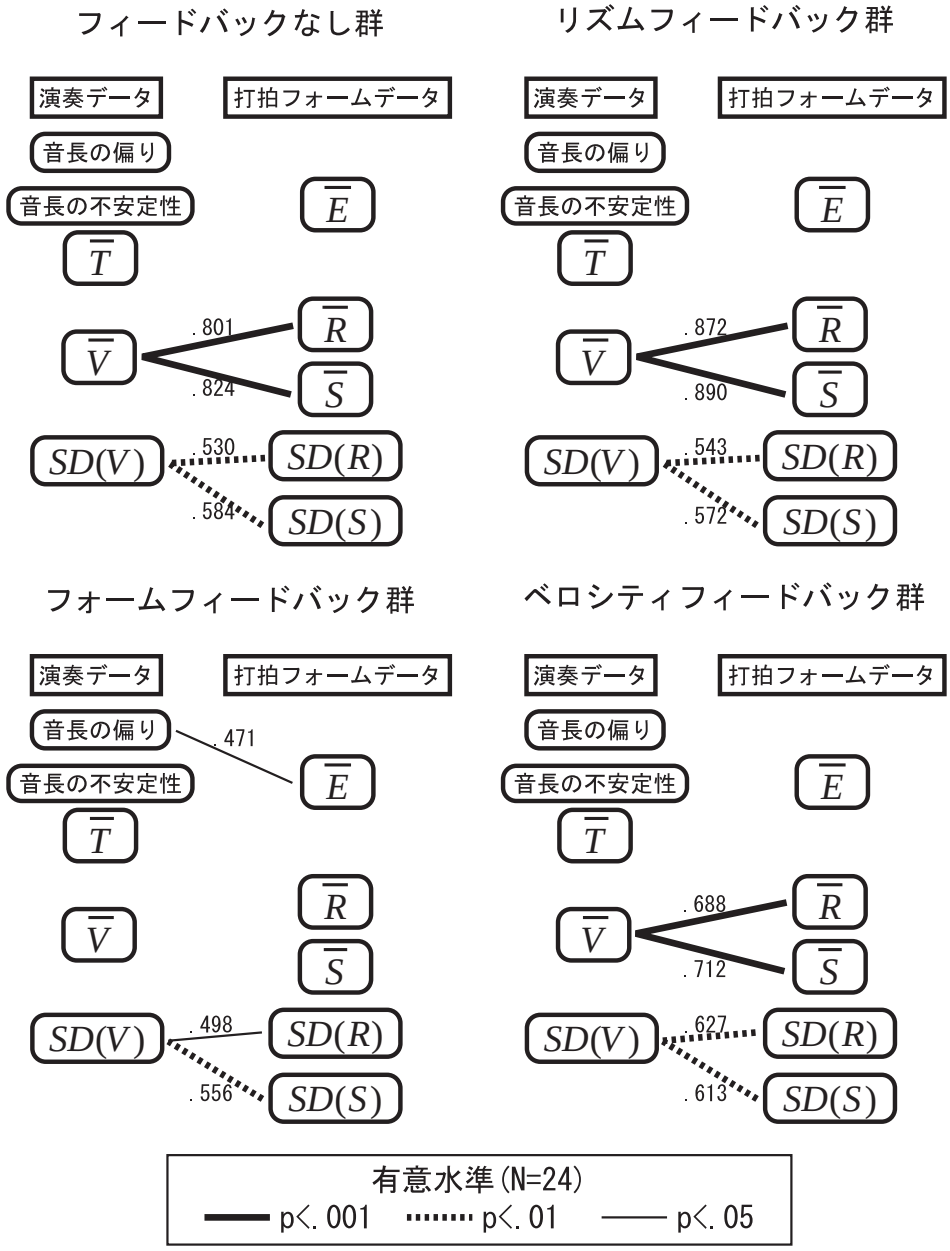


図 9.9: 打楽器経験者の特徴量間の相関

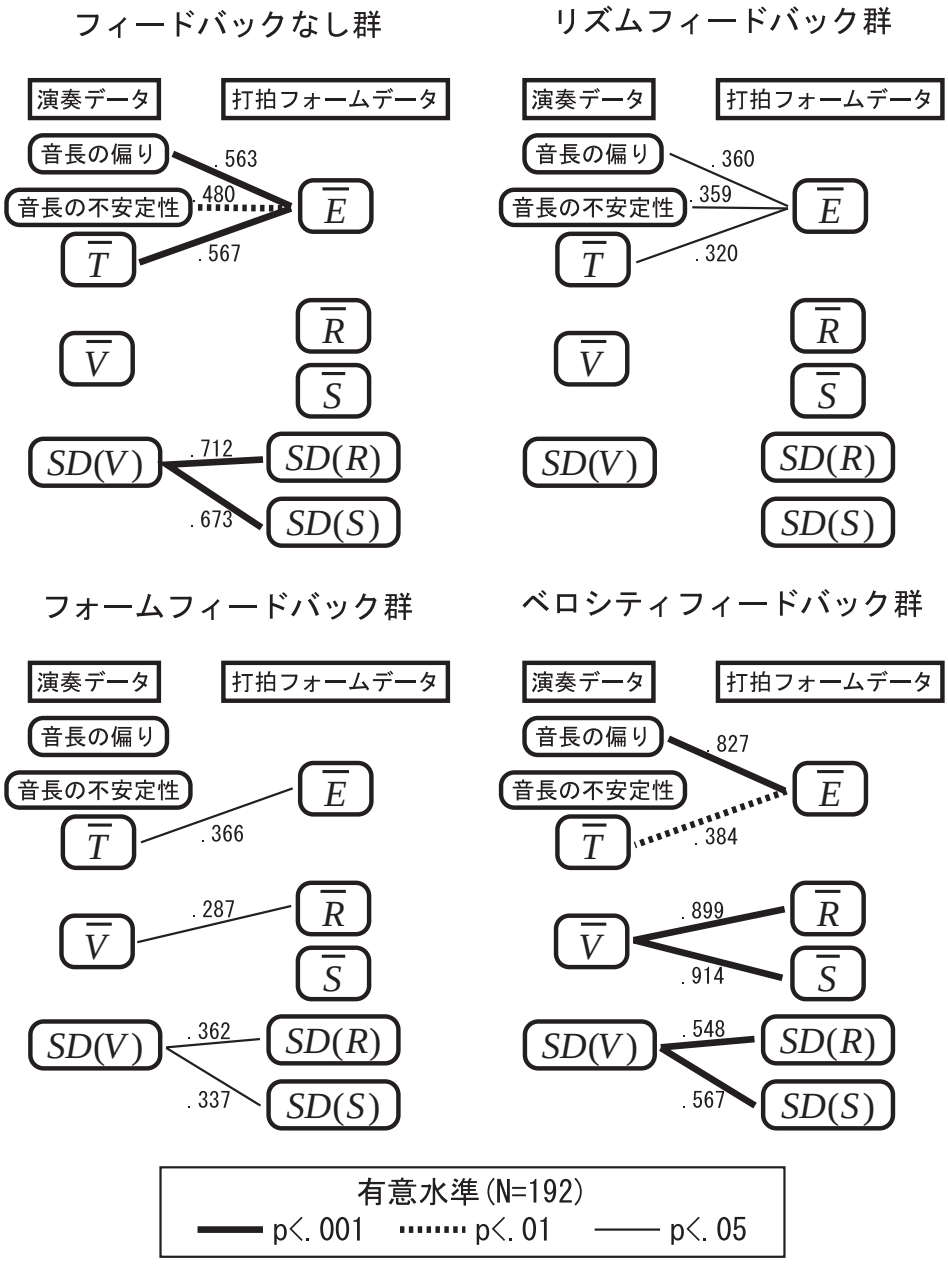


図 9.10: 打楽器未経験者の特徴量間の相関

4 段階で評価を行なった。“4” は「そう思う」、“3” は「どちらかといえばそう思う」、“2” は「どちらかといえばそう思わない」、“1” は「そう思わない」とした。評価が高い場合に、常に評価値が”4” になるように尺度の方向を定めた。質問内容は、システム画面の見やすさ、使いやすさと、診断結果提示の分かりやすさ、妥当性などである。打楽器経験者はリズム、ストロークフォーム、ベロシティに関するフィードバックをするフルバージョンのシステムを評価した。従って、アンケートの質問項目はそれらのフィードバックの分かりやすさ、妥当性を問う質問をすべて含んでいる。

回答人数が少ないため、因子分析で因子を抽出することはできなかった。表 9.1 に各質問の平均点の結果を示す。

表 9.1 を見ると、

- このシステムのアクセント付与機能は練習になる
- このシステムはリズム感を良くするのに役立つ
- このシステムはアクセントなどの、叩く強さを制御する練習をするのに役立つ

の項目がいずれも 3 人の被験者がすべて 4 の「そう思う」を選択しており、本システムを用いてリズム及びアクセントの練習をする場合に有効である可能性が示唆されたと言える。

また、平均点が低い設問としては、以下の設問があった。

- 画面の言葉使いは適切だった
- このシステムは良いストロークフォームを習得するのに役立つ
- 音声による診断結果の提示は全体的に分かりやすかった

これらについては今後の課題である。今回開発した本システムでは、あくまでユーザの基の演奏結果をベースに、かつ忠実な形でフィードバックしており、演奏や打拍フォームに対する教示をフィードバックしているのではない。従って、「このシステムは良いストロークフォームを習得するのに役立つ」の質問における得点が低いのは妥当と考えられる。

また、「このシステムを打楽器の練習に使ってみたい」の質問には、3 人中 2 人の被験者が「そう思う」と回答しているのに対して、1 人が「そう思わない」と回答した。「そう思わない」と回答した打楽器経験のある被験者は、その理由として「すぐ飽きてしまう。良い結果の場合に新しい楽譜で練習できるなど、もっとゲーム性があると良い。」という点を上げている。この点については今後の課題である。

表 9.1: 質問の平均点及びチャンスレベルとの比較-打楽器経験者

質問項目	平均点	
画面の提示は見やすかった。	4	+
提示された楽譜は見やすかった。	4	+
このシステムのアクセント付与機能は練習になる。	4	+
このシステムはリズム感を良くするのに役立つ。	4	+
このシステムはアクセントなどの、叩く強さを制御する練習をするのに役立つ。	4	+
90bpm のテンポは演奏しやすかった。	3.67	
音声提示のスピードは適切だった。	3.67	
ドラムパッドとスティックを用いた演奏は容易だった	3.67	
音の長さの診断結果の提示は分かりやすかった。	3.67	
音声による診断結果の提示は全体的に妥当であった。	3.67	
診断結果の画面表示は全体的に妥当であった。	3.67	
音の長さの診断結果の提示は妥当であった。	3.67	
右手の振り方の診断結果の提示は妥当であった。	3.67	
右手の振り幅の診断結果の提示は妥当であった。	3.67	
音の強さの診断結果の提示は妥当であった。	3.67	
画面の色使いは適切だった。	3.33	
音の強さの診断結果の提示は分かりやすかった。	3.33	
このシステムは安定した音量で演奏する能力を習得するのに役立つ。	3.33	
診断結果の画面表示は全体的に分かりやすかった。	3.33	
60 秒間の演奏時間はやりやすかった。	3.00	
手袋を装着しての演奏は、付けてない場合と比べて特に気にならなかった。	3.00	
このシステムを打楽器の練習に使ってみたい。	3.00	
右手の振り方の診断結果の提示は分かりやすかった。	3.00	
右手の振り幅の診断結果の提示は分かりやすかった。	3.00	
画面の提示は分かりやすかった	2.67	
このシステムは叩く際の腕の振り方を矯正するのに役立つ。	2.67	
このシステムは安定したストロークフォームを習得するのに役立つ。	2.67	
画面の言葉使いは適切だった。	2.33	
このシステムは良いストロークフォームを習得するのに役立つ。	2.33	
音声による診断結果の提示は全体的に分かりやすかった。	2.33	

+…被験者の回答が 3 人とも”4”

打楽器未経験者による主観評価

また、同様にリズム学習支援システムを使用した打楽器未経験者のユーザによる主観評価も行なった。25 人の被験者に対して、システムに対する評価を 4 段階回答の SD 法を用いて行なった点は経験者の場合と同様である。

経験者の時と異なるのは質問数である。未経験者の質問内容もシステム画面の見やすさ、使いやすさ及び診断結果提示の分かりやすさ、妥当性などであるが、打楽器未経験者は診断結果として、リズム、ストロークフォーム、ベロシティのどれか 1 種類のフィードバックのみ、もしくはフィードバックなしの打楽器学習支援システムを用いた。従って質問項目も、フィードバック内容の分かりやすさ及び妥当性を問うものは、用いたシステムのフィードバックの種類に合うもののみを用いた。

分析として始めにすべての打楽器未経験者に対して共通質問項目の因子分析を行なったが、因子は抽出できなかった。各質問項目に対する平均点を表 9.2 に示す。

この結果より未経験者のユーザは、本システムを用いる事により、リズム感、安定した音量、安定したストロークフォームが身につく事を期待できた事が分かる。また、診断結果の提示が妥当であり、分かりやすかったと回答している。しかし、その一方で

- 「どのようなストロークフォームで演奏したら良いか分からない。」
- 「フォームがぶれている、と言われてもどうしたら良いか分からない」
- 「数値で言われてもどうしたら良いか分からない」

という意見もあった。今回開発した打楽器学習支援システムは、どのような腕の振り方で演奏すればいいか、どのように演奏音量を調節したらよいか、といった演奏方法自体の教示は行なっていない。本システムは、腕の振り方がどのくらいぶれたか、腕の振り幅がどのくらいであったか、音の長さ、強さがどのくらいであったかのみをフィードバックするので、腕の振り方などの演奏方法の知識を持たない打楽器未経験者には少し敷居が高かった可能性がある。従って、今回開発した本システムは、ストロークフォームの知識を持つユーザに対して、より向いているシステムである可能性が示されたと言える。その点は今後の課題と考えられる。

9.6　9章のまとめ

本章では、前章の結果を基に、演奏データ及び打拍フォームデータより以下の特徴量を抽出し、フィードバックを可能とする打楽器学習支援システムを開発した。

- 1 小節あたりの音長の偏り

表 9.2: 質問の平均点及びチャンスレベルとの比較-打楽器未経験者

質問項目	平均点
このシステムは安定した音量で演奏する能力を習得するのに役立つ。	3.61
手袋を装着しての演奏は、付けてない場合と比べて特に気にならなかった。	3.50
画面の提示は見やすかった。	3.46
提示された楽譜は見やすかった。	3.42
画面の提示は分かりやすかった	3.33
画面の言葉使いは適切だった。	3.29
このシステムは安定したストロークフォームを習得するのに役立つ。	3.22
画面の色使いは適切だった。	3.13
診断結果の画面表示は全体的に妥当であった。	3.11
音声提示のスピードは適切だった。	3.04
90bpm のテンポは演奏しやすかった。	3.00
診断結果の画面表示は全体的に分かりやすかった。	3.00
音声による診断結果の提示は全体的に妥当であった。	2.89
音の長さの診断結果の提示は分かりやすかった。	3.50
このシステムはリズム感を良くするのに役立つ。	3.17
音の強さの診断結果の提示は妥当であった。	3.17
右手の振り方の診断結果の提示は分かりやすかった。	3.00
音の強さの診断結果の提示は分かりやすかった。	3.00
右手の振り方の診断結果の提示は妥当であった。	3.00
右手の振り幅の診断結果の提示は妥当であった。	3.00
このシステムはアクセントなどの、叩く強さを制御する練習をするのに役立つ。	2.94
音の長さの診断結果の提示は妥当であった。	2.83
右手の振り幅の診断結果の提示は分かりやすかった。	2.83
このシステムは叩く際の腕の振り方を矯正するのに役立つ。	2.72
60 秒間の演奏時間はやりやすかった。	2.63
音声による診断結果の提示は全体的に分かりやすかった。	2.61
ドラムパッドとスティックを用いた演奏は容易だった	2.54
このシステムは良いストロークフォームを習得するのに役立つ。	2.33

- 1 小節あたりの音長の不安定性（標準偏差）
- 各音符あたりのストロークフォームのずれ
  $\overline{E_k}$, k：音符数
- 各音符あたりの手の振り幅
  $\overline{S_k}$
- 各音符あたりの手の振り幅の不安定性（標準偏差）
  $SD(S_k)$
- 各音符の打拍の強さ（ベロシティ）
  $\overline{V_k}$
- 各音符の打拍の強さの不安定性（標準偏差）
  $SD(V_k)$

打楽器経験者 3 名、未経験者 24 名に対して、リズムパターン打拍を課題とした被験者実験を行ない、各フィードバックの効果を調査した。その結果、リズムフィードバック群において

- 音長のばらつきが小さくなる
- 手首の振り幅のばらつきが小さくなる
- ベロシティのばらつきが小さくなる

有意な効果が確認できた。

各特徴量を従属変数とした、打楽器経験群、フィードバック群間の二元配置の分散分析を行なった。これより、打楽器経験者は未経験者と比べて

- 安定した音量で演奏できる
- 短期的なリズムのずれが小さい
- 打拍フォームの高周波数成分の割合が大きい

傾向が見られた。

さらに、各グループにおいて、ストローク誤差 $\overline{E}$ とリズム関連の特徴量、振幅 RMS 値 $\overline{R}, SD(R)$ または振幅 $\overline{S}, SD(S)$ とベロシティの相関を求めた。

その結果、打楽器経験者は手首の振幅と打拍の強さ（ベロシティ）との関わりが大きく、また逆に打楽器未経験者の方が打拍ストローク誤差とリズム関連の特徴量に関わりの強い結果が得られた。

また、打楽器経験者 3 名に対して本システムの主観評価を行なわせた所、

- 「このシステムはリズム感を良くするのに役立つ」
- 「 このシステムのアクセント付与機能は練習になる」

に好意的な回答が確認できた。
　以上より、本システムの有効性が示された。

10章

結論

10.1　本研究で得られた結果

(1) ニューラルネットワークを用いた楽曲のジャンル推定に関して

- 音楽経験が豊富な被験者 4 名を対象とした、インタビュー方式による予備実験を行った。その結果、『ジャンルを推定する要因は、部分的な情報であることが多い』、『ジャンルを推定する要因である部分的な情報は、曖昧な情報が多い』などの知見が得られた。

- ニューラルネットワークを用いた楽曲のジャンル推定システムを開発した。推定には、「楽器と音色」、「リズムの分布」、「音名の分布」の 3 種類の特徴量を用いた。

- ニューラルネットワークに 4 ジャンル 120 曲の楽曲を学習させることにより、100%の同定ができた特徴量として、「楽器と音色」、「リズムの分布」、「音名の分布」の 3 つまたは「楽器と音色」、「リズムの分布」の 2 つのみを用いた場合にニューラルネットワークの学習が完了した。これより、「楽器の種類」、「リズム」を用いる事によりジャンルを推定できる可能性が示された。

(2) ニューラルネットワークと被験者によるジャンル推定比較に関して

- 特別な音楽経験を持たない 25 人の被験者に対して、4 ジャンル合計 8 曲のスタンダード MIDI ファイルを聴取させ、ジャンル推定させる実験を行なった。その結果、66.5%の正解率が得られた。

- 被験者がジャンルを推定した要因に対して有意差の検定をした所、ハードロックとジャズのジャンルにおいて、「楽器と音色」の要因を不正解者より

も正解者の方が有意に重要視していた事が分かった（どちらも $p < .05$）。

- 被験者がジャンルを推定する際の確信度に対して、4 ジャンル間の分散分析を行なった所、以下の両者間で有意な違いが確認できた。
  - ジャズ-演歌（$p < .001$）
  - ヘヴィメタル-演歌（$p < .01$）

  ジャズやヘヴィメタルと回答するよりも演歌と回答した場合の方が確信度が大きい事が分かった。

- 被験者実験で用いたのと同様の 8 曲を用い、Knife-Edge 法によりニューラルネットワークに対して未知曲としての推定実験を行なった。その結果、8 曲すべてに対して正しい判別ができた。

(3) 学習したニューラルネットワークの中間層の分析に関して

- 学習したニューラルネットワークの中間層の 5 個のノードに対して、ネーミングができた。
  - 「ハードな音色因子」
  - 「シンセサイザ音色因子」
  - 「ジャズ・アコースティック因子」
  - 「リズム傾向因子」
  - 「エレキ楽器とリズム傾向因子」

  この結果より、ジャンルを用いた支援システムの可能性が示された。

(4) 打拍フォームに対する教示の効果に関して

- 打楽器未経験者に対して、打拍フォームに関する教示を行なった後にリズムパターン打拍を行なう実験を行ない、教示の効果を検証した。

- 教示あり、なし群の間には音長の偏り、偏りの不安定性、リズム誤差、ストローク誤差、振り幅の不安定性で有意差が確認できた。

- 教示あり群の前後半の特徴量を比較した所、振り幅の不安定性のみ有意差が確認できた。

(5) 打楽器経験者、未経験者間の打拍フォームの違いについて

- 打楽器経験者・未経験者 14 名に対して、一定間隔打拍を行なわせ、右手首の位置情報を測定した。
- 得られた演奏時の手首の位置の時間波形から、「ストローク誤差」、「パワースペクトルピーク比」、「振幅 RMS 値」の 3 種類の特徴量を抽出した。
- 打楽器経験者と未経験者によって演奏の際の右腕のストローク波形に違いがあるのかを t 検定によって分析した結果、3 種類のテンポごとに特徴量において、経験者と未経験者間で有意な差が見られた。
- 被験者から得られた特徴量を独立変数とした、ステップワイズ法による判別分析を行ない、経験者グループと未経験者グループ間の線形的な判別を行なった結果、3 種類のテンポにおいて、90.0%、95.0%、78.6% の判別率が得られた。
- 被験者から得られた特徴量よりクラスター分析を行なった結果、5 つのクラスターを得た。このクラスターは同じ被験者はほぼ同じクラスターに属していた。

(6) 一定打拍時の演奏データと打拍フォームデータの関連について

- 一定打拍時の打楽器演奏時の手首の位置情報と、打拍時刻、打拍の強さ等の演奏情報の同時取得を行なった。
- 打楽器経験者及び未経験者間で特徴量の違いを調べた所、$\overline{R}$,$SD(R)$,$\overline{T}$,$SD(V)$,a,b に関して、有意水準 1%の有意差が見られた。
- ストローク誤差とリズム誤差に関して散布図を調べ、相関係数を求めた結果、打楽器経験者、未経験者共に軽い正の相関が見られた。
- 提示テンポ及び音量群間における、振幅 RMS 値とベロシティの平均、標準偏差、psd 関数の一次近似式の係数を従属変数とした二元配置の分散分析を行なった。その結果、打楽器未経験者は提示された音量だけでなくテンポが異なる場合にも打拍フォームに影響を受けるのに対し、経験者は提示テンポが異なっても打拍フォームには影響を受けないことが分かった。
- 振幅 RMS 値とベロシティの各平均値に関して、相関係数を求めた所、打楽器未経験者群よりも、打楽器経験者群においてやや強い正の相関が見られた。これは打楽器教則本における「打楽器経験者は、振幅を調節することによりベロシティを調節している」知見と一致する。

(7) 打楽器演奏能力とジャンル推定能力の関連性について

- 打楽器経験者と未経験者において、打楽器演奏能力とジャンル推定正解率とを比較した。
- 打楽器経験者 3 人と未経験者 5 人に対し、打楽器未経験者の平均推定率が 18.0%だったのに対し、打楽器経験者は 65.0%であり、有意な差が確認できた（$t(6) = -7.195, p < .001$）。
- 演奏能力としてリズム誤差を比較した結果、平均値$\overline{T}$において、有意に打楽器経験者の方が小さな値を取る事が分かった（それぞれ$t(70) = 2.799, p < .01$）。
- 打楽器経験者と未経験者では、打楽器パートのジャンル推定能力及びリズム安定性共にに有意に異なることが分かった。これより両者には何らかの関連性があると期待できる。

(8) 学習支援への応用に関して

- 演奏データ及び打拍フォームデータより特徴量を抽出し、“リズム”、“打拍フォーム”、” ベロシティ” に関するフィードバックを可能とする打楽器学習支援システムを開発した。
- 打楽器経験者 3 名、未経験者 24 名に対して、リズムパターン打拍を課題とした被験者実験を行ない、各フィードバックの効果を調査した。その結果、リズムフィードバック群において
  - 音長のばらつきが小さくなる
  - 手首の振り幅のばらつきが小さくなる
  - ベロシティのばらつきが小さくなる

  有意な効果が確認できた。
- 各特徴量を従属変数とした、打楽器経験群、フィードバック群間の二元配置の分散分析を行なった。これより、打楽器経験者は未経験者と比べて
  - 安定した音量で演奏できる
  - 短期的なリズムのずれが小さい
  - 打拍フォームの高周波数成分の割合が大きい

  傾向が見られた。

- 各グループにおいて、ストローク誤差 $\overline{E}$ とリズム関連の特徴量、振幅 RMS 値 $\overline{R}, SD(R)$ または振幅 $\overline{S}, SD(S)$ とベロシティの相関を求めた。その結果、打楽器経験者は手首の振幅と打拍の強さ（ベロシティ）との関わりが大きく、また逆に打楽器未経験者の方が打拍ストローク誤差とリズム関連の特徴量に関わりの強い結果が得られた。
- 打楽器経験者 3 名に対して本システムの主観評価を行なわせた所、
  - 「このシステムはリズム感を良くするのに役立つ」
  - 「 このシステムのアクセント付与機能は練習になる」

  に好意的な回答が確認できた。
- 以上より、本システムの有効性が示された。

10.2　今後の課題

本研究の今後の課題としては、さまざま考えられるが、列挙すれば次のようになる。

- ジャンルによる学習支援の実現
- 演奏評価における発展
  - アクセント付楽譜における演奏トレーニング
  - MIDI ⇒ アコースティックドラムの生音の評価
- さらなる身体情報と演奏情報の同時測定
  - 複数マーカ
  - 筋電位
  - 呼吸
- 音楽指導者の教示と評価指標との関連付け

本研究の 5 章〜9 章では、正確なリズムで演奏できる能力を実現することを研究対象にし、その支援手法を提案してきた。しかし、人間の時間分解能には限界があり、演奏のリズム正確度は限りなく追求すべきものでないことは明らかである。また、正確なリズムで演奏できることは、奏者が自分の演奏を制御できることの指標にはなるものの、それによって、いわゆる「良い演奏」ができるとは限らない。そこで、本研究の発展として、「良い演奏」について検討する必要がある。

一般に「演奏の良し悪し」は、聴取者の好みによって、ジャンルによって、また時代によっても評価が変り得るものであり、普遍性のある尺度が見出せるかどうかをまず検討する必要がある。このような人間の感性に関わる問題に取り組む方法として、次のような手段が考えられる。

- 演奏のゆらぎの測定
- 聴取者（専門家や非専門家）による主観評価
- 聴取者の脳波による客観評価

謝辞

本研究を進めるにあたって、常に熱心に御指導を賜わりました，本学教育工学開発センター 西方敦博 助教授 に深く感謝いたします．現在までに得た研究に関する方法や考え方，知識は，西方 助教授 の徹底的な御指導によるところが大変大きく，感謝しております．

特に機械学習に関連した研究を進める上で、研究というものについて一つ一つ教えて下さり，御多忙な中でも熱心に御指導を賜わりました，教育工学開発センター 赤堀侃司 教授 に心から感謝致します．

研究を進めるうえで，適切なご助言と御指導を賜わり，いつも身近に接して下さった，教育工学開発センター 中山 実 助教授に心から感謝致します．

御多忙の中、本研究のデザインに関して適切なアドバイスを頂きました、本学大学院 社会理工学研究科 人間行動システム専攻 中川正宣 教授に心から感謝致します．

また， 論文審査員として御指導いただきました 赤堀侃司 教授，中川正宣 教授，西原明法 教授，中山 実 助教授，に心より感謝いたします．

研究を進めるうえでの快適な研究環境と適切な御指導を賜わりました，教育工学開発センターの 吉原祐貴 技官並びに 西郷義則 元技官に心から感謝致します．

常に快適な計算機環境を整えて下さり，また，迅速で適切なアドバイスをして下さいました，教育工学開発センター の 青柳貴洋 助手に心から感謝いたします．

情報教育的な観点の御指導を頂くと共に、常に暖かく見守って下さいました、東京工業大学名誉教授 国立教育政策研究所 教育研究情報センター長 清水康敬 氏、同研究所 同センター研究員 榎本聡 氏に心より感謝致します。

本研究を進めるにあたって、修士課程時代に多大な御指導及びアドバイスを頂きました、東洋英和女学院大学 柳沢昌義 助教授、大東文化大学 杉村和枝 講師、目白大学 藤谷哲 講師に深く感謝致します。

本研究を進める上で、先行研究者として特に技術的な部分において適切かつ多大なご支援、アドバイスをして下さいました、人間行動システム専攻 OD、現在有限会社 HUB コンサルティング CEO 米川孝宏 氏に感謝致します。米川氏の先行研究及び支援が無ければ、本研究は存在し得なかったかもしれません。ここに心より感謝致します。

本研究を進める上で、音楽情報処理的観点から適切なアドバイスを頂きました、元東京工業大学大学院情報理工学研究科助手、現在株式会社 DUO システムズ 鈴木泰山 氏に心より感謝致します。

時に被験者としてなど、公私に渡り多大なご協力をして下さいました、人間行動システム専攻博士課程 森本容介 氏、本専攻 OM、現在株式会社日立製作所 加藤俊幸 氏、同じく本専攻 OM、現在株式会社サンマイクロシステムズ 鈴木尚志 氏に深く感謝致します。

同様に被験者としてもご協力下さり、快適な研究環境を提供して下さいました中山・西方・室田研究室の皆様に心から感謝致します．また、実験に御協力頂いたすべての被験者の皆様に心から感謝致します。

最後に，博士課程に進学することを許して下さった両親に心から感謝致します．

本研究に関する報告

投稿論文

(1) 辻 靖彦，赤堀侃司，“ニューラルネットワークを用いた楽曲のジャンル推定とその教育利用”，日本教育工学雑誌 Vol.24 No.3 pp.183-191 (2000.12)

…… 2 章、3 章、4 章

(2) 辻 靖彦、西方敦博、“統計処理を用いた打楽器のフォーム分析”, 教育システム情報学会誌 Vol.20 No.3 pp.274-283 (2003.07)

…… 6 章

(3) 辻 靖彦、西方敦博、“リズムと打拍フォームの同時測定に基づく打楽器の演奏分析”, 電子情報通信学会誌
(近日投稿予定)

…… 7 章

(4) 辻 靖彦、西方敦博、“リズムと打拍フォームに基づく打楽器学習支援システムの開発と評価”, 電子情報通信学会誌
(近日投稿予定)

…… 5 章、9 章

国際会議

(1) Yasuhiko TSUJI, Kanji AKAHORI, Atsuhiro NISHIKATA, “The Estimation of Music Genre Using Neural Network and Its Educational Use”, Proceeding of International Conference on Computers in Education(ICCE), pp.158-162, Taiwan (2000.11)

…… 2 章，3 章，4 章

学会における口頭発表

(1) 辻 靖彦、赤堀侃司，“ニューラルネットワークを用いた楽曲のジャンル推定とその教育利用” The Estimation of Music Genre Using Neural Network and Its Educational Use，日本教育工学会 第 15 回全国大会，2pK03 (1999.10)

(2) 辻 靖彦、赤堀侃司，西方敦博，“スタンダード MIDI ファイルを用いたニューラルネットワークによる楽曲のジャンル推定” The Estimation of Music Genre of Standart MIDI Files Using Neural Network，JCET 教育工学関連学協会第 6 回全国大会、12A19a4 (2000.11)

(3) 辻 靖彦，西方敦博，“打楽器のフォーム改善支援システムの開発” Development of Support System for Drumming Form Improvement，電子情報通信学会総合大会，D-15-4 (2002.3)

(4) 辻 靖彦，西方敦博，“統計処理を用いた打楽器のフォーム分析” Statistical Analysys of Drumming Form，情報処理学会 音楽情報科学研究会研究報告，MUS-47-3 (2002.10)

(5) 辻 靖彦，西方敦博，“演奏記録機能を有する打楽器フォーム測定システムの開発” Development of Drumming Form Recording System with Function of Recording Play，電子情報通信学会総合大会，D-15-30 (2003.3)

(6) 辻 靖彦，西方敦博，“リズムと打拍フォームに基づく打楽器学習支援システムの開発と評価” Development and Evaluation of Drum Learning Support System Based on Rhythm and Drumming Form，電子情報通信学会 教育工学研究会 (ET)，A-12 (2003.12)

(7) 辻 靖彦，西方敦博，“ドラム熟練者の打拍フォーム教示が未熟練者に与える演奏改善効果”， Improvement of Beginners’ Drumming Play by Expert’s Teaching for Drumming Form, 電子情報通信学会総合大会，D-15-2 (2004.3 発表予定)

文献

[1] 長嶋洋一、橋本周司、平賀 譲、平田圭二（編）, “bit 別冊コンピュータと音楽の世界”, 共立出版, 1998

[2] 長尾 真、宇津呂武仁、島津 明、匂坂芳典、井口征士、片寄晴弘 “岩波講座マルチメディア情報学 4：文字と音の情報処理”, 岩波書店, 2000

[3] Kathleen M.Glotti “Cognitive Psychology In and Out of the Laboratory(pp.164-193)”, Brooks/Cole, 1994

[4] 波多野誼余夫 編 “音楽と認知”, 東京大学出版会, 1987

[5] 植田 弘、赤岩和美 他 “はじめての音楽通：50 ジャンル CD500 枚”, 音楽の友社, 1993

[6] 渡邊健一 “音楽の正体”, ヤマハミュージックメディア, 1995

[7] 安西祐一郎 “岩波講座ソフトウェア科学 16　認識と学習”, 岩波書店, 1989

[8] 平野勝彦 “MIDI のフォーマットとアプリケーション”, 電子情報通信学会研究報告、EA86-82, 1986

[9] ローランド（株） “MIDI と MPU － 401”, 日本音響学会誌 41 巻 6 号, 1985

[10] 原田 勉、大照 完、橋本周司 “音楽のマルチメディアインターフェース”, テレビジョン学会技術報告、Vol.15,No.39,pp.27-32, 1991

[11] 加藤 誠巳、藤原 ひろみ “パーソナル・コンピュータを用いた音感訓練支援システム”, 第 38 回情報処理学会全国大会講演論文集、No.2,pp.682-683, 1989

[12] 難波 彰子、平松 茂、近藤 勲 “異機種構成のコンピュータシステムを採用した学習環境での学習者の反応：DTM をグループの合唱練習に使った場合”, 日本科学教育学会研究会研究報告、Vol.10,No.2,pp.23-36, 1995

[13] 長嶋 洋一、中村 文隆、後藤 真孝、片寄 晴弘、井口征士 “ネットワーク上で相互作用するアルゴリズム作曲系を用いた音楽教育システム”, 情報処理学会全国大会講演論文集、第 54 回,No.2,pp.273-274, 1997

[14] 伊藤 彰教 “Web を利用したコンピュータ音楽の初歩教育”，情報処理学会研究報告、MUS-96-102,15-22，1996

[15] 川畑 節幸、寺下 陽一 “コンピュータによる音楽教育”，日本科学教育学会研究会研究報告、Vol.13,No.6,pp.21-24，1999

[16] 有馬 哲、石村貞夫 “多変量解析のはなし”，東京図書，1987

[17] 平野広美 “C でつくるニューラルネットワーク”，パーソナルメディア，1991

[18] 甘利俊一 “ニューラルネットの新展開”，サイエンス社，1993

[19] 甘利俊一、酒田英夫 “脳とニューラルネット”，朝倉書店，1994

[20] 松原道男 “電気回路における生徒の全体的処理の評価”，日本理科教育学会研究紀要、Vol.39,No.1，1998.7

[21] 伊藤久祥、伊東宏起、赤堀侃司 “メニュー選択支援のためのニューラルネットの意味解析”， 電子情報通信学会技術研究報告、ET-96-500、教育工学、Vol.96,No.500,pp.53-60，1997

[22] 伊藤久祥、赤堀侃司 “初心者の操作習熟を支援するユーザインターフェースに関する研究”， 東京工業大学大学院総合理工学研究科システム科学専攻博士論文，1998

[23] 伊東宏起、赤堀侃司 “ニューラルネットによるメニュー選択支援の試み”，東京学芸大学教育学部数学教育専攻卒業論文，1995

[24] 米川孝弘、西方敦博 “リズムと和音に基づく音楽基礎能力の評価法および学習支援システムへの応用に関する研究”，東京工業大学大学院 社会理工学研究科 人間行動システム専攻 博士論文，2001

[25] LiMin Fu “Knowledge Discovery Based on Neural Networks”，COMMUNICATIONS of the ACM,Volume.42, pp.47-50, Number.11 ，1999

[26] LiMin Fu “Neural Networks in Computer Intelligence”， McGraw Hill,NewYork，1994

[27] LiMin Fu “A neural network model for learning domain rules based on its activation function characteristics”，IEEE transaction on neural networks,9,5，1998

[28] Masako Nishijima、Kazuyuki Watanabe “A neural network model for learning domain rules based on its activation function characteristics”， the Proceeding of International Computer Music Conference，1992

[29] 渡辺 和之、西嶋 正子、柿橋 正憲、村上 公一 “ニューラルネットワークを用いたジャズセッションシステム；ニューロミュージシャン”，情報処理学会全国大会講演論文集、第 44 回平成 4 年前期、Num.2.pp.199-200，1992

[30] 藩 浩之、乾 伸雄、野瀬 隆、小谷 善行 “ニューラルネットワークを用いたアドリブ生成のためのリズム学習”，情報処理学会研究報告、MUS 26-6,pp.39-44，1998

[31] 新井 肇、西岡 大祐、瀧口 伸雄、小谷 善行、西村 恕彦 “曲構造とメロディーのリズム解析による自動編曲”，情報処理学会全国大会講演論文集、第 44 回平成 4 年前期、Num.1,pp.401-402，1992

[32] Hajime Sano,B.Keith Jenkins “A Neural Network Model for Pitch Perception”，Computer Music Journal,Vol.13,No.3，1989

[33] 坂本 崇、梶川 嘉延、野村 康雄 “音楽感性空間における非線形判別分析を用いた曲印象別グループの分割”，情報処理学会論文誌,Vol.40,No.4，1999

[34] 池添 剛、梶川 嘉延、野村 康雄 “形容詞対を用いた音楽データベース検索システム”，情報処理学会研究報告、MUS-33-2,pp.7-14，2000

[35] 村上 豊、熊谷俊行、梶川 嘉延、野村 康雄 “ニューラルネットワークによるテンポに関する演奏者情報の抽出”，情報処理学会第 52 回（平成 8 年前期）全国大会、1-437，1996

[36] 今井健太郎、長尾智晴 “SD 法とニューラルネットワークを用いた旋律評価”，東京工業大学工学部情報工学科平成 9 年度卒業論文，1997

[37] 今井 繁、長尾智晴 “遺伝的アルゴリズムを用いた自動作曲に関する研究”，東京工業大学工学部情報工学科平成 9 年度卒業論文，1997

[38] 秋畑 誠、松居 辰則、岡本敏雄 “感性情報による楽曲の分析と創作に関する研究”，電子情報通信学会技術研究報告、ET、教育工学、Vol.98,Num183,pp.41-46，1998

[39] 秋畑 誠、岡本敏雄 “ニューラルネットワークに基づく知識エージェントによる局所的旋律変換と曲の創作”，平成 9・10・11 年度 文部省科学研究費補助金 基盤研究（B）（2）中間報告書 p.44-65

[40] 菅沼優子、長橋 宏 “ニューラルネットワークを用いたジャズの独奏部分における演奏特徴の抽出”，電子情報通信学会総合大会、A-15-27，1999

[41] 鈴木泰山、徳永健伸、田中穂積 “事例ベースの演奏表情生成手法に関する研究”, 情報処理学会研究報告、MUS-21,pp.7-12, 1997

[42] 宮本朋範、田中穂積 “旋律の類似度の自動評価に関する研究”, 東京工業大学工学部情報工学科平成 10 年度卒業論文, 1998

[43] 上符 裕一、青野 裕司、片寄 晴弘、井口 征士 “演奏ルールの抽出について”, 情報処理学会研究報告、MUS-96-53,pp.79-84, 1996

[44] 南高 純一、猪野 真弓、佐藤 邦雄、森川 重則 “曲風を考慮した自動作曲システム-MAGIC”, 情報処理学会全国大会講演論文集、第 39 回平成元年後期、Num.3,pp.2242-2243, 1989

[45] 小田 安彦、白川 健一、村上 豊、梶川 嘉延、野村 康雄 “演奏者情報を加味したピアノの自動演奏システムの構築〜ベロシティ、テンポにおける個人的弾き癖抽出に関する一提案〜”, 情報処理学会第 50 回（平成 7 年前期）全国大会、1-373, 1995

[46] 水谷公生 “ロックスタディ 作曲法”, 音楽の友社, 1992

[47] 水谷公生 “ロックスタディ 編曲法”, 音楽の友社, 1993

[48] 林 知行 “標準ポピュラー・コード理論”, シンコ―ミュージック, 1995

[49] 林 知行 “標準ポピュラー音楽理論”, シンコ―ミュージック, 1997

[50] Debbie Rohwer, “Effect of Movement Instruction on Steady Beat Perception, Synchronization, and Performance”, Journal of Research in Music Education, Vol. 46,no. 3, pp. 414-424, 1998

[51] 辻 靖彦, 赤堀 侃司, “ニューラルネットワークを用いた楽曲のジャンル推定とその教育利用” 日本教育工学雑誌, Vol.24,No.3, pp.183-191, 2000

[52] 米川孝宏, 西方 敦博, “2 純音和音の再生テストの成績と楽典テスト・自己評価との関係”, 電子情報通信学会論文誌, VOL.J85-DI, No.5, pp.484-488, 2002

[53] 米川孝宏, 西方 敦博, 中山実, 清水康敬, “テンポ変動度に基づく基礎リズム能力評価法の検討”, 日本教育工学雑誌,Vol.24,No.3 , pp.153-160, 2000

[54] 簡易モーションキャプチャセンサ DigitEye3D-M100 http://www.kansai-ri.co.jp/icr/digiteye/

[55] 永田 靖、棟近雅彦 “多変量解析法入門” サイエンス社, 2001

[56] “スタンダード MIDI ファイルライブラリー HyperGroove3” （株）アイデックス，1999

[57] 山本雄一“目で見て確認 ドラム” リットーミュージック，1998

[58] 坂田 稔“ドラマーのための弱点補強トレーニング” リットーミュージック, 2001

[59] 西井 雄一郎, 栗本 育三郎, “ドラムパッドを利用した実演音とクリック音との聴覚的ずれ検出装置について” 情報処理学会 音楽情報科学研究会研究報告,MUS-32,pp.17-23，1993

付録　ソースコード一覧

リスト　打楽器学習支援システム メインプログラム

```
  1: // drumSupportDlg.cpp : インプリメンテーション ファイル
  2: //
  3:
  4: #include "stdafx.h"
  5: #include "drumSupport.h"
  6: #include "drumSupportDlg.h"
  7:
  8: #ifdef _DEBUG
  9: #define new DEBUG_NEW
 10: #undef THIS_FILE
 11: static char THIS_FILE[] = __FILE__;
 12: #endif
 13:
 14: // ****************** 定数式 ********************
 15: #define PATTERN 8
 16:
 17:
 18: // ****************** global 変数 ********************
 19: // MIDI 入出力関連
 20: HMIDIOUT gs_hMidiOut;
 21: HMIDIIN gs_hMidiIn;
 22: static MIDIHDR*  gs_pMidiHdr;
 23: static MIDIHDR*  gs_pMidiOutHdr;
 24:
 25: static CString WAVE_PATH="C:/gibsdata/wave";   // wave データのパス
 26: static CString DATA_PATH="C:/gibsdata/subject";// 被験者データのパス
 27:
 28: // 制御関連
 29: int g_nState;      // 状態
 30: int g_nClickInterval;  // クリックのインターバル
 31: int g_nVelocity;    // クリックのベロシティ
 32: int g_nTime;      // 時刻
 33: bool g_is_start;    // フォーム測定が始まったか否か
 34: int g_nError;
 35: bool g_bIsAfter;    // 測定後か否か（測定後～ボタンを押すまで）

 36:
 37: // システム依存の DigitEye オフセット値
 38: static double g_dOffset = 0.06435;
 39:
 40: // スプライン補間に用いる周期の両端からはみ出た変数の個数
 41: static int g_nExtendWidth = 5;
 42:
 43: int g_tempo;          // テンポ
 44: int g_nOtehonBar = 2;      // お手本を鳴らす小節数
 45: int g_nNotePerBar = 2;      // 1 小節あたりの音符の個数
 46: int g_nPerformTime[PATTERN];  // 演奏回数：ラベルが 0～7 であることに要注意
 47:
 48: bool *g_accent;          // アクセント
 49: bool g_setAccent[4] = {0,1,0,1};  // 一定にセットしておくアクセント
 50: static double g_rateAccent[8][4] = {
 51:   {0.20, 0.725, 0, 0},
 52:   {0.20, 0.695, 0.845, 0},
 53:   {0.20, 0.515, 0.775, 0},
 54:   {0.20, 0.575, 0.785, 0},
 55:   {0.20, 0.55, 0.75, 0.875},
 56:   {0.20, 0.55, 0.675, 0.80},
 57:   {0.20, 0.490, 0.640, 0.790},
 58:   {0.20, 0.640, 0.765, 0.880}};
 59:
 60:
 61: static int g_nBius = 149;    // 楽譜画像の ID
 62: static int g_nMiniBius = 141;  // 小楽譜画像 ID
 63:
 64: // 一つ一つの音符間でのデータ
 65: double g_meanSerialEn;  // En 平均値
 66: double g_sdSerialEn;  // En 標準偏差
 67: double g_meanSerialVn;  // Vn 平均値
 68: double g_sdSerialVn;  // Vn 標準偏差
 69: double g_meanSerialTn;  // Tn 平均値
 70: double g_sdSerialTn;  // Tn 標準偏差
 71:
 72: // 小節間でのデータ
 73: double* g_meanEn;    // En 平均値
 74: double* g_sdEn;      // En 標準偏差
 75: double* g_meanRn;    // Rn 平均値
 76: double* g_sdRn;      // Rn 標準偏差
 77: double* g_meanSn;    // Sn 平均値
 78: double* g_sdSn;      // Sn 標準偏差
 79: double* g_meanTn;    // Vn 平均値
 80: double* g_sdTn;      // Vn 標準偏差
 81: double* g_meanVn;    // Tn 平均値
 82: double* g_sdVn;      // Tn 標準偏差
 83:
 84: // yonekawa データ
 85: double* g_meanSHIFT;  // logfitting 後の平均
 86: double* g_sdSHIFT;    // logfitting 後の偏差
 87:
 88:
 89: // Event 格納用構造体
 90: struct midiEvent
 91: {
 92:   int nCh;
 93:   long double dDiffTime;//timestamp
 94:   int dAbstime;
 95:   time_t eventTime;// timestamp
 96:   int eventTime_ms;// millisecond
 97: //  double dDifftime;
 98:   int nVelocity;
 99:   int nNoteNumber;
100:   int nNoteOnOff;
101:   midiEvent* pNextEvent;
102: };
103:
104: // 演奏データ保存用ポインタ
105: midiEvent *pStart;
106: midiEvent *pEnd;
107: midiEvent *pNow;
108:
109: // EnRn 計算用データ
110: struct positionForCalc
111: {
112:   int number;// 何周期目か
113:
114:   // その郡の先頭の打拍時刻及び次の先頭打拍時刻
115:   double dStartTime;
116:   double dEndTime;
117:
118:   // 打拍の強さ（Velocity）
119:   int nVelocity;
120:
121:   // 時刻
122:   double *dTime;
123:   // ラベル
124:   int *nLabel;
125:   // 1 周期分の位置データ
126:   int *posi_x;
127:   int *posi_y;
128:   int *posi_z;
129:
130:   int nData;// データ個数（0...99）
131:
132:   positionForCalc* pNext;
133: };
134: // 保存用ポインタ
135: positionForCalc* p_calcStart=NULL;
136: positionForCalc* p_calcEnd=NULL;
137: positionForCalc* p_calcNow=NULL;
138:
139: // 構造体「小節」
140: struct BarLine
141: {
142:   // n 小節目
143:   int number;
```

```
144:   // 打拍絶対時刻（音符数分）
145:   double* dTime;
146:   // 音符の長さ（音符数分）
147:   double* dDiffTime;
148:   // 位置情報を指すポインタ（音符数分）
149:   positionForCalc **pNote;
150:   // 音符の相対的長さ
151:   double* dRelativeTime;
152:   // フォーム誤差 En
153:   double* En;
154:   // 振幅 RMS（今のところ y だけ）
155:   double* Rn;
156:   // 最大振幅（今のところ y だけ）
157:   double* Sn;
158:   // リズム誤差 Tn
159:   double* Tn;
160:
161:   BarLine* pNextBar;
162: };
163: // 保存用ポインタ
164: BarLine* p_barStart=NULL;
165: BarLine* p_barEnd=NULL;
166: BarLine* p_barNow=NULL;
167:
168: // お手本リズムパターン
169: double * g_dTemplateNotes;
170:
171: // 表示用
172: CString g_strRes;
173:
174: // ****************** Callback functions ***************
175: // MIDI 入力用
176: void CALLBACK MidiInProc(HMIDIIN, UINT, DWORD, DWORD, DWORD);
177: // SetTimer で指定
178: void CALLBACK EXPORT MyTimerProc(
179:  HWND,    // SetTimer を呼び出した CWnd の ハンドル
180:  UINT,    // WM_TIMER
181:  UINT,    // タイマ 識別子
182:  DWORD    // システム 時刻
183: );
184:
185:
186: /////////////////////////////////////////////////////////
////////////////////
187: // アプリケーションのバージョン情報で使われている CAboutDlg ダ
イアログ
188:
189: class CAboutDlg : public CDialog
190: {
191: public:
192:   CAboutDlg();
193:
194: // ダイアログ データ
195:   //{{AFX_DATA(CAboutDlg)
196:   enum { IDD = IDD_ABOUTBOX };
197:   //}}AFX_DATA
198:
199:   // ClassWizard は仮想関数のオーバーライドを生成します
200:   //{{AFX_VIRTUAL(CAboutDlg)
201:   protected:
202:   virtual void DoDataExchange(CDataExchange* pDX);
// DDX/DDV のサポート
203:   //}}AFX_VIRTUAL
204:
205: // インプリメンテーション
206: protected:
207:   //{{AFX_MSG(CAboutDlg)
208:   //}}AFX_MSG
209:   DECLARE_MESSAGE_MAP()
210: };
211:
212: CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
213: {
214:   //{{AFX_DATA_INIT(CAboutDlg)
215:   //}}AFX_DATA_INIT
216: }
217:
218: void CAboutDlg::DoDataExchange(CDataExchange* pDX)
219: {
220:   CDialog::DoDataExchange(pDX);
221:   //{{AFX_DATA_MAP(CAboutDlg)
222:   //}}AFX_DATA_MAP
223: }
224:
225: BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
226:   //{{AFX_MSG_MAP(CAboutDlg)
227:     // メッセージ ハンドラがありません。
228:   //}}AFX_MSG_MAP
229: END_MESSAGE_MAP()
230:
231: /////////////////////////////////////////////////////////
////////////////////
232: // CRecordPlayMidiDlg ダイアログ
233:
234: CRecordPlayMidiDlg::CRecordPlayMidiDlg(CWnd* pParent
/*=NULL*/)
235:   : CDialog(CRecordPlayMidiDlg::IDD, pParent)
236: {
237:   //{{AFX_DATA_INIT(CRecordPlayMidiDlg)
238:   m_strRes = _T("");
239:   m_nTestSec = 0;
240:   m_sUserName = _T("");
241:   m_nSelectedPattern = 0;
242:   m_nPerformTime = 0;
243:   //}}AFX_DATA_INIT
244:   // メモ: LoadIcon は Win32 の DestroyIcon のサブシーケンス
を要求しません。
245:   m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
246: }
247:
248: void CRecordPlayMidiDlg::DoDataExchange(CDataExchange* pDX)
249: {
250:   CDialog::DoDataExchange(pDX);
251:   //{{AFX_DATA_MAP(CRecordPlayMidiDlg)
252:   DDX_Control(pDX, IDC_STROKE, m_ctrlStroke);
253:   DDX_Control(pDX, IDC_YOUR, m_ctrlYour);
254:   DDX_Control(pDX, IDC_CORRECT, m_ctrlCorrect);
255:   DDX_Control(pDX, IDC_BAR, m_barCtrl);
256:   DDX_Control(pDX, IDC_STATIC_TEXT, m_ctrlstrRes);
257:   DDX_Text(pDX, IDC_STATIC_TEXT, m_strRes);
258:   DDX_Text(pDX, IDC_TEST_SEC, m_nTestSec);
259:   DDV_MinMaxInt(pDX, m_nTestSec, 0, 300);
260:   DDX_Text(pDX, IDC_USERNAME, m_sUserName);
261:   DDV_MaxChars(pDX, m_sUserName, 20);
262:   DDX_Text(pDX, IDC_SELECTED, m_nSelectedPattern);
263:   DDV_MinMaxInt(pDX, m_nSelectedPattern, 1, 8);
264:   DDX_Text(pDX, IDC_PERFORM_TIME, m_nPerformTime);
265:   DDV_MinMaxInt(pDX, m_nPerformTime, 0, 10);
266:   //}}AFX_DATA_MAP
267: }
268:
269: BEGIN_MESSAGE_MAP(CRecordPlayMidiDlg, CDialog)
270:   //{{AFX_MSG_MAP(CRecordPlayMidiDlg)
271:   ON_WM_SYSCOMMAND()
272:   ON_WM_PAINT()
273:   ON_WM_QUERYDRAGICON()
274:   ON_BN_CLICKED(IDC_TEMPO_PLAY, OnTempoPlay)
275:   ON_WM_TIMER()
276:   ON_BN_CLICKED(IDC_PLAY, OnPlay)
277:   ON_MESSAGE(WM_DE_REAL, OnReal)// DigitEye
278:   ON_BN_CLICKED(IDC_BUTTON_TEST, OnButtonTest)
279:   ON_BN_CLICKED(IDC_MINI5, OnRhythm5)
280:   ON_BN_CLICKED(IDC_MINI6, OnRhythm6)
281:   ON_BN_CLICKED(IDC_MINI7, OnRhythm7)
282:   ON_BN_CLICKED(IDC_MINI1, OnRhythm1)
283:   ON_BN_CLICKED(IDC_MINI2, OnRhythm2)
284:   ON_BN_CLICKED(IDC_MINI3, OnRhythm3)
285:   ON_BN_CLICKED(IDC_MINI4, OnRhythm4)
286:   ON_BN_CLICKED(IDC_MINI8, OnRhythm8)
287:   ON_BN_CLICKED(IDC_BAR_ADVICE, OnBarAdvice)
288:   ON_BN_CLICKED(IDC_FORM_ADVICE, OnFormAdvice)
289:   ON_BN_CLICKED(IDC_VELO_ADVICE, OnVeloAdvice)
290:   ON_BN_CLICKED(IDC_PUT_ACCENT, OnPutAccent)
291:   //}}AFX_MSG_MAP
292: END_MESSAGE_MAP()
293:
294: /////////////////////////////////////////////////////////
////////////////////
295: // CRecordPlayMidiDlg メッセージ ハンドラ
296:
297: BOOL CRecordPlayMidiDlg::OnInitDialog()
298: {
299:   CDialog::OnInitDialog();
300:
301:   // "バージョン情報..." メニュー項目をシステム メニューへ追加
します。
302:
303:   // IDM_ABOUTBOX はコマンド メニューの範囲でなければなりませ
ん。
304:   ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
305:   ASSERT(IDM_ABOUTBOX < 0xF000);
306:
307:   CMenu* pSysMenu = GetSystemMenu(FALSE);
308:   if (pSysMenu != NULL)
309:   {
310:     CString strAboutMenu;
311:     strAboutMenu.LoadString(IDS_ABOUTBOX);
312:     if (!strAboutMenu.IsEmpty())
```

```
313:     {
314:       pSysMenu->AppendMenu(MF_SEPARATOR);
315:       pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
316:     }
317:   }
318:
319:   // このダイアログ用のアイコンを設定します。フレームワークはアプ
リケーションのメイン
320:   // ウィンドウがダイアログでない時は自動的に設定しません。
321:   SetIcon(m_hIcon, TRUE);       // 大きいアイコンを設定
322:   SetIcon(m_hIcon, FALSE);    // 小さいアイコンを設定
323:
324:   // TODO: 特別な初期化を行う時はこの場所に追加してください。
325: //  wavePlay(WAVE_PATH + "/bunsekisiteimasu.wav");
326: //  exit(0);
327:
328:   g_nState = IDLE;
329:   g_is_start = false;
330:   initMidiIn(0);
331:   initMidiOut();
332:   m_nTestSec = 60;// default 10[sec]
333:   m_sUserName = "";
334:   g_bIsAfter=0;
335:   m_isSetAccent=0;
336:
337:   // アクセントモード 0... なし 1... 一定 2... ランダム
338:   m_nAccentMode=1;
339:
340:   UpdateData(false);
341:
342:   // 回数の初期値
343:   {
344:     int i;
345:     for(i=0;i<PATTERN;i++)
346:     {
347:       g_nPerformTime[i]=1;
348:     }
349:     m_nPerformTime = 1;
350:   }
351:   // default は 90[bpm] にチェック
352:   static int first = 1;
353:   if(first)
354:   {
355:     first=0;
356:     // 90BPM
357:     CButton* radio90=(CButton*)GetDlgItem(IDC_TEMPO90_BIG);
358:     radio90->SetCheck(1);
359:     g_tempo=90;
360:     g_nVelocity=63;
361:
362:     // リズムパターン 1 をデフォルト
363:     m_nSelectedPattern = 1;
364:     UpdateData(false);
365:     g_nNotePerBar = 2;
366:     g_dTemplateNotes = new double[g_nNotePerBar];
367:     g_dTemplateNotes[0] = 0.75;
368:     g_dTemplateNotes[1] = 0.25;
369:     // 画像表示
370:     printBarPict();
371:     drawCorrectLength();// なぜか動かず
372:   }
373:
374:
375: // DigitEye3D の初期設定
376: //==>>
377:   int    i;
378:
379:   m_is_real = false;
380:   m_is_rec = false;
381:   for(i = 0 ; i < 4 ; i++)
382:     m_dx[i] = m_dy[i] = m_dz[i] = 0;
383:
384:   //レコーディングまわりの初期化
385:   {
386:     m_rec.idx = 0;
387:     m_rec.cycl = 0;
388:   }
389:
390:   //シリアルポート (COM1) をオープンし、全てのチャネルとビデオ映
像座標、エラー情報全てを送信してくるように設定
391:   //レンズテーブルを指定した上でシリアルポートを開く
392:   if(eye.open(1, "./DE-1-129.tbl", this->m_hWnd, DE_ALLCH |
DE_REAL | DE_NEEDSTATMSG) == TRUE)
393:   {
394:     //正常
395:     m_is_real = true;
396:     //メッセージ送信の開始
397:     eye.start();
398:   }
399:   else
400:   {
401:     //失敗
402:     m_is_real = false;
403:     MessageBox("レンズテーブルファイルの読込に失敗しました",
"Load", MB_OK);
404:     eye.close();
405:     exit(0);
406:   }
407: //<<==
408: //  MessageBox("OnInitDialog は無事に終了しました。", "Load",
MB_OK);
409:
410:   return TRUE;  // TRUE を返すとコントロールに設定したフォーカ
スは
失われません。
411: }
412:
413: void CRecordPlayMidiDlg::OnSysCommand(UINT nID, LPARAM lParam)
414: {
415:   if ((nID & 0xFFF0) == IDM_ABOUTBOX)
416:   {
417:     CAboutDlg dlgAbout;
418:     dlgAbout.DoModal();
419:   }
420:   else
421:   {
422:     CDialog::OnSysCommand(nID, lParam);
423:   }
424: }
425:
426: // もしダイアログボックスに最小化ボタンを追加するならば、アイコン
を描画する
427: // コードを以下に記述する必要があります。MFC アプリケーションは
document/view
428: // モデルを使っているので、この処理はフレームワークにより自動的に
処理されます。
429:
430: void CRecordPlayMidiDlg::OnPaint()
431: {
432:   if (IsIconic())
433:   {
434:     CPaintDC dc(this); // 描画用のデバイス コンテキスト
435:
436:     SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(),
0);
437:
438:     // クライアントの矩形領域内の中央
439:     int cxIcon = GetSystemMetrics(SM_CXICON);
440:     int cyIcon = GetSystemMetrics(SM_CYICON);
441:     CRect rect;
442:     GetClientRect(&rect);
443:     int x = (rect.Width() - cxIcon + 1) / 2;
444:     int y = (rect.Height() - cyIcon + 1) / 2;
445:
446:     // アイコンを描画します。
447:     dc.DrawIcon(x, y, m_hIcon);
448:   }
449:   else
450:   {
451:     CDialog::OnPaint();
452:
453:     // ---楽譜画像の表示----------------------------
454:     printBarPict();
455:     // ---アクセントの表示----------------------------
456:     if(m_isSetAccent==1)
457:     {
458:       drawAccent();
459:     }
460:     // ---教師リズムパターンの表示----------------------------
461:     drawCorrectLength();
462:     // ---各種結果の表示----------------------------
463:     if(g_bIsAfter==1)
464:     {
465:       drawYourLength();
466:       drawStrokeError();
467:       drawVelocity();
468:     }
469:   }
470: }
471:
472: // システムは、ユーザーが最小化ウィンドウをドラッグしている間、
473: // カーソルを表示するためにここを呼び出します。
474: HCURSOR CRecordPlayMidiDlg::OnQueryDragIcon()
475: {
476:   return (HCURSOR) m_hIcon;
477: }
478:
```

```
 479: /*****************************************************
**************
 480:  ** FUNCTION : ・MIDI 入力の初期処理
 481:  **      :
 482:  *****************************************************
*************/
 483: int CRecordPlayMidiDlg::initMidiIn(int nInDevNo)
 484: {
 485:   int     i;
 486:   int     nDevNum;
 487:   int     nKeyNo;
 488:   int     nErr = 0;
 489:   MMRESULT  mmRes;
 490:   MIDIINCAPS  miCaps;
 491:   char tmp[255];
 492:
 493:   /* 入力デバイスの表示 */
 494:   nDevNum = midiInGetNumDevs();
 495:   sprintf(tmp, "MIDI 入力デバイスの数は %d 個です\r\n", nDevNum);
 496:   m_strRes+=tmp;
 497:
 498:   for (i=0; i<nDevNum; i++)
 499:   {
 500:     midiInGetDevCaps(i, &miCaps, sizeof(miCaps));
 501:     sprintf(tmp, "  デバイス番号 %d : %s\r\n", i, miCaps.
szPname);
 502:     m_strRes+=tmp;
 503:   }
 504:
 505:   if (nInDevNo>=0 && nInDevNo<nDevNum)
 506:   {
 507:     sprintf(tmp, "選択された入力 MIDI デバイス ==> %d\r\n
\r\n", nInDevNo);
 508:     m_strRes+=tmp;
 509:     nKeyNo = nInDevNo;
 510:   }
 511:   else {
 512:
 513:     /* 入力デバイスの選択 */
 514:     fflush(stdin);
 515:
 516:     nKeyNo = 0;
 517:     sprintf(tmp, "選択された入力 MIDI デバイス ==> %d\r\n
\r\n", nKeyNo);
 518:     m_strRes+=tmp;
 519:   }
 520:
 521:   mmRes = midiInOpen(&gs_hMidiIn, nKeyNo, (DWORD)MidiInProc,
(DWORD)NULL, CALLBACK_FUNCTION);
 522:   if (mmRes == MMSYSERR_NOERROR) {
 523:     mmRes = midiInReset(gs_hMidiIn);
 524:
 525:     /* MIDIHDR バッファブロックを割り当てる */
 526:     gs_pMidiHdr = (MIDIHDR *)HeapAlloc(GetProcessHeap(),
HEAP_ZERO_MEMORY, sizeof(MIDIHDR));
 527:     if (gs_pMidiHdr)
 528:     {
 529:       gs_pMidiHdr->lpData =
(char *)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, 1024L);
 530:       gs_pMidiHdr->dwBufferLength = 1024L;
 531:     }
 532:     else
 533:     {
 534:       sprintf(tmp, "ERROR: HeapAlloc のエラー\r\n");
 535:       m_strRes+=tmp;
 536:       return NG;
 537:     }
 538:     nErr = 1;
 539:   }
 540:
 541:   if (mmRes == MMSYSERR_NOERROR) {
 542:     mmRes = midiInPrepareHeader(gs_hMidiIn, gs_pMidiHdr,
sizeof(MIDIHDR));
 543:   }
 544:   if (mmRes == MMSYSERR_NOERROR) {
 545:     mmRes = midiInAddBuffer(gs_hMidiIn, gs_pMidiHdr,
sizeof(MIDIHDR));
 546:   }
 547:   if (mmRes == MMSYSERR_NOERROR) {
 548:     mmRes = midiInStart(gs_hMidiIn);
 549:   }
 550:
 551:   if (mmRes != MMSYSERR_NOERROR) {
 552:     g_nError = mmRes;
 553:     return NG;
 554:   }
 555:
 556:   return OK;
 557: }
 558:
 559:
 560: // MIDI 出力
 561: int CRecordPlayMidiDlg::initMidiOut()
 562: {
 563:   char tmp[100]="";
 564:   char buf[1000]="";
 565:   int nn,i;
 566:   MMRESULT mmRes;
 567:   MIDIOUTCAPS  moCaps;
 568:
 569:
 570:   nn = midiOutGetNumDevs();
 571:   sprintf(tmp, "MIDI 出力デバイスの数は %d 個です\r\n", nn);
 572:   m_strRes+=tmp;
 573:
 574:   if(nn == 0)
 575:   {
 576:     sprintf(tmp,"midi 音源を使用できません。\r\n");
 577:     m_strRes += tmp;
 578:     return NG;
 579:   }
 580:
 581:   for (i=0; i<nn; i++) {
 582:     midiOutGetDevCaps(i, &moCaps, sizeof(moCaps));
 583:     sprintf(tmp, "  デバイス番号 %d : %s\r\n", i, moCaps.
szPname);
 584:     m_strRes += tmp;
 585:   }
 586:
 587:   UpdateData(FALSE);
 588:
 589:   sprintf(tmp, "選択された出力 MIDI デバイス ==> %d\r\n\r\n",
MIDI_DTX);
 590:   m_strRes += tmp;
 591:   UpdateData(FALSE);
 592:
 593:   // DTX が 0 番号
 594:   mmRes = midiOutOpen(&gs_hMidiOut, MIDI_DTX, 0, 0, 0);
 595:
 596:
 597:   if (mmRes == MMSYSERR_NOERROR) {
 598:     mmRes = midiOutReset(gs_hMidiOut);
 599:
 600:     /* MIDIHDR バッファブロックを割り当てる */
 601:     gs_pMidiOutHdr = (MIDIHDR *)HeapAlloc(GetProcessHeap(),
HEAP_ZERO_MEMORY, sizeof(MIDIHDR));
 602:     if (gs_pMidiOutHdr) {
 603:       gs_pMidiOutHdr->lpData =
(char *)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, 1024L);
 604:       gs_pMidiOutHdr->dwBufferLength = 1024L;
 605:     }
 606:     else {
 607:       MessageBox("ERROR: HeapAlloc のエラー");
 608:       return NG;
 609:     }
 610:   }
 611:   else
 612:   {
 613:     sprintf(tmp,"出力オープンエラー\r\n");
 614:     m_strRes += tmp;
 615:     UpdateData(FALSE);
 616:   }
 617:
 618:   if (mmRes == MMSYSERR_NOERROR)
 619:   {
 620:     mmRes = midiOutPrepareHeader(gs_hMidiOut, gs_pMidiOutHdr,
sizeof(MIDIHDR));
 621:   }
 622:
 623:
 624:   if(mmRes != MMSYSERR_NOERROR)
 625:   {
 626:     sprintf(tmp,"MIDI 音源がすでに使用されているか、その他の
理由で使用できません。\r\n");
 627:     MessageBox(tmp);
 628:     return NG;
 629:   }
 630:   else
 631:   {
 632:     sprintf(tmp,"midiOutPrepareHeader の結果: mmRes=%d\r\n",
mmRes);
 633:     m_strRes += tmp;
 634:     UpdateData(FALSE);
 635:   }
 636:
 637:   return OK;
 638: }
 639:
```

```
640: void CRecordPlayMidiDlg::OnOK()
641: {
642:   // TODO: この位置にその他の検証用のコードを追加してください
643:   MMRESULT ret;
644:
645:
646:   // 前のイベントを削除
647:   while(pStart!=NULL)
648:   {
649:     pNow=pStart;
650:     pStart=pStart->pNextEvent;
651:     delete(pNow);
652:   }
653:   pEnd = pStart;
654:   while(p_calcStart!=NULL)
655:   {
656:     p_calcNow=p_calcStart;
657:     p_calcStart=p_calcStart->pNext;
658:     // 各動的要素を削除
659:     delete(p_calcNow->dTime);
660:     delete(p_calcNow->nLabel);
661:     delete(p_calcNow->posi_x);
662:     delete(p_calcNow->posi_y);
663:     delete(p_calcNow->posi_z);
664:     // 全体を削除
665:     delete(p_calcNow);
666:   }
667:
668:   {
669:     int count=0;
670:
671:     p_calcEnd = p_calcStart;
672:     while(p_barStart!=NULL)
673:     {
674:       p_barNow=p_barStart;
675:       p_barStart=p_barStart->pNextBar;
676:       // 各動的要素を削除
677:       delete(p_barNow->dTime);
678:       delete(p_barNow->Rn);
679:       delete(p_barNow->Sn);
680:       delete(p_barNow->dDiffTime);
681:       delete(p_barNow->dRelativeTime);
682:       if(count > 0)
683:       {
684:         delete(p_barNow->En);
685:         delete(p_barNow->Tn);
686:       }
687:       // 全体を削除
688:       delete(p_barNow);
689:       count++;
690:     }
691:     p_barEnd = p_barStart;
692:   }
693:
694:   midiOutUnprepareHeader(gs_hMidiOut, gs_pMidiOutHdr,
sizeof(MIDIHDR));
695:   HeapFree(GetProcessHeap(),0,gs_pMidiOutHdr->lpData);
696:   HeapFree(GetProcessHeap(),0,gs_pMidiOutHdr);
697:
698:   ret = midiOutReset(gs_hMidiOut);
699:   ret = midiOutClose(gs_hMidiOut);
700:   ret = midiInReset(gs_hMidiIn);
701:   ret = midiInClose(gs_hMidiIn);
702:
703:   CDialog::OnOK();
704: }
705:
706: VOID CALLBACK MidiInProc
707: (
708:   HMIDIIN hMidiIn,
709:   UINT uMsg,
710:   DWORD dwInstance,
711:   DWORD dwPara1,
712:   DWORD dwPara2
713:   )
714: {
715:   BYTE  bytStatus;
716:   char tmp[256];
717:   time_t now;
718:   struct _timeb timebuffer;
719:
720:   time(&now);
721:   _ftime(&timebuffer);
722:
723:   switch (g_nState)
724:   {
725:   // ************* レコーディング中 **************
726:   case TESTING:
727:   case RECORDING:
728:
729:     switch (uMsg)
730:     {
731:     /* ショートメッセージのみ転送 */
732:     case MIM_DATA:
733:     case MIM_ERROR:
734:       bytStatus = LOBYTE(LOWORD(dwPara1));
735:       if (bytStatus < 0xf8)
736:       {   /* システムのリアルタイムメッセージは捨てる */
737:         if(LOBYTE(HIWORD(dwPara1)) > 0 &&
g_nTime < 0)
738:         {
739:           g_nTime = 0;
740:         }
741:         if(g_is_start == false)
742:         {
743:           g_is_start = true;
744:         }
745:
746:         // MIDI Event 追加
747:         if(LOBYTE(HIWORD(dwPara1)) > 0)
748:         {
749:           pNow = new midiEvent;
750:           pNow->pNextEvent = NULL;
751:           pNow->dAbstime = g_nTime;
752:           pNow->nCh = LOBYTE(LOWORD(dwPara1)) % 16;
753:           pNow->nNoteNumber = HIBYTE(LOWORD(dwPara1));
754:           pNow->eventTime=now;
755:           pNow->eventTime_ms=timebuffer.millitm;
756:           // Note on or Note off
757:           if(LOBYTE(HIWORD(dwPara1)) == 0)
758:           {
759:             pNow->nNoteOnOff = 0;
760:             pNow->nVelocity = 0;
761:           }
762:           else
763:           {
764:             pNow->nNoteOnOff = 0;
765:             pNow->nVelocity = LOBYTE(HIWORD(dwPara1));
766:           }
767:           // ポインタをつなげる
768:           if(pStart == NULL && pEnd == NULL)
769:           {
770:             // 絶対時刻
771:             pNow->dDiffTime = 0;
772:
773:             pStart = pNow;
774:             pEnd = pNow;
775:           }
776:           else if(pStart != NULL && pEnd != NULL)
777:           {
778:             // 絶対時刻
779:             pNow->dDiffTime =
  (double)now - (double)pStart->eventTime +
  (double)pNow->eventTime_ms / 1000 -
  (double)pStart->eventTime_ms / 1000;
780:
781:             pEnd->pNextEvent=pNow;
782:             pEnd=pEnd->pNextEvent;
783:           }
784:           else
785:           {
786:             g_strRes += "MIDI Error.\r\n";
787:           }
788:         }
789:       }
790:       break;
791:
792:     case MIM_LONGDATA:
793:       fprintf(stdout, "MIDI Event: MIM_LONGDATA\r\n");
794:       break;
795:
796:     case MIM_OPEN:
797:       sprintf(tmp, "MIDI Event: Short Message: %x %x %x\r\n",
LOBYTE(LOWORD(dwPara1)),
798:             HIBYTE(LOWORD(dwPara1)), LOBYTE(HIWORD(dwPara1)) );
799:       g_strRes += tmp;
800:       break;
801:
802:     case MIM_CLOSE:
803:       fprintf(stdout, "MIDI Event: MIM_CLOSE\r\n");
804:       break;
805:
806:     case MIM_LONGERROR:
807:       fprintf(stdout, "MIDI Event: MIM_LONGERROR\r\n");
808:       break;
809:
810:     case MIM_MOREDATA:
811:       fprintf(stdout, "MIDI Event: MIM_MOREDATA\r\n");
```

```
812:       break;
813:
814:     default:
815:       break;
816:     }
817:     break;
818:
819:     // ************* クリック中 **************
820:   case CLICKING:
821:     break;
822:
823:   default:
824:     break;
825:   }
826: }
827:
828: void CRecordPlayMidiDlg::OnTempoPlay()
829: {
830:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
831:   UpdateData(TRUE);
832:
833:   CButton* tempo90big=(CButton*)GetDlgItem(
IDC_TEMPO90_BIG);
834:   CButton* tempo120big=(CButton*)GetDlgItem(
IDC_TEMPO120_BIG);
835:
836:   if(g_nState == IDLE)
837:   {
838:     if(tempo90big->GetCheck()) {g_tempo=90;
g_nVelocity=63;}
839:     if(tempo120big->GetCheck()) {g_tempo=120;
g_nVelocity=63;}
840:
841:     g_nState = CLICKING;
842:     g_nClickInterval = getInterval(g_tempo);
843:
844:     // ボタンの名前を変更
845:     CButton* metro=(CButton*)GetDlgItem(IDC_TEMPO_PLAY);
846:     metro->SetWindowText("&ストップ");
847:     // 開始ボタンを使用不可能に
848:     CButton* kaishi=(CButton*)GetDlgItem(IDC_BUTTON_TEST);
849:     kaishi->EnableWindow(false);
850:     // リズムパターンボタンを使用不可能に
851:     enableRhythmButton(false);
852:
853:     m_strRes.Format("Clicking...\r\n");
854:     UpdateData(FALSE);
855:
856:     g_nTime = 0;
857:     SetTimer(ID_CLICK_TIMER,10,NULL);
858:   }
859:   else if(g_nState == CLICKING)
860:   {
861:     KillTimer(ID_CLICK_TIMER);
862:     g_nState = IDLE;
863:
864:     // ボタンの名前を戻す
865:     CButton* metro=(CButton*)GetDlgItem(IDC_TEMPO_PLAY);
866:     metro->SetWindowText("&メトロノーム");
867:     // 開始ボタンを使用可能に
868:     CButton* kaishi=(CButton*)GetDlgItem(IDC_BUTTON_TEST);
869:     kaishi->EnableWindow(true);
870:     // リズムパターンボタンを使用可能に
871:     enableRhythmButton(true);
872:
873:     m_strRes.Format("Stopped.\r\n");
874:     UpdateData(FALSE);
875:   }
876:   else
877:   {
878:     MessageBox("fatal error.");
879:   }
880:
881: }
882:
883: int CRecordPlayMidiDlg::getInterval(int tempo)
884: {
885:   return (60 * 1000 / tempo);
886: }
887:
888: void CRecordPlayMidiDlg::OnTimer(UINT nIDEvent)
889: {
890:   // TODO: この位置にメッセージ ハンドラ用のコードを追加するか
またはデフォルトの処理を呼び出してください
891:
892:   MMRESULT ret;
893:   DWORD dwMsg1,dwMsg2_4;
894:   DWORD dwMsg1_off,dwMsg2_4_off;
895:   char tmp[100];
896:
897:   dwMsg1 = 0x7F2599;
898:   dwMsg1_off = 0x002599;
899:   dwMsg2_4 = 0x7F2599;
900:   dwMsg2_4_off = 0x002599;
901:
902:   if(nIDEvent == ID_CLICK_TIMER)
903:   {
904:     // 現在時刻
905:     g_nTime = g_nTime + 10;
906:
907:     dwMsg1 = (g_nVelocity << 16) + 0x2599;// リムの音
908: //    dwMsg1 = (g_nVelocity << 16) + 0x1f99;// スネアの音
909:     // 一定間隔を過ぎれば鳴らす
910:     if(g_nTime >= g_nClickInterval)
911:     {
912:       ret = midiOutShortMsg(gs_hMidiOut, dwMsg1);
913:       g_nTime = 0;
914: //      ret = midiOutShortMsg(gs_hMidiOut, dwMsg1_off);
915:     }
916:   }
917:   else if(nIDEvent == ID_REC_TIMER ||
nIDEvent == ID_TEST_TIMER)
918:   {
919:     // 現在時刻
920:     if(g_nTime >= 0)
921:     {
922:       g_nTime = g_nTime + 1;
923:     }
924:     if(g_strRes != "")
925:     {
926:       m_strRes += g_strRes;
927:       g_strRes = "";
928:       UpdateData(FALSE);
929:     }
930:   }
931:   else if(nIDEvent == ID_PLAY_TIMER)
932:   {
933:     // 現在時刻
934:     g_nTime = g_nTime + 1;
935:     if(pNow == NULL)
936:     {
937:       KillTimer(ID_PLAY_TIMER);
938:       g_nState = IDLE;
939:       m_strRes += "playing finished\r\n";
940:       UpdateData(FALSE);
941:       if(g_strRes != "")
942:       {
943:         m_strRes += g_strRes;
944:         g_strRes = "";
945:         UpdateData(FALSE);
946:       }
947:     }
948:     else if(g_nTime > pNow->dAbstime)
949:     {
950:       dwMsg1 = (pNow->nVelocity << 16) +
(pNow->nNoteNumber << 8) + (9 << 4) + pNow->nCh;
951:       ret = midiOutShortMsg(gs_hMidiOut, dwMsg1);
952:
953:       sprintf(tmp,"%d %d\r\n",pNow->eventTime,
pNow->eventTime_ms);
954:
955:       pNow = pNow->pNextEvent;
956:       m_strRes += tmp;
957:       UpdateData(FALSE);
958:       if(g_strRes != "")
959:       {
960:         m_strRes += g_strRes;
961:         g_strRes = "";
962:         UpdateData(FALSE);
963:       }
964:     }
965:   }
966:
967:   CDialog::OnTimer(nIDEvent);
968: }
969:
970: int CRecordPlayMidiDlg::addStrRes(CString tmp)
971: {
972:   m_strRes += tmp;
973:
974:   return 1;
975: }
976:
977: void CRecordPlayMidiDlg::OnPlay()
978: {
979:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
```

```
 980:   if(g_nState == IDLE)
 981:   {
 982:     g_nState = PLAYING;
 983:
 984:     if(pStart != NULL)
 985:     {
 986:       m_strRes.Format("Playing...\r\n");
 987:       UpdateData(FALSE);
 988:
 989:       pNow=pStart;
 990:       g_nTime = 0;
 991:       SetTimer(ID_PLAY_TIMER,10,NULL);
 992:     }
 993:     else
 994:     {
 995:       m_strRes.Format("no play data.\r\n");
 996:     }
 997:
 998:   }
 999:   else if(g_nState == CLICKING)
1000:   {
1001:     KillTimer(ID_CLICK_TIMER);
1002:     g_nState = IDLE;
1003:     m_strRes.Format("It's clicking now.\r\n
So stopped.\r\n");
1004:     UpdateData(FALSE);
1005:   }
1006:   else if(g_nState == RECORDING)
1007:   {
1008:     // 録音終了
1009:     KillTimer(ID_REC_TIMER);
1010:     m_strRes.Format("recording stopped.\r\n");
1011:     g_nState = IDLE;
1012:     UpdateData(FALSE);
1013:   }
1014:   else if(g_nState == PLAYING)
1015:   {
1016:     // 録音終了
1017:     KillTimer(ID_PLAY_TIMER);
1018:     m_strRes.Format("Playing stopped.\r\n");
1019:     g_nState = IDLE;
1020:     UpdateData(FALSE);
1021:   }
1022:   else
1023:   {
1024:     MessageBox("fatal error.");
1025:   }
1026: }
1027:
1028:
1029: //WM_DE_REAL メッセージを受け取ったとき
1030: void CRecordPlayMidiDlg::OnReal(WPARAM wParam,
LPARAM lParam)
1031: {
1032:   int    ch, x, y, z;
1033:   struct _timeb now;
1034:   time_t tmptime;
1035:   int tmpms;
1036:
1037:   _ftime(&now);
1038:
1039:   tmptime=now.time;
1040:   tmpms=now.millitm;
1041:
1042:   //引数よりチャネル番号とＸＹＺ座標を取り出す
1043:   //座標系はカメラから見た形になっており、
1044:   //カメラレンズの先端を (0, 0, 0) とし
1045:   // Ｘは右方向に増加
1046:   // Ｙは上方向に増加
1047:   // Ｚは奥に向かって増加する
1048:   ch  = DE_CH(wParam, lParam) % 4;
1049:   x  = DE_REALX(wParam, lParam);
1050:   y  = DE_REALY(wParam, lParam);
1051:   z  = DE_REALZ(wParam, lParam);
1052:
1053:   m_dx[ch] = x;
1054:   m_dy[ch] = y;
1055:   m_dz[ch] = z;
1056:   m_eventTime[ch]=tmptime;
1057:   m_eventTime_ms[ch]=tmpms;
1058:
1059:   if(m_is_rec && g_is_start == TRUE && ch == 3)
1060:     recording();
1061: }
1062:
1063: void CRecordPlayMidiDlg::OnButtonTest()
1064: {
1065:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
1066:   MMRESULT ret;
1067:   char lfolder[256];
1068:
1069:   UpdateData(TRUE);// ダイアログからデータを取る（TRUE）
1070:   // テンポ取得
1071:   CButton* tempo90big=(CButton*)GetDlgItem(
IDC_TEMPO90_BIG);
1072:   CButton* tempo120big=(CButton*)GetDlgItem(
IDC_TEMPO120_BIG);
1073:   if(tempo90big->GetCheck()) {g_tempo=90;}
1074:   if(tempo120big->GetCheck()) {g_tempo=120;}
1075:
1076:   if(g_nState == IDLE)
1077:   {
1078:     // 診断開始
1079:     if(!m_is_rec)
1080:     {
1081:       // 診断時間エラーチェック
1082:       if(m_nTestSec <= 0 || m_nTestSec > 300)
1083:       {
1084:         MessageBox("秒数は 0-300 の数値を指定して
ください。");
1085:         return;
1086:       }
1087:       // ユーザ名チェック
1088:       if(m_sUserName == "")
1089:       {
1090:         MessageBox("ユーザ名を入力してください。");
1091:         return;
1092:       }
1093:       else
1094:       {
1095:         // フォルダが存在しないなら作成（VC）
1096:         sprintf(lfolder,"%s/%s", DATA_PATH,
m_sUserName);
1097:         if(!IsExistFile(lfolder,true))
1098:         {
1099:           if(!CreateDirectory(lfolder,NULL))
1100:           {
1101:             MessageBox("データ用フォルダが新たに
作成できません。");
1102:             return;
1103:           }
1104:         }
1105:       }
1106:       g_nState = TESTING;
1107:
1108:       // メトロノームボタンを使用不可能に
1109:       CButton* metro=(CButton*)GetDlgItem(
IDC_TEMPO_PLAY);
1110:       metro->EnableWindow(false);
1111:       // リズムパターンボタンを使用不可能に
1112:       enableRhythmButton(false);
1113:
1114:       // 前のイベントを削除
1115:       while(pStart!=NULL)
1116:       {
1117:         pNow=pStart;
1118:         pStart=pStart->pNextEvent;
1119:         delete(pNow);
1120:       }
1121:       pEnd = pStart;
1122:       while(p_calcStart!=NULL)
1123:       {
1124:         p_calcNow=p_calcStart;
1125:         p_calcStart=p_calcStart->pNext;
1126:         // 各動的要素を削除
1127:         delete(p_calcNow->dTime);
1128:         delete(p_calcNow->nLabel);
1129:         delete(p_calcNow->posi_x);
1130:         delete(p_calcNow->posi_y);
1131:         delete(p_calcNow->posi_z);
1132:         // 全体を削除
1133:         delete(p_calcNow);
1134:       }
1135:
1136:       {
1137:         int count=0;
1138:
1139:         p_calcEnd = p_calcStart;
1140:         while(p_barStart!=NULL)
1141:         {
1142:           p_barNow=p_barStart;
1143:           p_barStart=p_barStart->pNextBar;
1144:           // 各動的要素を削除
1145:           delete(p_barNow->dTime);
1146:           delete(p_barNow->Rn);
1147:           delete(p_barNow->Sn);
1148:           delete(p_barNow->dDiffTime);
```

```
1149:           delete(p_barNow->dRelativeTime);
1150:           if(count > 0)
1151:           {
1152:             delete(p_barNow->En);
1153:             delete(p_barNow->Tn);
1154:           }
1155:           // 全体を削除
1156:           delete(p_barNow);
1157:           count++;
1158:         }
1159:         p_barEnd = p_barStart;
1160:       }
1161:
1162:       g_nState = TESTING;
1163:       m_is_rec = TRUE;
1164:       g_is_start = FALSE;
1165:       m_rec.idx = 0;
1166:       m_rec.cycl = m_nTestSec * 15;
1167:
1168:       // MIDI 初期設定
1169:       ret = initMidiLong();
1170:
1171:       m_strRes.Format("Form and MIDI TWIN Recording...
Please play.\r\n");
1172:       UpdateData(false);
1173:
1174:
1175:       g_nTime = -100;
1176:
1177:       SetTimer(ID_TEST_TIMER,10,NULL);
1178:
1179:       GetDlgItem(IDC_BUTTON_TEST)->EnableWindow(FALSE);
1180:       GetDlgItem(IDC_USERNAME)->EnableWindow(FALSE);
1181:
1182:     }
1183:     else
1184:     {
1185:       m_is_rec = FALSE;
1186:       g_is_start = FALSE;
1187:       KillTimer(ID_TEST_TIMER);
1188:     }
1189:   }
1190: }
1191:
1192: void CRecordPlayMidiDlg::recording()
1193: {
1194:   int    i;
1195:
1196:   for(i = 0 ; i < 4 ; i++)  //1 サイクル分まとめて記録
1197:   {
1198:     m_rec.x[i][m_rec.idx] = m_dx[i];
1199:     m_rec.y[i][m_rec.idx] = m_dy[i];
1200:     m_rec.z[i][m_rec.idx] = m_dz[i];
1201:     // by tuzzy
1202:     m_rec.eventTime[i][m_rec.idx] = m_eventTime[i];
1203:     m_rec.eventTime_ms[i][m_rec.idx] = m_eventTime_ms[i];
1204:
1205:   }
1206:   m_rec.idx++;
1207:   if(m_rec.idx >= m_rec.cycl)  //記録終了
1208:   {
1209:     m_is_rec = false;
1210:     g_is_start = false;
1211:     g_nState = IDLE;
1212:     // メトロノーム、開始ボタンを使用可能に
1213:     GetDlgItem(IDC_BUTTON_TEST)->EnableWindow(TRUE);
1214:     GetDlgItem(IDC_TEMPO_PLAY)->EnableWindow(TRUE);
1215:     // リズムパターンボタンを使用可能に
1216:     enableRhythmButton(true);
1217:
1218:     // 砂時計
1219:     BeginWaitCursor();
1220:
1221:     // 終わりの合図
1222:     m_strRes += "Analyzing...\r\n";
1223:     UpdateData(false);
1224:     wavePlay(WAVE_PATH + "/bunsekisiteimasu.wav");
1225:
1226:     // フォーム／演奏生データの保存
1227:     autoSaveForm();
1228:     autoSaveMidi();
1229:
1230:     // フォームデータを一本化
1231:     formCustumize();
1232:     autoSaveCForm();
1233:
1234:     // 郡への分割
1235:     form2Group();
1236:     // 小節を構成
1237:     form2Bar();
1238:
1239:     // クラスター分析をして小節をカスタマイズ
1240:     {
1241:       int ret;
1242:       if((ret = clusterBarLine())==0)
1243:       {
1244:         drawYourLength();
1245:       }
1246:       else
1247:       {
1248:         m_strRes += "分析に失敗しました。\r\n もう一度やり直し
てください。\r\n";
1249:         UpdateData(false);
1250:         return;
1251:       }
1252:     }
1253:
1254:     // 小節をファイルへ保存、小節から各値の計算
1255:     printBarLine();
1256:     // フォーム誤差のフィードバック
1257:     drawStrokeError();
1258:     // ベロシティのフィードバック
1259:     drawVelocity();
1260:     // ふり幅のフィードバック
1261:     drawFurihaba();
1262:
1263:     // 演奏終わり
1264:     g_bIsAfter = 1;
1265:
1266:     // g_meanEn,g_meanSHIFT から音声によるフィードバック
1267:     waveFeedBack();
1268:
1269:     // 統計用データの保存
1270:     autoSaveTotal();
1271:
1272:     // 統計用個人データの保存
1273:     autoSaveSelfLog();
1274:
1275:     // 砂時計終了
1276:     EndWaitCursor();
1277:
1278:     // 回数をインクリメント
1279:     g_nPerformTime[m_nSelectedPattern-1]++;
1280:     m_nPerformTime = g_nPerformTime[m_nSelectedPattern-1];
1281:     UpdateData(false);
1282:
1283:     // コメント表示を最下段にスクロール
1284:     m_ctrlstrRes.LineScroll(m_ctrlstrRes.GetLineCount());
1285:   }
1286: }
1287:
1288: MMRESULT CRecordPlayMidiDlg::initMidiLong()
1289: {
1290:   // ***************** my code *****************
1291:   MMRESULT ret;
1292:   BYTE bBuf[]={0xF0, 0x7E, 0x00, 0x06, 0x01, 0xF7};
1293:
1294:   midiInReset(gs_hMidiIn);
1295:   midiInStart(gs_hMidiIn);
1296:
1297:   gs_pMidiOutHdr->lpData = (char *)bBuf;
1298:   gs_pMidiOutHdr->dwBufferLength = sizeof(bBuf);
1299:   ret=midiOutLongMsg(gs_hMidiOut, gs_pMidiOutHdr,
sizeof(MIDIHDR));
1300:
1301:   return ret;
1302: }
1303:
1304: void CRecordPlayMidiDlg::autoSaveMidi()
1305: {
1306:   FILE*  fp;
1307:   time_t  tim;
1308:   char lfilename[100];
1309:
1310:   time(&tim);
1311:   CTime  ctim(tim);
1312:
1313:   sprintf(lfilename,"%s/%s/%d_%d_%dmidi", DATA_PATH,
m_sUserName, g_tempo, m_nSelectedPattern,
g_nPerformTime[m_nSelectedPattern-1]);
1314:   fp = fopen(lfilename, "w");
1315:
1316:   fprintf(fp, "DATA: diffTime(sec) 9nCH notenumber
velocity eventTime(sec) eventTime(ms)\n");
1317:
1318:   pNow=pStart;
1319:   while(pNow != NULL)
1320:   {
```

```
1321:     fprintf(fp, "%f 99 %2d %2d %d %3d\n",
pNow->dDiffTime, pNow->nNoteNumber, pNow->nVelocity,
pNow->eventTime, pNow->eventTime_ms);
1322:     pNow = pNow->pNextEvent;
1323:   }
1324:   fclose(fp);
1325: }
1326:
1327: void CRecordPlayMidiDlg::autoSaveForm()
1328: {
1329:   FILE*  fp;
1330:   int    i;
1331:   time_t  tim;
1332:   char lfilename[100];
1333:
1334:   time(&tim);
1335:   CTime  ctim(tim);
1336:
1337:   sprintf(lfilename,"%s/%s/%d_%d_%dform", DATA_PATH,
m_sUserName, g_tempo, m_nSelectedPattern,
g_nPerformTime[m_nSelectedPattern-1]);
1338:   fp = fopen(lfilename, "w");
1339:   fprintf(fp, "%s\n", (m_is_real) ?
("3Dimension-Data") : ("DigitEye3D Original-Data"));
1340:   fprintf(fp, "%d/%2d/%2d %2d:%2d\n",
ctim.GetYear(), ctim.GetMonth(), ctim.GetDay(),
ctim.GetHour(), ctim.GetMinute());
1341:   fprintf(fp, "Sampling-Time %d sec ( %d cycles )\n",
m_rec.cycl / 15, m_rec.cycl);
1342:   if(m_is_real)
1343:     fprintf(fp, "DATA: x0 y0 z0   x1 y1 z1
   x2 y2 z2   x3 y3 z3\n");
1344:   else
1345:     fprintf(fp, "DATA: x0 y0 s0   x1 y1 s1
   x2 y2 s2   x3 y3 s3\n");
1346:
1347:   for(i = 0 ; i < m_rec.cycl ; i++)
1348:   {
1349:     if(m_is_real)
1350:       fprintf(fp, "%d %3d %+5d %+5d %+5d   %d %3d %+5d
 %+5d %+5d   %d %3d %+5d %+5d %+5d   %d %3d %+5d %+5d %+5d\n",
1351:         m_rec.eventTime[0][i], m_rec.eventTime_ms[0][i],
m_rec.x[0][i], m_rec.y[0][i], m_rec.z[0][i],
1352:         m_rec.eventTime[1][i], m_rec.eventTime_ms[1][i],
m_rec.x[1][i], m_rec.y[1][i], m_rec.z[1][i],
1353:         m_rec.eventTime[2][i], m_rec.eventTime_ms[2][i],
m_rec.x[2][i], m_rec.y[2][i], m_rec.z[2][i],
1354:         m_rec.eventTime[3][i], m_rec.eventTime_ms[3][i],
m_rec.x[3][i], m_rec.y[3][i], m_rec.z[3][i]);
1355:     else
1356:       fprintf(fp, "%d %3d %4d %4d %5d   %d %3d %4d %4d
 %5d   %d %3d %4d %4d %5d   %d %3d %4d %4d %5d\n",
1357:         m_rec.eventTime[0][i], m_rec.eventTime_ms[0][i],
m_rec.x[0][i], m_rec.y[0][i], m_rec.z[0][i],
1358:         m_rec.eventTime[1][i], m_rec.eventTime_ms[1][i],
m_rec.x[1][i], m_rec.y[1][i], m_rec.z[1][i],
1359:         m_rec.eventTime[2][i], m_rec.eventTime_ms[2][i],
m_rec.x[2][i], m_rec.y[2][i], m_rec.z[2][i],
1360:         m_rec.eventTime[3][i], m_rec.eventTime_ms[3][i],
m_rec.x[3][i], m_rec.y[3][i], m_rec.z[3][i]);
1361:   }
1362:   fclose(fp);
1363: }
1364:
1365: // Form データと MIDI データを基に、周期データを作成
1366: void CRecordPlayMidiDlg::form2Group()
1367: {
1368:   int from, to, count, nVelocity;
1369:   double dStartTime;
1370:
1371:   from = -10;
1372:   nVelocity = -1000;
1373:   to = 0;
1374:   count = 1;
1375:
1376:   p_calcStart = NULL;
1377:   p_calcEnd = p_calcStart;
1378:   p_calcNow = p_calcStart;
1379:
1380:   pNow=pStart;
1381:   while(pNow != NULL && to < m_nTestSec * 60)
1382:   {
1383:     if(pNow->dDiffTime < m_simple.dDiffTime[to]
1384:       && to >= g_nExtendWidth
1385:       && to < m_nTestSec * 60 - g_nExtendWidth)
1386:     {
1387:       // 郡を作成。配列を追加
1388:       if(from >= 0)
1389:       {
1390:         // form～(to-1)
1391:         makeGroup(from, to - 1, count, dStartTime,
 (double)pNow->dDiffTime, nVelocity);
1392:         count++;
1393:       }
1394:
1395:       dStartTime = (double)pNow->dDiffTime;
1396:       nVelocity = pNow->nVelocity;
1397:       from = to;
1398:       pNow = pNow->pNextEvent;
1399:     }
1400:     to++;
1401:   }
1402: }
1403:
1404: void CRecordPlayMidiDlg::formCustumize()
1405: {
1406:   int i,j,count,justChanged;
1407:   count=0;
1408:   CString tmp;
1409:   justChanged=0;
1410:   // データを移す
1411:   for(i = 0 ; i < m_rec.cycl ; i++)
1412:   {
1413:     for(j=0;j<4;j++)
1414:     {
1415:       m_simple.x[count]=m_rec.x[j][i];
1416:       m_simple.y[count]=m_rec.y[j][i];
1417:       m_simple.z[count]=m_rec.z[j][i];
1418:       m_simple.eventTime[count]=m_rec.eventTime[j][i];
1419:       m_simple.eventTime_ms[count]=m_rec.eventTime_ms[j][i];
1420:       m_simple.dDiffTime[count] = (double)m_rec.eventTime
[j][i] - (double)pStart->eventTime;
1421:       m_simple.dDiffTime[count] = m_simple.dDiffTime[count]
 + (double)m_rec.eventTime_ms[j][i] / 1000
 - (double)pStart->eventTime_ms / 1000;
1422:       m_simple.dDiffTime[count] =
1423:         m_simple.dDiffTime[count] - g_dOffset;
1424:
1425:       // ここで時刻補正（補正直後でなく、i==0 でなく、2 点の時間
差が 0.012sec 以下ならば補正）
1426:       if(justChanged == 0 && count>0 && (
m_simple.dDiffTime[count] - m_simple.dDiffTime[count-1]) < 0.012)
1427:       {
1428:         m_simple.dDiffTime[count] = m_simple.dDiffTime
[count] + ((double)1 / (double)300);
1429:         m_simple.dDiffTime[count-1] = m_simple.dDiffTime
[count-1] - ((double)1 / (double)300);
1430:         justChanged = 1;// 補正直後の印
1431:       }
1432:       else if(justChanged == 1)
1433:       {
1434:         justChanged = 0;
1435:       }
1436:       count++;
1437:     }
1438:   }
1439:   // [1]-[0] と [count-1]-[count-2] に補正... すべて間が均等
になるように
1440:   if((m_simple.dDiffTime[1] - m_simple.dDiffTime[0])
 > 0.018)
1441:   {
1442:     m_simple.dDiffTime[0] = m_simple.dDiffTime[0]
 + ((double)1 / (double)300);
1443:   }
1444:   if((m_simple.dDiffTime[count-1]
 - m_simple.dDiffTime[count-2]) > 0.018)
1445:   {
1446:     m_simple.dDiffTime[count-1] =
m_simple.dDiffTime[count-1] - ((double)1 / (double)300);
1447:   }
1448: }
1449:
1450:
1451:
1452: void CRecordPlayMidiDlg::makeGroup(int from,
int tominusone, int count, double startTime,
double endTime, int velocity)
1453: {
1454:   int i;
1455:
1456:   p_calcNow = new positionForCalc;
1457:
1458:   p_calcNow->number = count;          // n 周期目
1459:   p_calcNow->nData = tominusone + 1 - from;  // データ個数
1460:   p_calcNow->dStartTime = startTime;      // 郡の先頭打拍時
刻
1461:   p_calcNow->dEndTime = endTime;        // 郡の終了打拍時刻
（次の先頭打拍時刻）
```

```
1462:   p_calcNow->nVelocity = velocity;      // Velocity
1463:
1464:   p_calcNow->dTime = new double[tominusone+1-from];
1465:   p_calcNow->nLabel = new int[tominusone+1-from];
1466:   p_calcNow->posi_x = new int[tominusone+1-from];
1467:   p_calcNow->posi_y = new int[tominusone+1-from];
1468:   p_calcNow->posi_z = new int[tominusone+1-from];
1469:   // かかないと NULL と見てもらえない
1470:   p_calcNow->pNext = NULL;
1471:
1472:   if(p_calcNow->posi_x == NULL ||
1473:     p_calcNow->posi_y == NULL ||
 p_calcNow->posi_z == NULL)
1474:   {
1475:     MessageBox("メモリ割り当てエラー");
1476:     return;
1477:   }
1478:
1479:   for(i=from;i<=tominusone;i++)
1480:   {
1481:     p_calcNow->dTime[i-from] = m_simple.dDiffTime[i];
1482:     p_calcNow->posi_x[i-from] = m_simple.x[i];
1483:     p_calcNow->posi_y[i-from] = m_simple.y[i];
1484:     p_calcNow->posi_z[i-from] = m_simple.z[i];
1485:     p_calcNow->nLabel[i-from] = i;
1486:   }
1487:
1488:   // ポインタのつなぎ合わせ
1489:   if(p_calcStart == NULL && p_calcEnd == NULL)
1490:   {
1491:     p_calcStart = p_calcNow;
1492:     p_calcEnd = p_calcNow;
1493:   }
1494:   else if(p_calcStart != NULL && p_calcEnd != NULL)
1495:   {
1496:     p_calcEnd->pNext = p_calcNow;
1497:     p_calcEnd = p_calcEnd->pNext;
1498:   }
1499:   else
1500:   {
1501:     MessageBox("不明なエラー");
1502:     exit(0);
1503:   }
1504: }
1505:
1506: // Vn を計算する
1507: void CRecordPlayMidiDlg::calcVn()
1508: {
1509:   double tmp=0;
1510:   int count=0;
1511:
1512:   pNow=pStart;
1513:   while(pNow != NULL)
1514:   {
1515:     tmp += pNow->nVelocity;
1516:     count++;
1517:     pNow = pNow->pNextEvent;
1518:   }
1519:
1520:   g_meanSerialVn = tmp / count;
1521:
1522:   tmp=0;
1523:   pNow=pStart;
1524:   while(pNow != NULL)
1525:   {
1526:     tmp += pow((pNow->nVelocity - g_meanSerialVn),2);
1527:     pNow = pNow->pNextEvent;
1528:   }
1529:
1530:   g_sdSerialVn = sqrt(tmp / count);
1531: }
1532:
1533: // Tn を計算する
1534: void CRecordPlayMidiDlg::calcTn()
1535: {
1536:   FILE * fp;
1537:   double tmpTime;
1538:   double tmpTimeminus=0;
1539:   double tmpDeltaTime=0;
1540:   double tmpDeltaTimeminus=0;
1541:   double tmpTn=0;
1542:   double tmpTnminus=0;
1543:   double tmpAllTn=0;
1544:   double tmpsd=0;
1545:   int count=0;
1546:   time_t StartTime;
1547:   double StartTime_ms;
1548:   char tmp[100];
1549:   char lfilename[256];
1550:
1551:   sprintf(lfilename,"%s/%s/%d_%d_%dTn", DATA_PATH,
 m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1]);
1552:   fp = fopen(lfilename,"w");
1553:
1554:   pNow=pStart;
1555:   StartTime = pNow->eventTime;
1556:   StartTime_ms = (double)pNow->eventTime_ms / 1000;
1557:   while(pNow != NULL)
1558:   {
1559:     tmpTime = pNow->eventTime - StartTime
 + (double)pNow->eventTime_ms / 1000;
1560:     tmpTime = tmpTime - StartTime_ms;
1561:
1562:     if(count > 0)
1563:     {
1564:       tmpDeltaTime = tmpTime - tmpTimeminus;
1565:       if(count > 1)
1566:       {
1567:         tmpTn = (tmpDeltaTime - tmpDeltaTimeminus)*2
 / (tmpDeltaTime + tmpDeltaTimeminus);
1568:         if(tmpTn < 0)
1569:           tmpTn = tmpTn * (-1);
1570:         tmpAllTn += tmpTn;
1571:         fprintf(fp,"%d\t%.5f\n", count-2, tmpTn);
1572:         sprintf(tmp,"Tn[%d]=%.5f\r\n", count, tmpTn);
1573:         m_strRes += tmp;
1574:         UpdateData(false);
1575:       }
1576:       tmpDeltaTimeminus = tmpDeltaTime;
1577:     }
1578:     tmpTimeminus = tmpTime;
1579:     count++;
1580:
1581:     pNow = pNow->pNextEvent;
1582:   }
1583:
1584:   g_meanSerialTn = tmpAllTn / (count-2);
1585:
1586:   count = 0;
1587:   pNow=pStart;
1588:   while(pNow != NULL)
1589:   {
1590:     tmpTime = pNow->eventTime - StartTime
 + (double)pNow->eventTime_ms / 1000;
1591:     tmpTime = tmpTime - StartTime_ms;
1592:
1593:     if(count > 0)
1594:     {
1595:       tmpDeltaTime = tmpTime - tmpTimeminus;
1596:       if(count > 1)
1597:       {
1598:         tmpTn = (tmpDeltaTime - tmpDeltaTimeminus)*2
 / (tmpDeltaTime + tmpDeltaTimeminus);
1599:         if(tmpTn < 0)
1600:           tmpTn = tmpTn * (-1);
1601:
1602:         tmpsd += pow(tmpTn - g_meanSerialTn,2);
1603:       }
1604:       tmpDeltaTimeminus = tmpDeltaTime;
1605:     }
1606:     tmpTimeminus = tmpTime;
1607:     count++;
1608:
1609:     pNow = pNow->pNextEvent;
1610:   }
1611:
1612:   g_sdSerialTn = sqrt(tmpsd / (count - 2));
1613:
1614:   fprintf(fp,"mean\tsd\t%.6f\t%6f\n", g_meanSerialTn,
 g_sdSerialTn);
1615:   sprintf(tmp,"meanTn=%.5f sdTn=%.5f\r\n",
 g_meanSerialTn, g_sdSerialTn);
1616:   m_strRes += tmp;
1617:   UpdateData(false);
1618:
1619:   fclose(fp);
1620: }
1621:
1622: // En を計算する
1623: void CRecordPlayMidiDlg::calcEn()
1624: {
1625:   FILE * fp;
1626:   FILE * fp_en;
1627:   positionForCalc* lp_calcPre = p_calcStart;
1628:   positionForCalc* lp_tmp = p_calcStart;
1629:   double * lp_En;
1630:   double * lp_x;
```

```
1631:   double * lp_y;
1632:   double * lp_z;// 補間した導関数用
1633:   double * lp_time;
1634:   double * lp_posi_x;
1635:   double * lp_posi_y;
1636:   double * lp_posi_z;
1637:   // 座標用
1638:   double dTDash;
1639:   double dXnMinus, dYnMinus, dZnMinus;
1640:   double dBarXn, dBarYn, dBarZn;
1641:   double tmpSumOfNumerator, tmpSumOfDenominator;
1642:   double deltaNumerator, deltaDenominator;
1643:   int i, j, nEn=0;
1644:   int count=0;
1645:   int n=2;
1646:   char tmp[100];
1647:   char lfilename[256];
1648:   char lfilename_en[256];
1649:   // 計算時間用
1650:   struct _timeb now;
1651:   time_t tmpStartTime;
1652:   int tmpStartms;
1653:   time_t tmpEndTime;
1654:   int tmpEndms;
1655:
1656:   // by tuzzy
1657:   _ftime(&now);
1658:
1659:   tmpStartTime=now.time;
1660:   tmpStartms=now.millitm;
1661:
1662:   p_calcNow=p_calcStart;
1663:
1664:   if(p_calcNow == NULL)
1665:   {
1666:     return;
1667:   }
1668:   else
1669:   {
1670:     p_calcNow = p_calcNow->pNext;
1671:   }
1672:
1673:   // En 格納用
1674:   while(lp_tmp != NULL)
1675:   {
1676:     nEn++;
1677:     lp_tmp = lp_tmp->pNext;
1678:   }
1679:   lp_En = new double[nEn];
1680:
1681:   sprintf(lfilename,"%s/%s/%d_%d_%dhokan", DATA_PATH,
 m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1]);
1682:   sprintf(lfilename_en,"%s/%s/%d_%d_%dEn", DATA_PATH,
 m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1]);
1683:   fp = fopen(lfilename, "w");
1684:   fp_en = fopen(lfilename_en, "w");
1685:
1686:   g_meanSerialEn = 0;
1687:   g_sdSerialEn = 0;
1688:   while(p_calcNow != NULL)
1689:   {
1690:     // Bar の計算
1691:     dBarXn=0;dBarYn=0;dBarZn=0;
1692:     for(i=0;i<p_calcNow->nData;i++)
1693:     {
1694:       dBarXn += (double)p_calcNow->posi_x[i];
1695:       dBarYn += (double)p_calcNow->posi_y[i];
1696:       dBarZn += (double)p_calcNow->posi_z[i];
1697:     }
1698:     dBarXn = dBarXn / p_calcNow->nData;
1699:     dBarYn = dBarYn / p_calcNow->nData;
1700:     dBarZn = dBarZn / p_calcNow->nData;
1701:
1702:     fprintf(fp,"%d 番目のグループ（時刻%.5f〜%.5f[sec
]）\n[barxn baryn barzn] = %.3f %.3f %.3f\n",
1703:       n++, p_calcNow->dStartTime, p_calcNow->dEndTime,
1704:       dBarXn,dBarYn,dBarZn);
1705:
1706:     // double 型配列の作成
1707:     lp_time = new double[lp_calcPre->nData+2
 * g_nExtendWidth];
1708:     lp_posi_x = new double[lp_calcPre->nData+2
 * g_nExtendWidth];
1709:     lp_posi_y = new double[lp_calcPre->nData+2
 * g_nExtendWidth];
1710:     lp_posi_z = new double[lp_calcPre->nData+2
 * g_nExtendWidth];
1711:     for(i=0;i<lp_calcPre->nData+2*g_nExtendWidth;i++)
1712:     {
1713:       // lp_calcPre の範囲± g_nExtendWidth の範囲のラベルを
取得
1714:       j=i+lp_calcPre->nLabel[0]-g_nExtendWidth;
1715:
1716:       // ラベルが取得位置データの範囲を超えないかチェック
1717:       if(j < 0)
1718:       {
1719:         MessageBox("LabelError in calcEn(low)");
1720:         exit(0);
1721:       }
1722:       else if(j > (m_rec.cycl * 4))
1723:       {
1724:         MessageBox("LabelError in calcEn(high)");
1725:         exit(0);
1726:       }
1727:       // その周期の範囲± g_nExtendWidth の範囲の時刻、位置を
取得
1728:       lp_time[i] = m_simple.dDiffTime[j];
1729:       lp_posi_x[i] = m_simple.x[j];
1730:       lp_posi_y[i] = m_simple.y[j];
1731:       lp_posi_z[i] = m_simple.z[j];
1732:     }
1733:     // 対象となる、波形を比較する 2 郡を取得
1734:     // x,y,z 軸において、各差をスプライン補間により計算
1735:     lp_x = new double[lp_calcPre->nData+2*g_nExtendWidth];
1736:     lp_y = new double[lp_calcPre->nData+2*g_nExtendWidth];
1737:     lp_z = new double[lp_calcPre->nData+2*g_nExtendWidth];
1738:
1739:     // n-1 周期目の波形を補間
1740:     splineMakeTable(lp_calcPre->nData+2*g_nExtendWidth,
1741:       lp_time,
1742:       lp_posi_x,
1743:       lp_x);
1744:     splineMakeTable(lp_calcPre->nData+2*g_nExtendWidth,
1745:       lp_time,
1746:       lp_posi_y,
1747:       lp_y);
1748:     splineMakeTable(lp_calcPre->nData+2*g_nExtendWidth,
1749:       lp_time,
1750:       lp_posi_z,
1751:       lp_z);
1752:
1753:     tmpSumOfNumerator=0;
1754:     tmpSumOfDenominator=0;
1755:     deltaNumerator=0;
1756:     deltaDenominator=0;
1757:
1758:     fprintf(fp, "番号 元時刻 t [x y z] dTDash [dXnMinus
 dYnMinus dZnMinus] Δ nume Δ deno\n");
1759:     sprintf(tmp,"dTDash dXnMinus dYnMinus dZnMinus\r\n");
1760:     m_strRes += tmp;
1761:     UpdateData(false);
1762:
1763:     for(i=0;i<p_calcNow->nData;i++)
1764:     {
1765:       // t' の計算
1766:       dTDash = splineMakeTDash(
1767:         p_calcNow->dEndTime - p_calcNow->dStartTime,
1768:         p_calcNow->dTime[i],
1769:         p_calcNow->dEndTime,
1770:         p_calcNow->dStartTime,
1771:         lp_calcPre->dStartTime);
1772:
1773:       // Xn-1,Yn-1,Zn-1 の計算
1774:       dXnMinus = splineSpline(dTDash,
1775:         lp_calcPre->nData+2*g_nExtendWidth,
1776:         lp_time,
1777:         lp_posi_x,
1778:         lp_x);
1779:       dYnMinus = splineSpline(dTDash,
1780:         lp_calcPre->nData+2*g_nExtendWidth,
1781:         lp_time,
1782:         lp_posi_y,
1783:         lp_y);
1784:       dZnMinus = splineSpline(dTDash,
1785:         lp_calcPre->nData+2*g_nExtendWidth,
1786:         lp_time,
1787:         lp_posi_z,
1788:         lp_z);
1789:
1790:       deltaNumerator =
1791:         pow(p_calcNow->posi_x[i] - dXnMinus, 2) +
1792:         pow(p_calcNow->posi_y[i] - dYnMinus, 2) +
1793:         pow(p_calcNow->posi_z[i] - dZnMinus, 2);
1794:       tmpSumOfNumerator += deltaNumerator;
1795:
1796:       deltaDenominator =
```

```
1797:          pow(p_calcNow->posi_x[i] - dBarXn, 2) +
1798:          pow(p_calcNow->posi_y[i] - dBarYn, 2) +
1799:          pow(p_calcNow->posi_z[i] - dBarZn, 2);
1800:        tmpSumOfDenominator += deltaDenominator;
1801:
1802:        // log
1803:        fprintf(fp,"%d 番目：t=%.5f [%d %d %d] t'=
 %.5f [%.3f %.3f %.3f] %.5f %.5f\n",
1804:          i, p_calcNow->dTime[i],
1805:          p_calcNow->posi_x[i], p_calcNow->posi_y[i],
 p_calcNow->posi_z[i],
1806:          dTDash, dXnMinus, dYnMinus, dZnMinus,
1807:          deltaNumerator, deltaDenominator);
1808:
1809:        sprintf(tmp,"%.3f %.3f %.3f %.3f\r\n",
1810:          dTDash, dXnMinus, dYnMinus, dZnMinus);
1811:        m_strRes += tmp;
1812:        UpdateData(false);
1813:
1814:      }
1815:
1816:      // 各 En を計算
1817:      lp_En[count++] = tmpSumOfNumerator
 / tmpSumOfDenominator;
1818:      g_meanSerialEn += tmpSumOfNumerator
 / tmpSumOfDenominator;
1819:
1820:      fprintf(fp,"En(%d) = %.5f / %.5f = %.5f\n",
1821:        count-1, tmpSumOfNumerator, tmpSumOfDenominator,
1822:        lp_En[count-1]);
1823:      fprintf(fp_en,"%d\t%.6f\n", count-1,
 lp_En[count-1]);
1824:      sprintf(tmp,"lp_En[%d]=%.6f\r\n",count-1,
 lp_En[count-1]);
1825:      m_strRes += tmp;
1826:      UpdateData(false);
1827:
1828:      delete(lp_x);
1829:      delete(lp_y);
1830:      delete(lp_z);
1831:      delete(lp_posi_x);
1832:      delete(lp_posi_y);
1833:      delete(lp_posi_z);
1834:      p_calcNow = p_calcNow->pNext;
1835:      lp_calcPre = lp_calcPre->pNext;
1836:    }
1837:
1838:    // En の平均値を計算
1839:    g_meanSerialEn = g_meanSerialEn / count;
1840:
1841:    // En の標準偏差を計算
1842:    for(i=0;i<count;i++)
1843:    {
1844:      g_sdSerialEn = g_sdSerialEn + pow(lp_En[i]
 - g_meanSerialEn,2);
1845:    }
1846:    g_sdSerialEn = sqrt(g_sdSerialEn / count);
1847:
1848:    fprintf(fp,"meanEn = %.6f gdEn = %.6f\n",
 g_meanSerialEn, g_sdSerialEn);
1849:    fprintf(fp_en,"mean\tsd\t%.6f\t%.6f\n",
 g_meanSerialEn, g_sdSerialEn);
1850:    // by tuzzy
1851:    _ftime(&now);
1852:    tmpEndTime=now.time;
1853:    tmpEndms=now.millitm;
1854:    fprintf(fp,"En 計算時間 = %d[sec]\n",
 tmpEndTime - tmpStartTime);
1855:
1856:    fclose(fp);
1857:    fclose(fp_en);
1858:
1859:    sprintf(tmp,"En 計算時間 = %d[sec]\r\n",
 tmpEndTime - tmpStartTime);
1860:    m_strRes += tmp;
1861:    UpdateData(false);
1862: }
1863:
1864: // positionForCalc 型の 2 つの構造体を対象に En を計算する
1865: // スプライン補間分割バージョン
1866: double CRecordPlayMidiDlg::calcOneEnAnother(
char * before, char * after)
1867: {
1868:    positionForCalc * lp_before =
 (positionForCalc *)before;
1869:    positionForCalc * lp_after =
 (positionForCalc *)after;
1870:
1871:    double * lp_x;
1872:    double * lp_y;
1873:    double * lp_z;// 補間した導関数用
1874:    double * lp_time;
1875:    double * lp_posi_x;
1876:    double * lp_posi_y;
1877:    double * lp_posi_z;
1878:
1879:    char lfilename[256];
1880:
1881:    // 座標用
1882:    int nCenterLabel;
1883:    double dTDash;
1884:    double dXnMinus, dYnMinus, dZnMinus;
1885:    double dBarXn, dBarYn, dBarZn;
1886:    double tmpSumOfNumerator, tmpSumOfDenominator;
1887:    double deltaNumerator, deltaDenominator;
1888:
1889:    int i,j,k;
1890:    FILE* fp;
1891:
1892:    sprintf(lfilename,"%s/%s/%d_%d_%dOneEn", DATA_PATH,
 m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1]);
1893:    fp=fopen(lfilename,"a");
1894:
1895:    // Bar の計算
1896:    dBarXn=0;dBarYn=0;dBarZn=0;
1897:    for(i=0;i<lp_after->nData;i++)
1898:    {
1899:      dBarXn += (double)lp_after->posi_x[i];
1900:      dBarYn += (double)lp_after->posi_y[i];
1901:      dBarZn += (double)lp_after->posi_z[i];
1902:    }
1903:    dBarXn = dBarXn / lp_after->nData;
1904:    dBarYn = dBarYn / lp_after->nData;
1905:    dBarZn = dBarZn / lp_after->nData;
1906:
1907:    fprintf(fp,"◇◇◇後◇◇◇%d 番目のグループ
（時刻%.5f～%.5f[sec]）\n[barxn baryn barzn]
 = %.3f %.3f %.3f\n",
1908:      lp_after->number, lp_after->dStartTime,
 lp_after->dEndTime,
1909:      dBarXn,dBarYn,dBarZn);
1910:    fprintf(fp,"◆◆◆前◆◆◆%d 番目のグループ
（時刻%.5f～%.5f[sec]）\n",
1911:      lp_before->number, lp_before->dStartTime,
 lp_before->dEndTime);
1912:
1913:    // double 型配列の作成
1914:    lp_time = new double[2*g_nExtendWidth];
1915:    lp_posi_x = new double[2*g_nExtendWidth];
1916:    lp_posi_y = new double[2*g_nExtendWidth];
1917:    lp_posi_z = new double[2*g_nExtendWidth];
1918:    // x,y,z 軸において、各差をスプライン補間により計算
1919:    lp_x = new double[2*g_nExtendWidth];
1920:    lp_y = new double[2*g_nExtendWidth];
1921:    lp_z = new double[2*g_nExtendWidth];
1922:
1923:    tmpSumOfNumerator=0;
1924:    tmpSumOfDenominator=0;
1925:    deltaNumerator=0;
1926:    deltaDenominator=0;
1927:
1928:    fprintf(fp, "番号 元時刻 t [x y z] dTDash
 [dXnMinus dYnMinus dZnMinus] Δ nume Δ deno\n");
1929:
1930:    for(i=0;i<lp_after->nData;i++)
1931:    {
1932:      // t' の計算
1933:      dTDash = splineMakeTDashDiscrete(
1934:        lp_after->dTime[i],
1935:        lp_after->dEndTime,
1936:        lp_after->dStartTime,
1937:        lp_before->dEndTime,
1938:        lp_before->dStartTime
1939:        );
1940:
1941:      // ---------------------------------------
1942:      // ここで分割したスプラインテーブルを作成
1943:      // ---------------------------------------
1944:      // 中心となるラベルを求める（0 <= nCenter
Label <= lp_before->nData)
1945:      nCenterLabel = (int)((double)lp_before->nData
 * (double)i / (double)lp_after->nData);
1946:
1947:      fprintf(fp,"nCenterLabel = %d\n",nCenterLabel);
1948:
1949:      // そのラベルから時刻、座標の配列を作成
1950:      for(k=0;k<2*g_nExtendWidth;k++)
```

```
1951:     {
1952:       // nCenterLabel ± g_nExtendWidth の範囲のラベルを取得
1953:       j=nCenterLabel+k+lp_before->nLabel[0]
 - g_nExtendWidth;
1954:       // ラベルが取得位置データの範囲を超えないかチェック
1955:       if(j < 0)
1956:       {
1957:         MessageBox("LabelError in calcEn(low)");
1958:         exit(0);
1959:       }
1960:       else if(j > (m_rec.cycl * 4))
1961:       {
1962:         MessageBox("LabelError in calcEn(high)");
1963:         exit(0);
1964:       }
1965:       // その周期の範囲± g_nExtendWidth の範囲の時刻、位置を
取得
1966:       lp_time[k] = m_simple.dDiffTime[j];
1967:       lp_posi_x[k] = m_simple.x[j];
1968:       lp_posi_y[k] = m_simple.y[j];
1969:       lp_posi_z[k] = m_simple.z[j];
1970:     }
1971:
1972:     // n-1 周期目の波形の一部を補間（スプラインテーブルの作成）
1973:     splineMakeTable(2*g_nExtendWidth,
1974:       lp_time,
1975:       lp_posi_x,
1976:       lp_x);
1977:     splineMakeTable(2*g_nExtendWidth,
1978:       lp_time,
1979:       lp_posi_y,
1980:       lp_y);
1981:     splineMakeTable(2*g_nExtendWidth,
1982:       lp_time,
1983:       lp_posi_z,
1984:       lp_z);
1985:     // ------------------------------------------------
1986:
1987:
1988:     // Xn-1,Yn-1,Zn-1 の計算
1989:     dXnMinus = splineSpline(dTDash,
1990:       2*g_nExtendWidth,
1991:       lp_time,
1992:       lp_posi_x,
1993:       lp_x);
1994:     dYnMinus = splineSpline(dTDash,
1995:       2*g_nExtendWidth,
1996:       lp_time,
1997:       lp_posi_y,
1998:       lp_y);
1999:     dZnMinus = splineSpline(dTDash,
2000:       2*g_nExtendWidth,
2001:       lp_time,
2002:       lp_posi_z,
2003:       lp_z);
2004:
2005:     deltaNumerator =
2006:       pow(lp_after->posi_x[i] - dXnMinus, 2) +
2007:       pow(lp_after->posi_y[i] - dYnMinus, 2) +
2008:       pow(lp_after->posi_z[i] - dZnMinus, 2);
2009:     tmpSumOfNumerator += deltaNumerator;
2010:
2011:     deltaDenominator =
2012:       pow(lp_after->posi_x[i] - dBarXn, 2) +
2013:       pow(lp_after->posi_y[i] - dBarYn, 2) +
2014:       pow(lp_after->posi_z[i] - dBarZn, 2);
2015:     tmpSumOfDenominator += deltaDenominator;
2016:
2017:     fprintf(fp,"%d 番目：t=%.5f [%d %d %d] t'=
 %.5f [%.3f %.3f %.3f] %.5f %.5f\n",
2018:       i, lp_after->dTime[i],
2019:       lp_after->posi_x[i], lp_after->posi_y[i],
 lp_after->posi_z[i],
2020:       dTDash, dXnMinus, dYnMinus, dZnMinus,
2021:       deltaNumerator, deltaDenominator);
2022:   }
2023:
2024:   fclose(fp);
2025:   // 各 En を計算
2026:   return tmpSumOfNumerator/tmpSumOfDenominator;
2027: }
2028:
2029: void CRecordPlayMidiDlg::splineMakeTable(int N,
 double x[], double y[], double z[])
2030: {
2031:   int i;
2032:   double t;
2033:   double * h;
2034:   double * d;
2035:   h = new double[N];
2036:   d = new double[N];
2037:
2038:   z[0] = 0;  z[N - 1] = 0;  /* 両端点での y''(x) / 6 */
2039:   for (i = 0; i < N - 1; i++) {
2040:     h[i  ] =  x[i + 1] - x[i];
2041:     d[i + 1] = (y[i + 1] - y[i]) / h[i];
2042:   }
2043:   z[1] = d[2] - d[1] - h[0] * z[0];
2044:   d[1] = 2 * (x[2] - x[0]);
2045:   for (i = 1; i < N - 2; i++) {
2046:     t = h[i] / d[i];
2047:     z[i + 1] = d[i + 2] - d[i + 1] - z[i] * t;
2048:     d[i + 1] = 2 * (x[i + 2] - x[i]) - h[i] * t;
2049:   }
2050:   z[N - 2] -= h[N - 2] * z[N - 1];
2051:   for (i = N - 2; i > 0; i--)
2052:     z[i] = (z[i] - h[i] * z[i + 1]) / d[i];
2053: }
2054:
2055: double CRecordPlayMidiDlg::splineMakeTDash(
double D, double t, double zero, double one, double two)
2056: {
2057:   if(D == 0)
2058:     return 0;
2059:   return two*(zero - t)/D + one * (t - one)/D;
2060: }
2061:
2062:
2063: double CRecordPlayMidiDlg::splineSpline(
double t, int N, double x[], double y[], double z[])
2064: {
2065:   int i, j, k;
2066:   double d, h;
2067:
2068:   i = 0;  j = N - 1;
2069:   while (i < j) {
2070:     k = (i + j) / 2;
2071:     if (x[k] < t) i = k + 1;  else j = k;
2072:   }
2073:   if (i > 0) i--;
2074:   h = x[i + 1] - x[i];  d = t - x[i];
2075:   return (((z[i + 1] - z[i]) * d / h + z[i] * 3) * d
2076:     + ((y[i + 1] - y[i]) / h
2077:     - (z[i] * 2 + z[i + 1]) * h)) * d + y[i];
2078: }
2079:
2080: double CRecordPlayMidiDlg::splineMakeTDash2(
double D, double t, double zero, double onedash,
double one, double two)
2081: {
2082:   if(D == 0)
2083:     return 0;
2084:   return two + (t - onedash) * (one - two)/D;
2085: }
2086:
2087: // 小節を追加
2088: // g_nNotePerBar を使用
2089: void CRecordPlayMidiDlg::makeBarLine(
char *lp_tmp, int count)
2090: {
2091:   int i;
2092:   double barLength=0;
2093:   positionForCalc *lp_note;
2094:
2095:   // キャスト
2096:   lp_note = (positionForCalc *)lp_tmp;
2097:
2098:   // 新規小節
2099:   p_barNow = new BarLine;
2100:
2101:   p_barNow->number = count;// 小節番号
2102:   p_barNow->dDiffTime = new double [g_nNotePerBar];
2103:   p_barNow->dRelativeTime = new double [g_nNotePerBar];
2104:   p_barNow->dTime = new double [g_nNotePerBar];
2105: //  p_barNow->En = new double[g_nNotePerBar];
2106:   p_barNow->pNote = new positionForCalc *[g_nNotePerBar];
2107: //  p_barNow->Tn = new double[g_nNotePerBar];
2108:   p_barNow->pNextBar = NULL;
2109:
2110:   for(i=0;i<g_nNotePerBar;i++)
2111:   {
2112:     p_barNow->dDiffTime[i] = lp_note->dEndTime
 - lp_note->dStartTime;
2113:     p_barNow->dTime[i] = lp_note->dStartTime;
2114:     p_barNow->pNote[i] = lp_note;
2115:     barLength += p_barNow->dDiffTime[i];
2116:     if(lp_note != NULL)
2117:     {
```

```
2118:       lp_note=lp_note->pNext;
2119:     }
2120:     else if(i != g_nNotePerBar-1)
2121:     {
2122:       MessageBox("fatal error at makeBarLine()");
2123:     }
2124:   }
2125:
2126:   for(i=0;i<g_nNotePerBar;i++)
2127:   {
2128:     p_barNow->dRelativeTime[i] =
 p_barNow->dDiffTime[i] / barLength;
2129:   }
2130:
2131:   // ポインタのつなぎ合わせ
2132:   if(p_barStart == NULL && p_barEnd == NULL)
2133:   {
2134:     p_barStart = p_barNow;
2135:     p_barEnd = p_barNow;
2136:   }
2137:   else if(p_barStart != NULL && p_barEnd != NULL)
2138:   {
2139:     p_barEnd->pNextBar = p_barNow;
2140:     p_barEnd = p_barEnd->pNextBar;
2141:   }
2142:   else
2143:   {
2144:     MessageBox("不明なエラー");
2145:     exit(0);
2146:   }
2147: }
2148:
2149: // 小節構造のデータを作成
2150: void CRecordPlayMidiDlg::form2Bar()
2151: {
2152:   int i, count, flag;
2153:   double * tmpNote;
2154:   double dSumOfBar=0;
2155:   positionForCalc *lp_calcNow;
2156:   positionForCalc *lp_tmp;
2157:
2158:   count = 0;
2159:   flag = 1;
2160:
2161:   p_barStart = NULL;
2162:   p_barEnd = p_barStart;
2163:   p_barNow = p_barStart;
2164:
2165:   tmpNote = new double[g_nNotePerBar];
2166:
2167:   // positionForCalc 型データを見る
2168:   lp_calcNow=p_calcStart;
2169:   while(lp_calcNow != NULL)
2170:   {
2171:     lp_tmp=lp_calcNow;
2172:     dSumOfBar = 0;
2173:     // g_nNotePerBar 個数分
2174:     for(i=0;i<g_nNotePerBar;i++)
2175:     {
2176:       if(lp_tmp==NULL)
2177:       {
2178:         flag=0;
2179:         break;
2180:       }
2181:       tmpNote[i]=lp_tmp->dEndTime - lp_tmp->dStartTime;
2182:       dSumOfBar += tmpNote[i];
2183:       lp_tmp=lp_tmp->pNext;
2184:     }
2185:
2186:     if(flag)
2187:     {
2188:       for(i=0;i<g_nNotePerBar;i++)
2189:       {
2190:         tmpNote[i] = tmpNote[i] / dSumOfBar;
2191:       }
2192:
2193:       // 郡を作成。配列を追加
2194:       makeBarLine((char *)lp_calcNow, count);
2195:       count++;
2196:     }
2197:     else
2198:     {
2199:       break;
2200:     }
2201:     lp_calcNow=lp_calcNow->pNext;
2202:   }
2203: }
2204:
2205: // BarLine 型をすべて表示
2206: void CRecordPlayMidiDlg::printBarLine()
2207: {
2208:   BarLine * lp_tmp;
2209:   BarLine * lp_pre;// 一つ前の小節
2210:   FILE *fp;
2211:   FILE *fp2;
2212:   time_t tim;
2213:   char lfilename[100];
2214:   char lfilename2[100];
2215:   int i, count;
2216:   // ----------------------En 計算時間用
2217:   struct _timeb now;
2218:   time_t tmpStartTime;
2219:   int tmpStartms;
2220:   // ---------------------------------
2221:
2222:   time(&tim);
2223:   CTime  ctim(tim);
2224:
2225:   sprintf(lfilename,"%s/%s/%d_%d_%dBarLine",
 DATA_PATH, m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1]);
2226:   fp = fopen(lfilename, "w");
2227:   fprintf(fp,"UserName=%s, Date:%4d%2d%2d%2d%2d%2d\n"
2228:       , m_sUserName
2229:       , ctim.GetYear(), ctim.GetMonth(),
 ctim.GetDay()
2230:       , ctim.GetHour(), ctim.GetMinute(),
 ctim.GetSecond());
2231:
2232:   lp_tmp=p_barStart;
2233:   count = 0;
2234:
2235:   // ----------------------En 計算時間用
2236:   _ftime(&now);
2237:   tmpStartTime=now.time;
2238:   tmpStartms=now.millitm;
2239:   // ---------------------------------
2240:
2241:   // 初期設定
2242:   g_meanEn = new double[g_nNotePerBar];
2243:   g_meanRn = new double[g_nNotePerBar];
2244:   g_meanSn = new double[g_nNotePerBar];
2245:   g_meanTn = new double[g_nNotePerBar];
2246:   g_meanVn = new double[g_nNotePerBar];
2247:   g_sdEn = new double[g_nNotePerBar];
2248:   g_sdRn = new double[g_nNotePerBar];
2249:   g_sdSn = new double[g_nNotePerBar];
2250:   g_sdTn = new double[g_nNotePerBar];
2251:   g_sdVn = new double[g_nNotePerBar];
2252:   for(i=0;i<g_nNotePerBar;i++)
2253:   {
2254:     g_meanEn[i]=0;
2255:     g_meanRn[i]=0;
2256:     g_meanSn[i]=0;
2257:     g_meanTn[i]=0;
2258:     g_meanVn[i]=0;
2259:     g_sdEn[i]=0;
2260:     g_sdRn[i]=0;
2261:     g_sdSn[i]=0;
2262:     g_sdTn[i]=0;
2263:     g_sdVn[i]=0;
2264:   }
2265:
2266:   while(lp_tmp!=NULL)
2267:   {
2268:     count++;
2269:     fprintf(fp, "[%d 小節目] 時刻：%.3f〜%.3f\n"
2270:       , lp_tmp->number, lp_tmp->dTime[0],
 ((lp_tmp->dTime[g_nNotePerBar-1])
 + (lp_tmp->dDiffTime[g_nNotePerBar-1])));
2271:
2272:     lp_tmp->Rn = new double[g_nNotePerBar];
2273:     lp_tmp->Sn = new double[g_nNotePerBar];
2274:     if(count > 1)
2275:     {
2276:       lp_tmp->En = new double[g_nNotePerBar];
2277:       lp_tmp->Tn = new double[g_nNotePerBar];
2278:     }
2279:
2280:     for(i=0;i<g_nNotePerBar;i++)
2281:     {
2282:       fprintf(fp,"\t%d 番目の音符:時刻%.3f〜%.3f 音符長=%.3f 最
初の座標 [x y z]=[%d %d %d]\n"
2283:         , lp_tmp->pNote[i]->number,
 lp_tmp->pNote[i]->dStartTime, lp_tmp->pNote[i]->dEndTime,
 lp_tmp->pNote[i]->dEndTime - lp_tmp->pNote[i]->dStartTime
2284:         , lp_tmp->pNote[i]->posi_x[0],
 lp_tmp->pNote[i]->posi_y[0], lp_tmp->pNote[i]->posi_z[0]);
```

```
2285:
2286:       lp_tmp->Rn[i] = calcOneRyn(
(char *)lp_tmp->pNote[i]);
2287:       lp_tmp->Sn[i] = calcOneFurihaba(
(char *)lp_tmp->pNote[i]);
2288:       if(count > 1)
2289:       {
2290:         lp_tmp->En[i] = calcOneEn(
(char *)lp_pre->pNote[i], (char *)lp_tmp->pNote[i]);
2291:         lp_tmp->Tn[i] = calcOneTn(
(char *)lp_pre->pNote[i], (char *)lp_tmp->pNote[i]);
2292:         fprintf(fp, "En[%d][%d]=%.6f Rn[%d][%d]=%.6f
 Sn[%d][%d]=%.6f Tn[%d][%d]=%.6f Vn[%d][%d]=%d\n",
2293:           lp_tmp->number, i, lp_tmp->En[i],
2294:           lp_tmp->number, i, lp_tmp->Rn[i],
2295:           lp_tmp->number, i, lp_tmp->Sn[i],
2296:           lp_tmp->number, i, lp_tmp->Tn[i],
2297:           lp_tmp->number, i,
 lp_tmp->pNote[i]->nVelocity);
2298:         g_meanEn[i] += lp_tmp->En[i];
2299:         g_meanTn[i] += lp_tmp->Tn[i];
2300:         g_sdEn[i] += pow(lp_tmp->En[i], 2);
2301:         g_sdTn[i] += pow(lp_tmp->Tn[i], 2);
2302:       }
2303:       g_meanRn[i] += lp_tmp->Rn[i];
2304:       g_meanSn[i] += lp_tmp->Sn[i];
2305:       g_sdRn[i] += pow(lp_tmp->Rn[i], 2);
2306:       g_sdSn[i] += pow(lp_tmp->Sn[i], 2);
2307:       g_meanVn[i] += (double)lp_tmp->pNote[i]->nVelocity;
2308:     }
2309:     lp_pre=lp_tmp;
2310:     lp_tmp=lp_tmp->pNextBar;
2311:   }
2312:
2313:   sprintf(lfilename2,"%s/%s/%d_%d_%dcluster",
 DATA_PATH, m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1]);
2314:   fp2 = fopen(lfilename2, "a");
2315:
2316:   for(i=0;i<g_nNotePerBar;i++)
2317:   {
2318:     g_meanEn[i] /= (count-1);  // 要素が一つ少ない
2319:     g_meanRn[i] /= count;
2320:     g_meanSn[i] /= count;
2321:     g_meanTn[i] /= (count-1);  // 要素が一つ少ない
2322:     g_meanVn[i] /= count;
2323:     g_sdEn[i] /= (count-1);  // 要素が一つ少ない
2324:     g_sdRn[i] /= count;
2325:     g_sdSn[i] /= count;
2326:     g_sdTn[i] /= (count-1);  // 要素が一つ少ない
2327: //    g_sdVn[i] /= count;
2328:     g_sdEn[i] = sqrt(g_sdEn[i] - pow(g_meanEn[i], 2));
2329:     g_sdRn[i] = sqrt(g_sdRn[i] - pow(g_meanRn[i], 2));
2330:     g_sdSn[i] = sqrt(g_sdSn[i] - pow(g_meanSn[i], 2));
2331:     g_sdTn[i] = sqrt(g_sdTn[i] - pow(g_meanTn[i], 2));
2332: //    g_sdVn[i] = sqrt(g_sdVn[i] - pow(g_meanVn[i], 2));
2333:     fprintf(fp, "meanEn[%d]=%.6f meanRn[%d]=%.6f
 meanSn[%d]=%.6f meanTn[%d]=%.6f meanVn[%d]=%.6f\n"
2334:         , i, g_meanEn[i], i, g_meanRn[i], i,
 g_meanSn[i], i, g_meanTn[i], i, g_meanVn[i]);
2335:     fprintf(fp2, "# meanEn(%d)=%.6f\n", i, g_meanEn[i]);
2336:   }
2337:   for(i=0;i<g_nNotePerBar;i++)
2338:   {
2339:     fprintf(fp2, "# meanRn(%d)=%.6f\n", i, g_meanRn[i]);
2340:   }
2341:   for(i=0;i<g_nNotePerBar;i++)
2342:   {
2343:     fprintf(fp2, "# meanSn(%d)=%.6f\n", i, g_meanSn[i]);
2344:   }
2345:   for(i=0;i<g_nNotePerBar;i++)
2346:   {
2347:     fprintf(fp2, "# meanTn(%d)=%.6f\n", i, g_meanTn[i]);
2348:   }
2349:   for(i=0;i<g_nNotePerBar;i++)
2350:   {
2351:     fprintf(fp2, "# meanVn(%d)=%.6f\n", i, g_meanVn[i]);
2352:   }
2353:
2354:   // sd を求める
2355:   lp_tmp=p_barStart;
2356:   while(lp_tmp != NULL)
2357:   {
2358:     for(i=0;i<g_nNotePerBar;i++)
2359:     {
2360:       g_sdVn[i] += pow((double)lp_tmp->pNote[i]->nVelocity
 - g_meanVn[i],2);
2361:     }
2362:     lp_pre=lp_tmp;
2363:     lp_tmp=lp_tmp->pNextBar;
2364:   }
2365:   for(i=0;i<g_nNotePerBar;i++)
2366:   {
2367:     g_sdVn[i] /= (count-1);    // 偏差は要素数ー１で割る
2368:     g_sdVn[i] = sqrt(g_sdVn[i]);
2369:   }
2370:
2371:   fclose(fp2);
2372:   fclose(fp);
2373: }
2374:
2375: // 2 つのストローク波形が離散の場合
2376: double CRecordPlayMidiDlg::splineMakeTDashDiscrete(
double t, double zero, double onedash, double one, double two)
2377: {
2378:   return (two + (t - onedash) * (one - two)
 / (zero - onedash));
2379: }
2380:
2381: // 二つの音符間の時間的差（Tn）を計算する
2382: double CRecordPlayMidiDlg::calcOneTn(
char *before, char *after)
2383: {
2384:   double tmpDeltaTime=0;
2385:   double tmpDeltaTimeminus=0;
2386:   double tmpTn=0;
2387:   positionForCalc * lp_before = (positionForCalc *)before;
2388:   positionForCalc * lp_after = (positionForCalc *)after;
2389:
2390:   tmpDeltaTimeminus = lp_before->dEndTime
 - lp_before->dStartTime;
2391:   tmpDeltaTime = lp_after->dEndTime
 - lp_after->dStartTime;
2392:
2393:   tmpTn = (tmpDeltaTime - tmpDeltaTimeminus)*2
 / (tmpDeltaTime + tmpDeltaTimeminus);
2394:   if(tmpTn < 0)
2395:     tmpTn = tmpTn * (-1);
2396:
2397:   return tmpTn;
2398: }
2399:
2400: // クラスター分析を行い、その後に小節を再構成
2401: int CRecordPlayMidiDlg::clusterBarLine()
2402: {
2403:   FILE *fs;
2404:   char lfilename[256];
2405:
2406:   int n=1;         // 要素数
2407:   int nc=g_nNotePerBar;  // 最終クラスタ数
2408:   int cnc;         // 現在のクラスタ数
2409:
2410:   BarLine *lp_now=p_barStart;
2411:
2412:   double **e;         // クラスタの要素（n 番目）
2413:   double **newe;       // クラスタの要素（n+1 番目）
2414:   double *ratio;
2415:
2416:   // use at logfitting
2417:   double *diff;       // 生打拍時間差
2418:   double **logfitted;    // 対数によるフィッティング
2419:   double shiftValueAtMinError;
2420:
2421:   int *cluster;       // i 番目の要素が属するクラスターのラベル
2422:   int *existcluster;    // クラスターフラグ
2423:   int *clus2obs;       // 最終クラスター？
2424:   int *mincluster;
2425:   int *label;
2426:
2427:   int isFittingDoing;    // logfitting が収束したか否か
2428:
2429:   // 要素数（小節数）を求める
2430:   while(lp_now->pNextBar != NULL)
2431:   {
2432:     lp_now=lp_now->pNextBar;
2433:     n++;
2434:   }
2435:
2436:   // 目標とするクラスター数よりも事例数が少ない場合はエラー
2437:   if( n < nc )
2438:   {
2439:     m_strRes += "cluster: The number of data was less
 than the number of cluster.\r\ncluster:aborted...\r\n";
2440:     UpdateData(FALSE);
2441:     MessageBox("cluster: クラスター化に失敗したので、診断結果
は出ません。\r\n もう一度打拍をやり直してください。");
2442:     return -1;
2443:   }
```

```
2444:
2445:   // logfile
2446:   sprintf(lfilename,"%s/%s/%d_%d_%dcluster",
 DATA_PATH, m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1]);
2447:   fs = fopen(lfilename,"w+");
2448:
2449:   // 変数の領域確保
2450:   e = new double *[n];
2451:   newe = new double *[n];
2452:   logfitted = new double *[n];
2453:   {// 2 次元配列を動的に割り当てるには 1 段階ずつ作成する
2454:     int i;
2455:     for(i=0;i<n;i++)
2456:     {
2457:       e[i] = new double[nc];
2458:       newe[i] = new double[nc];
2459:       logfitted[i] = new double[nc];
2460:     }
2461:   }
2462:   cluster = new int[n];
2463:   existcluster = new int[n];
2464:   label = new int[n];
2465:   diff = new double[n + nc - 1];
2466:
2467:   clus2obs = new int[nc];
2468:   mincluster = new int[nc];
2469:
2470:   // 初期のクラスタ番号
2471:   {
2472:     int i,k;
2473:     for(i=0;i<n;i++)
2474:     {
2475:       cluster[i] = i;     // クラスタ番号（i 番目のクラスター
はすべて i が中心）
2476:       existcluster[i] = 1;  // クラスタ存在フラグ
2477:       for(k=0;k<nc;k++)
2478:       {
2479:         logfitted[i][k] = 0;
2480:       }
2481:     }
2482:   }
2483:
2484:   // [nc] の初期設定
2485:   g_meanSHIFT = new double[nc];
2486:   g_sdSHIFT = new double[nc];
2487:   ratio = new double [nc];
2488:   {
2489:     int i;
2490:     for(i=0;i<nc;i++)
2491:     {
2492:       ratio[i] = g_dTemplateNotes[i];
2493:       g_meanSHIFT[i] = 0;
2494:       g_sdSHIFT[i] = 0;
2495:     }
2496:   }
2497:   // e,newe の初期設定
2498:   {
2499:     int i,k;
2500:     double l;
2501:     lp_now=p_barStart;
2502:     for(i=0;i<n;i++)
2503:     {
2504:       l=0;
2505:       for(k=0;k<nc;k++)
2506:       {
2507:         e[i][k]=lp_now->pNote[k]->dEndTime
 - lp_now->pNote[k]->dStartTime;
2508:         newe[i][k]=lp_now->pNote[k]->dEndTime
 - lp_now->pNote[k]->dStartTime;
2509:         diff[i+k]=e[i][k];
2510:         l+=e[i][k];
2511:       }
2512:       lp_now=lp_now->pNextBar;
2513:       for(k=0;k<nc;k++)
2514:       {
2515:         e[i][k] /= (1.0 * l);
2516:         newe[i][k] /= (1.0 * l);
2517:       }
2518:     }
2519:   }
2520:
2521:   // クラスタリング
2522:   cnc = n;  // 現在のクラスタ数
2523:   // cnc が目標となるクラスタ数 (nc) になるまでループ
2524:   while( cnc > nc  )
2525:   {
2526:     // -経過表示--------------------------------
2527:     if( 0 )
2528:     {
2529:       fprintf(fs,"cnc=[%d] ",cnc);
2530:       fprintf(fs,"n=[%d]\n",n);
2531:     }
2532:     // ----------------------------------------
2533:
2534:     {
2535:       int i,j,k,num;
2536:       int nearest[2]; // 最も近い 2{点|クラスタ}のデータ要素
番号
2537:       double d,dmin;
2538:
2539:       for(i=0;i<n;i++)
2540:       {
2541: //        fprintf(fs,"検索中...i=%d\n",i);
2542:         // i 番目のクラスターが存在したら...
2543:         if( existcluster[i] == 1 )
2544:         {
2545:           for(j=i+1;j<n;j++)
2546:           {
2547:             if( existcluster[i] == 1 )
2548:             {
2549:               d = 0;
2550:               for(k=0;k<nc;k++)
2551:               {
2552:                 // 一次絶対値距離の和
2553:                 d += fabs(newe[i][k] - newe[j][k]);
2554:               }
2555:               if(i==0 && j==1)
2556:               {
2557:                 // これが最小の d
2558:                 dmin = d;
2559:               }
2560:               else if( d < dmin )
2561:               {
2562:                 dmin = d;
2563:                 nearest[0] = i;
2564:                 nearest[1] = j;
2565:               }
2566:             }
2567:           }
2568:         }
2569:       }
2570:       existcluster[nearest[1]] = 0; // クラスタ消滅 (i に吸
収された)
2571:       {
2572:         int m;
2573:         for(m=0;m<n;m++)
2574:         {
2575:           if( m == nearest[1] )
2576:           {
2577:             cluster[m] = nearest[0];  // クラスタ番号更新
2578:           }
2579:           else if( cluster[m] == nearest[1] )
2580:           {
2581:             cluster[m] = nearest[0];  // クラスタ番号更新
2582:           }
2583:         }
2584:       }
2585:
2586:       // -経過表示--------------------------------
2587:       if( 0 )
2588:       {
2589:         int i,k;
2590:         for(i=0;i<n;i++)// n:小節系列個数
2591:         {
2592:           fprintf(fs,"existcluster[%d]=[%d] ",i,
existcluster[i]);
2593:           fprintf(fs,"cluster[%d]=[%d] ",i,cluster[i]);
2594:           for(k=0;k<nc;k++)
2595:           {
2596:             fprintf(fs,"newe[%d][%d]=[%f] ",i,k,newe[i][k]);
2597:           }
2598:           fprintf(fs,"\n");
2599:         }
2600:       }
2601:       // ----------------------------------------
2602:
2603:       cnc--;
2604:
2605:       for(i=0;i<n;i++)
2606:       {
2607:         if(existcluster[i] == 1)
2608:         {
2609:           for(k=0;k<nc;k++) newe[i][k] = 0;
2610:           num = 0;
2611:           for(j=i;j<n;j++)
2612:           {
2613:             if(cluster[j] == i)
```

```
            {
              for(k=0;k<nc;k++)
              {
                newe[i][k] += e[j][k];
              }
              num++;
            }
          }
          // クラスタの重心を更新？
          for(k=0;k<nc;k++)
          {
            newe[i][k] = newe[i][k] / (1.0 * num);
          }
        }
        else
        {// existcluster != 1
          for(k=0;k<nc;k++)
          {
            newe[i][k] = -1;
          }
        }
      }
    }
  }  // end of while


  // clus2obs[] の作成
  {
    int i,j,k;
    j = 0;
    for(i=0;i<n;i++)
    {
      if( existcluster[i] == 1 )
      {
        clus2obs[j++] = i;// クラスターの中心の集合を作っている？
      }
    }
  }

  // clus2obs の表示
  {
    int k;
    for(k=0;k<nc;k++)
    {
      fprintf(fs,"clus2obs[%d]=[%d]\n",k,clus2obs[k]);
    }
  }

  // テンプレート・リズムパターンを導入
  {
    int i,j,k,kk;
    int uniqflag;
    double **error;
    double minerror;
    // error 初期設定
    error = new double *[nc];
    for(i=0;i<nc;i++)
    {
      error[i] = new double[nc];
      for(j=0;j<nc;j++)
      {
        error[i][j] = 0;
      }
    }
    for(i=0;i<nc;i++)
    {
      for(k=0;k<nc;k++)
      {
        for(kk=0;kk<nc;kk++)
        {
          // クラスターの中心と教師リズム値の差（1 乗距離）を計算
          error[i][kk] += fabs(newe[clus2obs[i]][k] - ratio[(k + kk) % nc]);
        }
      }
    }
    for(i=0;i<nc;i++)
    {
      for(j=0;j<nc;j++)
      {
        fprintf(fs,"error[%d][%d]=[%f] ",i,j,error[i][j]);
      }
      fprintf(fs,"\n");
    }


    // 教師リズム値ともっとも誤差が少ないクラスターを探している
    uniqflag = 1;
    for(i=0;i<nc;i++)
    {
      mincluster[i] = -1;
      for(j=0;j<nc;j++)
      {
        if( j==0 )
        {
          minerror = error[j][i];
          mincluster[i] = j;
        }
        else if( error[j][i] < minerror )
        {
          minerror = error[j][i];
          mincluster[i] = j;
        }
      }
      fprintf(fs,"mincluster[%d]=[%d]\n",i,mincluster[i]);
    }

    // unique であるかどうかのチェック
    for(i=0;i<nc;i++)
    {
      for(j=i+1;j<nc;j++)
      {
        if( mincluster[i] == mincluster[j] )
        {
          uniqflag = 0;
        }
      }
    }

    fprintf(fs,"uniqflag=[%d]\n",uniqflag);
    if(uniqflag != 1)
    {
      m_strRes += "cluster: The given time train was not distinguished by this algorithm.\r\n";
      m_strRes += "cluster: aborted...\r\n";

      UpdateData(FALSE);
      fclose(fs);
      MessageBox("cluster: クラスター化に失敗したので、診断結果は出ません。\r\n もう一度打拍をやり直してください。");

      return -1;
    }
  }

  // そぐわない小節を削除
  {
    int i,j;
    BarLine *lp_tmp=p_barStart;
    BarLine *lp_pre=NULL;
    for(i=0;i<nc;i++)
    {
      // テンプレートリズムパターンに合う小節を採用
      if(mincluster[0] == i)
      {
        // i:クラスタのラベル n:小節数
        for(j=0;j<n;j++)
        {
          if(lp_tmp==NULL)
          {
            break;
          }
          if(cluster[j] == i)
          {
            // lp_tmp が指している小節は残る
            lp_pre=lp_tmp;
            lp_tmp=lp_tmp->pNextBar;
          }
          else
          {
            // リストの削除
            if(lp_pre != NULL)
            {
              // lp_tmp が指している小節をデリート
              lp_pre->pNextBar=lp_tmp->pNextBar;
              // 各動的要素を削除
              delete(lp_tmp->dTime);
              delete(lp_tmp->dDiffTime);
              delete(lp_tmp->dRelativeTime);

              delete(lp_tmp);
              lp_tmp = lp_pre->pNextBar;
            }
            else
            {
              // リストの先頭を削除する場合
```

```
2787:               lp_pre = lp_tmp;
2788:               lp_tmp=lp_tmp->pNextBar;
2789:               // 各動的要素を削除
2790:               delete(lp_pre->dTime);
2791:               delete(lp_pre->dDiffTime);
2792:               delete(lp_pre->dRelativeTime);
2793:
2794:               delete(lp_pre);
2795:               lp_pre = NULL;
2796:               // これを忘れると先頭がなくなる
2797:               p_barStart = lp_tmp;
2798:             }
2799:           }
2800:
2801:         }
2802:       }
2803:     }
2804:   }
2805:
2806:   // logfitting を行う
2807:   isFittingDoing = 1;
2808:   {
2809:     int i,k,s;
2810:     double logerror = 0;
2811:     double prevlogerror = 0;
2812:     int nscan = 0;
2813:     int maxscan = 800;
2814:     int half = (int)(maxscan / 2.0);
2815:     int minShift = 0;
2816:     double times,prevtimes;
2817:     double seqtempl[16];
2818:     double len;
2819:
2820:   //  for(i=0;i<(n - nc + 1);i++)
2821:     for(i=0;i<n;i++)
2822:     {
2823:       if( cluster[i] == mincluster[0] )
2824:       {
2825:         // log-fitting 計算
2826:         for(s=0;s<=maxscan;s++)
2827:         {
2828:           prevtimes = times;
2829:           // ずらし量設定
2830:           times = 1.0 + ((s - half + 1) * 0.001);
2831:           if( 0 )
2832:           {
2833:             fprintf(stderr,"s=[%d]  times=[%f]",s,times);
2834:             fprintf(stderr," prevlogerror=[%f]
 logerror=[%f]",prevlogerror,logerror);
2835:             fprintf(stderr,"\r");
2836:           }
2837:           for(k=0;k<nc;k++)
2838:           {
2839:             seqtempl[k] = ratio[k] * times;
2840:           }
2841:           prevlogerror = logerror;
2842:           logerror = 0;
2843:           for(k=0;k<nc;k++)
2844:           {
2845:             logerror += fabs( log( e[i][k] )
 - log( seqtempl[k] ) );
2846:           }
2847:           if( s > 1 )
2848:           {
2849:             if( logerror > prevlogerror )
2850:             {
2851:               shiftValueAtMinError = prevtimes;
2852:               break;
2853:             }
2854:           }
2855:         }
2856:       }
2857:     }
2858:
2859:     // 収束判定
2860:     if( s >= maxscan )
2861:     {
2862:       isFittingDoing = -1;  // 収束しなかった
2863:     }
2864:     else
2865:     {
2866:       fprintf(fs,"\nshiftValueAtMinError=[%f] 収束しました.
\n",shiftValueAtMinError);
2867:     }
2868:
2869:     // 収束しなかったらここには来ない
2870:     // log-fitting 後の真数の算出
2871:     if(isFittingDoing == 1)
2872:     {
2873:       for(i=0;i<n;i++)
2874:       {
2875:         if( cluster[i] == mincluster[0] )
2876:         {
2877:           len = 0;
2878:           for(k=0;k<nc;k++)
2879:           {
2880:             logfitted[i][k] = (e[i][k]
 / shiftValueAtMinError);
2881:             len += logfitted[i][k];
2882:           }
2883:
2884:           //正規化
2885:           for(k=0;k<nc;k++)
2886:           {
2887:             logfitted[i][k] /= (1.0 * len);
2888:           }
2889:         }
2890:       }
2891:       isFittingDoing = 0;  // 正常
2892:     }
2893:   }
2894:
2895:   fprintf(fs,"n=[%d]\n",n);
2896:   fprintf(fs,"nc=[%d]\n",nc);
2897:   {
2898:     int i,k;
2899:     int meas;
2900:     int validnotes;
2901:     int errornotes;
2902:     int lasttick;
2903:     int desire;
2904:     int recog;
2905:     double recograte;
2906:     for(i=0;i<n;i++) label[i] = 0;
2907:
2908:     // logfitting 結果
2909:     if( isFittingDoing != 0 )
2910:     {
2911:       m_strRes += "cluster: logfitting の際, 収束しませんでし
た. \r\n";
2912:       m_strRes += "cluster: aborted...\r\n";
2913:       UpdateData(FALSE);
2914:       fclose(fs);
2915:       MessageBox("cluster: クラスター化に失敗したので、診断結
果は出ません。\r\n もう一度打拍をやり直してください。");
2916:       return -3;
2917:     }
2918:
2919:     meas = 0;
2920:     for(i=0;i<n;i++)
2921:     {
2922:       // 小節として正確に認識している部分
2923:       if( cluster[i] == mincluster[0] )
2924:       {
2925:       meas++;
2926:       // 1 拍目〜nc 拍目
2927:       for(k=0;k<nc;k++)
2928:       {
2929:         label[i+k] = k+1;// 12341234...
2930:         {
2931:           // 演奏音符と理想音符の差を計算
2932:           double shiftpcnt = 100.0 * (logfitted[i][k]
 - ratio[k]) / (1.0 * ratio[k]);
2933:           g_meanSHIFT[k] += shiftpcnt;
2934:           g_sdSHIFT[k] += pow(shiftpcnt, 2);
2935:         }
2936:       }
2937:     }
2938:   }
2939:
2940:
2941:   // 小節あたりの平均、偏差
2942:   for(k=0;k<nc;k++)
2943:   {
2944:     g_meanSHIFT[k] /= (1.0 * meas);
2945:     g_sdSHIFT[k] /= (1.0 * meas);
2946:     g_sdSHIFT[k] = sqrt(g_sdSHIFT[k]
 - pow(g_meanSHIFT[k], 2));
2947:   }
2948:
2949:   recog = 0;
2950:   validnotes = 0;
2951:   errornotes = 0;
2952:
2953:   // 要約統計量の計算
2954:   for(i=0;i<n;i++)
2955:   {
2956:     desire = (lasttick + 1)%nc;
```

```
2957:     if( desire == 0 ) desire = nc;
2958:     if( i == 0 )
2959:     {
2960:       validnotes++;
2961:     }
2962:     else if( label[i] == desire )
2963:     {
2964:       validnotes++;
2965:     }
2966:     else
2967:     {
2968:       errornotes++;
2969:     }
2970:     lasttick = label[i];
2971:     if( label[i] > 0 ) recog++;
2972:
2973:     // ラベル付けされた入力データの表示
2974:     fprintf(fs,"%f %d\n",diff[i],label[i]);
2975:   }
2976:   recograte = recog / (1.0 * n);
2977:
2978:   // フィードバック
2979:   fprintf(fs,"# clusters=%d\n",nc);
2980:   fprintf(fs,"# input notes=%d\n",n);
2981:   fprintf(fs,"# shiftValueAtMinError=%f\n",
shiftValueAtMinError);
2982:   fprintf(fs,"# recognized notes=%d\n",recog);
2983:   fprintf(fs,"# valid notes=%d\n",validnotes);
2984:   fprintf(fs,"# rate of recognized notes=%f\n",recograte);
2985:   fprintf(fs,"# error notes=%d\n",errornotes);
2986:   fprintf(fs,"# meas=%d\n",meas);
2987:   fprintf(fs,"#\n");
2988:   for(k=0;k<nc;k++) fprintf(fs,"# g_meanSHIFT(%d)=%f\n",
k+1,g_meanSHIFT[k]);
2989:   for(k=0;k<nc;k++) fprintf(fs,"# g_sdSHIFT(%d)=%f\n",
k+1,g_sdSHIFT[k]);
2990:   }
2991:
2992:   fclose(fs);
2993:   return 0;
2994: }
2995:
2996:
2997: // リズムパターンを作成し、画像を張る
2998: void CRecordPlayMidiDlg::OnRhythm1()
2999: {
3000:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
3001:   UpdateData(TRUE);
3002:   g_bIsAfter=0;
3003:   m_nSelectedPattern = 1;
3004:   UpdateData(false);
3005:   printBarPict();
3006:   g_nNotePerBar = 2;
3007:   delete(g_dTemplateNotes);
3008:   g_dTemplateNotes = new double[g_nNotePerBar];
3009:   g_dTemplateNotes[0] = 0.75;
3010:   g_dTemplateNotes[1] = 0.25;
3011:
3012:   // アクセント提示
3013:   setAccent();
3014:   drawCorrectLength();
3015:   playRhythm(g_nOtehonBar);
3016: }
3017:
3018: // お手本リズムパターン画像貼り付け関数（引数：パターンナンバー）
3019: int CRecordPlayMidiDlg::printBarPict()
3020: {
3021:   CDC* pDC = m_barCtrl.GetDC();
3022:   CDC myDC;
3023:   CBitmap bmpBar;
3024:
3025:   bmpBar.LoadBitmap(m_nSelectedPattern + g_nBius);
3026:   m_nPerformTime = g_nPerformTime[m_nSelectedPattern-1];
3027:   UpdateData(false);
3028:   myDC.CreateCompatibleDC(pDC);
3029:   CBitmap* oldBMP=myDC.SelectObject(&bmpBar);
3030:   pDC->BitBlt(0,0,562,116,&myDC,0,0,SRCCOPY);
3031:
3032:   return 0;
3033: }
3034:
3035: void CRecordPlayMidiDlg::OnRhythm2()
3036: {
3037:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
3038:   UpdateData(TRUE);
3039:   g_bIsAfter=0;
3040:   m_nSelectedPattern = 2;
3041:   UpdateData(false);
3042:   printBarPict();
3043:   g_nNotePerBar = 3;
3044:   delete(g_dTemplateNotes);
3045:   g_dTemplateNotes = new double[g_nNotePerBar];
3046:   g_dTemplateNotes[0] = 0.75;
3047:   g_dTemplateNotes[1] = 0.125;
3048:   g_dTemplateNotes[2] = 0.125;
3049:
3050:   // アクセント提示
3051:   setAccent();
3052:   drawCorrectLength();
3053:   playRhythm(g_nOtehonBar);
3054: }
3055:
3056: void CRecordPlayMidiDlg::OnRhythm3()
3057: {
3058:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
3059:   UpdateData(TRUE);
3060:   g_bIsAfter=0;
3061:   m_nSelectedPattern = 3;
3062:   UpdateData(false);
3063:   printBarPict();
3064:   g_nNotePerBar = 3;
3065:   delete(g_dTemplateNotes);
3066:   g_dTemplateNotes = new double[g_nNotePerBar];
3067:   g_dTemplateNotes[0] = 0.5;
3068:   g_dTemplateNotes[1] = 0.375;
3069:   g_dTemplateNotes[2] = 0.125;
3070:
3071:   // アクセント提示
3072:   setAccent();
3073:   drawCorrectLength();
3074:   playRhythm(g_nOtehonBar);
3075: }
3076:
3077: void CRecordPlayMidiDlg::OnRhythm4()
3078: {
3079:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
3080:   UpdateData(TRUE);
3081:   g_bIsAfter=0;
3082:   m_nSelectedPattern = 4;
3083:   UpdateData(false);
3084:   printBarPict();
3085:   g_nNotePerBar = 3;
3086:   delete(g_dTemplateNotes);
3087:   g_dTemplateNotes = new double[g_nNotePerBar];
3088:   g_dTemplateNotes[0] = 0.5;
3089:   g_dTemplateNotes[1] = 0.25;
3090:   g_dTemplateNotes[2] = 0.25;
3091:
3092:   // アクセント提示
3093:   setAccent();
3094:   drawCorrectLength();
3095:   playRhythm(g_nOtehonBar);
3096: }
3097:
3098: void CRecordPlayMidiDlg::OnRhythm5()
3099: {
3100:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
3101:   UpdateData(TRUE);
3102:   g_bIsAfter=0;
3103:   m_nSelectedPattern = 5;
3104:   UpdateData(false);
3105:   printBarPict();
3106:   g_nNotePerBar = 4;
3107:   delete(g_dTemplateNotes);
3108:   g_dTemplateNotes = new double[g_nNotePerBar];
3109:   g_dTemplateNotes[0] = 0.5;
3110:   g_dTemplateNotes[1] = 0.25;
3111:   g_dTemplateNotes[2] = 0.125;
3112:   g_dTemplateNotes[3] = 0.125;
3113:
3114:   // アクセント提示
3115:   setAccent();
3116:   drawCorrectLength();
3117:   playRhythm(g_nOtehonBar);
3118: }
3119:
3120: void CRecordPlayMidiDlg::OnRhythm6()
3121: {
3122:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
3123:   UpdateData(TRUE);
3124:   g_bIsAfter=0;
3125:   m_nSelectedPattern = 6;
```

```
3126:   UpdateData(false);
3127:   printBarPict();
3128:   g_nNotePerBar = 4;
3129:   delete(g_dTemplateNotes);
3130:   g_dTemplateNotes = new double[g_nNotePerBar];
3131:   g_dTemplateNotes[0] = 0.5;
3132:   g_dTemplateNotes[1] = 0.125;
3133:   g_dTemplateNotes[2] = 0.125;
3134:   g_dTemplateNotes[3] = 0.25;
3135:
3136:   // アクセント提示
3137:   setAccent();
3138:   drawCorrectLength();
3139:   playRhythm(g_nOtehonBar);
3140: }
3141:
3142: void CRecordPlayMidiDlg::OnRhythm7()
3143: {
3144:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加して
ください
3145:   UpdateData(TRUE);
3146:   g_bIsAfter=0;
3147:   m_nSelectedPattern = 7;
3148:   UpdateData(false);
3149:   printBarPict();
3150:   g_nNotePerBar = 4;
3151:   delete(g_dTemplateNotes);
3152:   g_dTemplateNotes = new double[g_nNotePerBar];
3153:   g_dTemplateNotes[0] = 0.5;
3154:   g_dTemplateNotes[1] = 0.125;
3155:   g_dTemplateNotes[2] = 0.25;
3156:   g_dTemplateNotes[3] = 0.125;
3157:
3158:   // アクセント提示
3159:   setAccent();
3160:   drawCorrectLength();
3161:   playRhythm(g_nOtehonBar);
3162: }
3163:
3164: void CRecordPlayMidiDlg::OnRhythm8()
3165: {
3166:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加して
ください
3167:   UpdateData(TRUE);
3168:   g_bIsAfter=0;
3169:   m_nSelectedPattern = 8;
3170:   UpdateData(false);
3171:   printBarPict();
3172:   g_nNotePerBar = 4;
3173:   delete(g_dTemplateNotes);
3174:   g_dTemplateNotes = new double[g_nNotePerBar];
3175:   g_dTemplateNotes[0] = 0.625;
3176:   g_dTemplateNotes[1] = 0.125;
3177:   g_dTemplateNotes[2] = 0.125;
3178:   g_dTemplateNotes[3] = 0.125;
3179:
3180:   // アクセント提示
3181:   setAccent();
3182:   drawCorrectLength();
3183:   playRhythm(g_nOtehonBar);
3184: }
3185:
3186:
3187: // 正しい間隔を描画
3188: int CRecordPlayMidiDlg::drawCorrectLength()
3189: {
3190:   UpdateData(TRUE);
3191:
3192:   int lnWidth;    // 横幅
3193:   int lnHeight;    // 縦幅
3194:   int i;
3195:   double left=0;
3196:   int fromx, fromy;    // 始点
3197:   int tox, toy;      // 終点
3198:   // グラフィック関係
3199:   CWnd* barCorrect = GetDlgItem(IDC_CORRECT);
3200:   CClientDC correctDC(barCorrect);
3201:   CPen blackLine(PS_SOLID,2,RGB(0,0,0));
3202:   CPen* oldPen = correctDC.SelectObject(&blackLine);
3203:
3204:   // ------------------ 全体を塗りつぶし --------------------
3205:   CRect allBar;
3206:   // 四角形の全体を取得
3207:   barCorrect->GetClientRect(allBar);
3208:   // 塗りつぶしブラシ
3209:   CBrush correctBrush;
3210:   CBrush* naka;
3211:   correctBrush.CreateSolidBrush(RGB(0,128,255));
3212:   // correctBrush を有効にする
3213:   naka = correctDC.SelectObject(&correctBrush);
3214:   correctDC.Rectangle(allBar);
3215:
3216:   // 縦幅、横幅の取得
3217:   lnWidth = (int)allBar.Width();
3218:   lnHeight = (int)allBar.Height();
3219:
3220:   // 縦棒を引く
3221:   for(i=0;i<g_nNotePerBar-1;i++)
3222:   {
3223:     fromx = (int)((left + g_dTemplateNotes[i]) * lnWidth);
3224:     tox = fromx;
3225:     fromy = 0;
3226:     toy = lnHeight-1;
3227:     correctDC.MoveTo(fromx, fromy);
3228:     correctDC.LineTo(tox, toy);
3229:     left += g_dTemplateNotes[i];
3230:   }
3231:
3232:   return 0;
3233: }
3234:
3235: // 時間差をログへ表示
3236: void CRecordPlayMidiDlg::printDeltaTime()
3237: {
3238:   FILE* fp;
3239:   int i;
3240:   char lfilename[100];
3241:
3242:   sprintf(lfilename,"%s/%s/%d_%d_%ddeltatime",
 DATA_PATH, m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1]);
3243:   fp = fopen(lfilename, "w");
3244:
3245:   for(i=1;i<m_nTestSec * 60;i++)
3246:   {
3247:     fprintf(fp,"deltatime = %f\n",m_simple.dDiffTime[i]
 - m_simple.dDiffTime[i-1]);
3248:   }
3249:
3250:   fclose(fp);
3251: }
3252:
3253:
3254: double CRecordPlayMidiDlg::calcOneEn(
char *before, char *after)
3255: {
3256:   positionForCalc * lp_before = (positionForCalc *)before;
3257:   positionForCalc * lp_after = (positionForCalc *)after;
3258:
3259:   double * lp_x;
3260:   double * lp_y;
3261:   double * lp_z;// 補間した導関数用
3262:   double * lp_time;
3263:   double * lp_posi_x;
3264:   double * lp_posi_y;
3265:   double * lp_posi_z;
3266:   // 座標用
3267:   double dTDash;
3268:   double dXnMinus, dYnMinus, dZnMinus;
3269:   double dBarXn, dBarYn, dBarZn;
3270:   double tmpSumOfNumerator, tmpSumOfDenominator;
3271:   double deltaNumerator, deltaDenominator;
3272:
3273:   int i,j;
3274:
3275:   // Bar の計算
3276:   dBarXn=0;dBarYn=0;dBarZn=0;
3277:   for(i=0;i<lp_after->nData;i++)
3278:   {
3279:     dBarXn += (double)lp_after->posi_x[i];
3280:     dBarYn += (double)lp_after->posi_y[i];
3281:     dBarZn += (double)lp_after->posi_z[i];
3282:   }
3283:   dBarXn = dBarXn / lp_after->nData;
3284:   dBarYn = dBarYn / lp_after->nData;
3285:   dBarZn = dBarZn / lp_after->nData;
3286:
3287:   // double 型配列の作成
3288:   lp_time = new double[lp_before->nData+2*g_nExtendWidth];
3289:   lp_posi_x = new double[lp_before->nData+2
 * g_nExtendWidth];
3290:   lp_posi_y = new double[lp_before->nData+2
 * g_nExtendWidth];
3291:   lp_posi_z = new double[lp_before->nData+2
 * g_nExtendWidth];
3292:   for(i=0;i<lp_before->nData+2*g_nExtendWidth;i++)
3293:   {
3294:     // lp_before の範囲± g_nExtendWidth の範囲のラベルを取得
```

```
3295:     j=i+lp_before->nLabel[0]-g_nExtendWidth;
3296:
3297:     // ラベルが取得位置データの範囲を超えないかチェック
3298:     if(j < 0)
3299:     {
3300:       MessageBox("LabelError in calcEn(low)");
3301:       exit(0);
3302:     }
3303:     else if(j >= (m_rec.cycl * 4))
3304:     {
3305:       MessageBox("LabelError in calcEn(high)");
3306:       exit(0);
3307:     }
3308:     // その周期の範囲± g_nExtendWidth の範囲の時刻、位置を取得
3309:     lp_time[i] = m_simple.dDiffTime[j];
3310:     lp_posi_x[i] = m_simple.x[j];
3311:     lp_posi_y[i] = m_simple.y[j];
3312:     lp_posi_z[i] = m_simple.z[j];
3313:   }
3314:   // 対象となる、波形を比較する 2 郡を取得
3315:   // x,y,z 軸において、各差をスプライン補間により計算
3316:   lp_x = new double[lp_before->nData+2*g_nExtendWidth];
3317:   lp_y = new double[lp_before->nData+2*g_nExtendWidth];
3318:   lp_z = new double[lp_before->nData+2*g_nExtendWidth];
3319:
3320:   // n-1 周期目の波形を補間
3321:   splineMakeTable(lp_before->nData+2*g_nExtendWidth,
3322:     lp_time,
3323:     lp_posi_x,
3324:     lp_x);
3325:   splineMakeTable(lp_before->nData+2*g_nExtendWidth,
3326:     lp_time,
3327:     lp_posi_y,
3328:     lp_y);
3329:   splineMakeTable(lp_before->nData+2*g_nExtendWidth,
3330:     lp_time,
3331:     lp_posi_z,
3332:     lp_z);
3333:
3334:   tmpSumOfNumerator=0;
3335:   tmpSumOfDenominator=0;
3336:   deltaNumerator=0;
3337:   deltaDenominator=0;
3338:
3339:   for(i=0;i<lp_after->nData;i++)
3340:   {
3341:     // t' の計算
3342:     dTDash = splineMakeTDashDiscrete(
3343:       lp_after->dTime[i],
3344:       lp_after->dEndTime,
3345:       lp_after->dStartTime,
3346:       lp_before->dEndTime,
3347:       lp_before->dStartTime
3348:       );
3349:
3350:     // Xn-1,Yn-1,Zn-1 の計算
3351:     dXnMinus = splineSpline(dTDash,
3352:       lp_before->nData+2*g_nExtendWidth,
3353:       lp_time,
3354:       lp_posi_x,
3355:       lp_x);
3356:     dYnMinus = splineSpline(dTDash,
3357:       lp_before->nData+2*g_nExtendWidth,
3358:       lp_time,
3359:       lp_posi_y,
3360:       lp_y);
3361:     dZnMinus = splineSpline(dTDash,
3362:       lp_before->nData+2*g_nExtendWidth,
3363:       lp_time,
3364:       lp_posi_z,
3365:       lp_z);
3366:
3367:     deltaNumerator =
3368:       pow(lp_after->posi_x[i] - dXnMinus, 2) +
3369:       pow(lp_after->posi_y[i] - dYnMinus, 2) +
3370:       pow(lp_after->posi_z[i] - dZnMinus, 2);
3371:     tmpSumOfNumerator += deltaNumerator;
3372:
3373:     deltaDenominator =
3374:       pow(lp_after->posi_x[i] - dBarXn, 2) +
3375:       pow(lp_after->posi_y[i] - dBarYn, 2) +
3376:       pow(lp_after->posi_z[i] - dBarZn, 2);
3377:     tmpSumOfDenominator += deltaDenominator;
3378:
3379:   }
3380:
3381:   // 各 En を計算
3382:   return tmpSumOfNumerator/tmpSumOfDenominator;
3383: }
3384:
3385: // m_meanSHIFT に従って描画
3386: int CRecordPlayMidiDlg::drawYourLength()
3387: {
3388:   int lnWidth;    // 横幅
3389:   int lnHeight;    // 縦幅
3390:   int i;
3391:   double left=0;
3392:   int fromx, fromy;    // 始点
3393:   int tox, toy;      // 終点
3394:
3395:   // デバイスコンテクストの指定
3396:   // 音長
3397:   CWnd* barCorrect = GetDlgItem(IDC_YOUR);
3398:   CClientDC noteDC(barCorrect);
3399:   // 不安定性
3400:   CWnd* strInstable = GetDlgItem(IDC_INSTABLE);
3401:   CRect instableRect;
3402:   strInstable->GetClientRect(instableRect);
3403:   CClientDC instableDC(strInstable);
3404:   // 前の結果を塗りつぶし
3405:   instableDC.FillSolidRect(instableRect,
 RGB(192,192,192));
3406:
3407:   // フォント
3408:   CFont barFont;
3409:   barFont.CreateFont(18, 0, 0, 0, FW_BOLD,
3410:     0,0,0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
3411:     CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH|
3412:     FF_DONTCARE, "Century");
3413:
3414:   // PEN
3415:   CPen blackLine(PS_SOLID,1,RGB(0,0,0));
3416:   CPen* notePen = noteDC.SelectObject(&blackLine);
3417:
3418:   // BRUSH
3419:   CBrush* p_noteBrush;
3420:   p_noteBrush = new CBrush[g_nNotePerBar];
3421:
3422:   // 小節全体の四角形
3423:   CRect allBar;
3424:   barCorrect->GetClientRect(allBar);
3425:
3426:   // 縦幅、横幅の取得
3427:   lnWidth = (int)allBar.Width();
3428:   lnHeight = (int)allBar.Height();
3429:
3430:   // 各音符分の四角形
3431:   CRect *noteRect;
3432:   noteRect = new CRect[g_nNotePerBar];
3433:
3434:   CString msgString;
3435:   // 四角形を一つずつ記述
3436:   for(i=0;i<g_nNotePerBar;i++)
3437:   {
3438:     fromx = (int)left;
3439:     fromy = 0;
3440:     tox = (int)(left + g_dTemplateNotes[i]
 * (double)lnWidth * ((double)1 + (double)g_meanSHIFT[i]/100));
3441:     if(i==g_nNotePerBar-1)
3442:     {
3443:       tox = lnWidth;
3444:     }
3445:     toy = lnHeight;
3446:
3447:     // 色の設定
3448:     if(g_meanSHIFT[i] > 4)
3449:     {
3450:       if(g_meanSHIFT[i] > 24)
3451:       {
3452:         //  "かなり長い "場合
3453:         p_noteBrush[i].CreateSolidBrush(RGB(128,0,0));
3454:         noteDC.SetTextColor(RGB(255,255,255));
3455:       }
3456:       else if(g_meanSHIFT[i] > 12)
3457:       {
3458:         //  "長い "場合
3459:         p_noteBrush[i].CreateSolidBrush(RGB(144,64,64));
3460:         noteDC.SetTextColor(RGB(255,255,255));
3461:       }
3462:       else if(g_meanSHIFT[i] > 4)
3463:       {
3464:         //  "やや長い "場合
3465:         p_noteBrush[i].CreateSolidBrush(RGB(192,128,128));
3466:         noteDC.SetTextColor(RGB(0,0,0));
3467:       }
3468:       msgString.Format("+%.1f%%\r\n 長め",g_meanSHIFT[i]);
3469:     }
3470:     else if(g_meanSHIFT[i] < -4)
```

```
3471:     {
3472:       if(g_meanSHIFT[i] < -24)
3473:       {
3474:         //  "かなり短い "場合
3475:         p_noteBrush[i].CreateSolidBrush(RGB(0,128,0));
3476:         noteDC.SetTextColor(RGB(255,255,255));
3477:       }
3478:       else if(g_meanSHIFT[i] < -12)
3479:       {
3480:         //  "短い "場合
3481:         p_noteBrush[i].CreateSolidBrush(RGB(64,144,64));
3482:         noteDC.SetTextColor(RGB(255,255,255));
3483:       }
3484:       else if(g_meanSHIFT[i] < -4)
3485:       {
3486:         //  "やや短い "場合
3487:         p_noteBrush[i].CreateSolidBrush(RGB(128,192,128));
3488:         noteDC.SetTextColor(RGB(0,0,0));
3489:       }
3490:       msgString.Format("%.1f%%\r\n 短め",g_meanSHIFT[i]);
3491:     }
3492:     else
3493:     {
3494:       // long short の表示無い場合
3495:       p_noteBrush[i].CreateSolidBrush(RGB(0,192,255));
3496:       noteDC.SetTextColor(RGB(0,0,0));
3497:       msgString.Format("%.1f%%\r\n 適切",g_meanSHIFT[i]);
3498:     }
3499:     noteDC.SelectObject(p_noteBrush[i]);
3500:
3501:     // 四角形の描画
3502:     noteRect[i].SetRect(fromx, fromy, tox, toy);
3503:     noteDC.Rectangle(noteRect[i]);
3504:     noteDC.SelectObject(notePen);
3505:     noteDC.SelectObject(&barFont);
3506:     noteDC.SetBkMode(TRANSPARENT);// 背景を透明に
3507:     noteDC.DrawText(msgString, &noteRect[i], DT_CENTER);
3508:
3509:     // 不安定性の表示の範囲を指定
3510:     instableDC.SelectObject(&barFont);
3511:     instableDC.SetBkMode(TRANSPARENT);// 背景を透明に
3512:     msgString.Format("%.1f%%",g_sdSHIFT[i]);
3513:     instableDC.DrawText(msgString, &noteRect[i],
 DT_CENTER);
3514:
3515:     left = tox;
3516:   }
3517:
3518:   return 0;
3519:
3520: }
3521:
3522: //
3523: // ファイル、またはディレクトリが存在するかチェック
3524: //
3525: // パラメータ
3526: //    lpszFileName
3527: //      有効なパス名とファイル名が入った NULL で終わる文字列へ
3528: //      のポインタを指定
3529: //    bDirectory
3530: //      TRUE ならディレクトリの検索。FALSE ならファイルの検索。
3531: //
3532: // 戻り値
3533: //    ファイル、またはディレクトリが存在するなら TRUE
3534: //    それ以外は FALSE
3535: //
3536: bool CRecordPlayMidiDlg::IsExistFile(
LPCTSTR lpszFileName, bool bDirectory)
3537: {
3538:   WIN32_FIND_DATA ffd;
3539:
3540:   // 指定されたファイル名と一致するファイルを検索
3541:   HANDLE  hFindFile = FindFirstFile(lpszFileName, &ffd);
3542:
3543:   // ファイルハンドルが無効
3544:   if (hFindFile != INVALID_HANDLE_VALUE)
3545:   {
3546:     // 検索ハンドルをクローズ
3547:     // (hFindFile が INVALID_HANDLE_VALUE ならハンドルをクロー
ズ
3548:     //  する必要はない。)
3549:     FindClose(hFindFile);
3550:
3551:     // ディレクトリかどうかをチェック
3552:     if ( (bDirectory) &&
3553:       (ffd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
3554:       return TRUE;
3555:     // ファイルかどうかをチェック
3556:     if ( !(bDirectory) &&
3557:       !(ffd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
3558:       return TRUE;
3559:   }
3560:   return FALSE;
3561: }
3562:
3563:
3564: // 音声フィードバック
3565: int CRecordPlayMidiDlg::waveFeedBack()
3566: {
3567:   OnBarAdvice();
3568:   OnFormAdvice();
3569:   OnVeloAdvice();
3570:
3571:   return 0;
3572: }
3573:
3574: // 指定されたファイル名の wave ファイルを再生
3575: int CRecordPlayMidiDlg::wavePlay(CString fname)
3576: {
3577:   MCI_PLAY_PARMS myPParms;
3578:   MCI_GENERIC_PARMS myGParms;
3579:   MCIERROR error;
3580:
3581:   char buf[129];
3582:
3583:   m_openParms.lpstrDeviceType
 = (LPCTSTR)MCI_DEVTYPE_WAVEFORM_AUDIO;
3584:
3585:   // ファイル名指定
3586:   m_openParms.lpstrElementName = fname;
3587:
3588:   // wave デバイスをオープン
3589:   if(error = mciSendCommand(0, MCI_OPEN,
3590:     MCI_WAIT | MCI_OPEN_TYPE | MCI_OPEN_ELEMENT |
3591:     MCI_OPEN_TYPE_ID, (DWORD)&m_openParms))
3592:   {
3593:     mciGetErrorString(error, buf, sizeof(buf));
3594:     AfxMessageBox(buf,MB_OK);
3595:
3596:     return -1;
3597:   }
3598:
3599:   // 再生 (MCI_WAIT オプションをつけると制御が進まない)
3600:   myPParms.dwCallback = (DWORD)this->GetSafeHwnd();
3601:   myPParms.dwFrom = 0;
3602:   if(error = mciSendCommand(m_openParms.wDeviceID, MCI_PLAY,
3603:     MCI_WAIT|MCI_NOTIFY|MCI_FROM, (DWORD)&myPParms))
3604:   {
3605:     mciGetErrorString(error, buf, sizeof(buf));
3606:     AfxMessageBox(buf,MB_OK);
3607:
3608:     return -1;
3609:   }
3610:
3611:   // クローズ
3612:   if(error = mciSendCommand(m_openParms.wDeviceID, MCI_CLOSE,
3613:     MCI_WAIT, (DWORD)&myGParms))
3614:   {
3615:     mciGetErrorString(error, buf, sizeof(buf));
3616:     AfxMessageBox(buf,MB_OK);
3617:
3618:     return -1;
3619:   }
3620:
3621:   return 0;
3622: }
3623:
3624:
3625: int CRecordPlayMidiDlg::drawStrokeError()
3626: {
3627:   int lnWidth;    // 横幅
3628:   int lnHeight;    // 縦幅
3629:   int i;
3630:   double left=0;
3631:   int fromx, fromy;    // 始点
3632:   int tox, toy;      // 終点
3633:
3634:   int printMode = 0;  // 横幅をリズムと合わせる...0 手本と同
様...1
3635:
3636:   // デバイスコンテクストの指定
3637:   CWnd* barCorrect = GetDlgItem(IDC_STROKE);
3638:   CClientDC noteDC(barCorrect);
3639:
3640:   // フォント
3641:   CFont barFont;
3642:   barFont.CreateFont(14, 0, 0, 0, FW_BOLD,
3643:     0,0,0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
```

```
3644:     CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH|
3645:     FF_DONTCARE, "Century");
3646:
3647:   // PEN
3648:   CPen blackLine(PS_SOLID,1,RGB(0,0,0));
3649:   CPen* notePen = noteDC.SelectObject(&blackLine);
3650:
3651:   // BRUSH
3652:   CBrush* p_noteBrush;
3653:   p_noteBrush = new CBrush[g_nNotePerBar];
3654:
3655:   // 小節全体の四角形
3656:   CRect allBar;
3657:   barCorrect->GetClientRect(allBar);
3658:
3659:   // 縦幅、横幅の取得
3660:   lnWidth = (int)allBar.Width();
3661:   lnHeight = (int)allBar.Height();
3662:
3663:   // 各音符分の四角形
3664:   CRect *noteRect;
3665:   noteRect = new CRect[g_nNotePerBar];
3666:
3667:   CString msgString="安定";
3668:   // 四角形を一つずつ記述
3669:   for(i=0;i<g_nNotePerBar;i++)
3670:   {
3671:     fromx = (int)left;
3672:     fromy = 0;
3673:     if(printMode == 0)
3674:     {
3675:       // リズム誤差と縦軸の位置を合わせる
3676:       tox = (int)(left + g_dTemplateNotes[i]
 * (double)lnWidth * ((double)1 + (double)g_meanSHIFT[i]/100));
3677:     }
3678:     else if(printMode == 1)
3679:     {
3680:       // 縦軸の位置はデフォルトのまま
3681:       tox = (int)(left + g_dTemplateNotes[i]
 * (double)lnWidth * ((double)1));
3682:     }
3683:     if(i==g_nNotePerBar-1)
3684:     {
3685:       tox = lnWidth;
3686:     }
3687:     toy = lnHeight;
3688:
3689:     // 色の設定
3690:     if(g_meanEn[i] > 2)
3691:     {
3692:       //  "かなりずれている "場合
3693:       p_noteBrush[i].CreateSolidBrush(RGB(128,0,0));
3694:       noteDC.SetTextColor(RGB(255,255,255));
3695:       msgString.Format("%.3f\r\n ぶれ有",g_meanEn[i]);
3696:     }
3697:     else if(g_meanEn[i] > 1)
3698:     {
3699:       //  "ずれている "場合
3700:       p_noteBrush[i].CreateSolidBrush(RGB(144,64,64));
3701:       noteDC.SetTextColor(RGB(255,255,255));
3702:       msgString.Format("%.3f\r\n ぶれ有",g_meanEn[i]);
3703:     }
3704:     else if(g_meanEn[i] > 0.53)
3705:     {
3706:       //  "ややずれている "場合
3707:       p_noteBrush[i].CreateSolidBrush(RGB(192,128,128));
3708:       noteDC.SetTextColor(RGB(0,0,0));
3709:       msgString.Format("%.3f\r\n ぶれ有",g_meanEn[i]);
3710:     }
3711:     else
3712:     {
3713:       p_noteBrush[i].CreateSolidBrush(RGB(0,192,255));
3714:       noteDC.SetTextColor(RGB(0,0,0));
3715:       msgString.Format("%.3f\r\n 安定",g_meanEn[i]);
3716:     }
3717:     noteDC.SelectObject(p_noteBrush[i]);
3718:
3719:     // 四角形の描画
3720:     noteRect[i].SetRect(fromx, fromy, tox, toy);
3721:     noteDC.Rectangle(noteRect[i]);
3722:     noteDC.SelectObject(notePen);
3723:     noteDC.SelectObject(&barFont);
3724:     noteDC.SetBkMode(TRANSPARENT);// 背景を透明に
3725:     noteDC.DrawText(msgString, &noteRect[i], DT_CENTER);
3726:
3727:     left = tox;
3728:   }
3729:
3730:   return 0;
3731: }
3732:
3733: // 打拍の強さのグラフ描画
3734: int CRecordPlayMidiDlg::drawVelocity()
3735: {
3736:   int lnWidth;    // 横幅
3737:   int lnHeight;    // 縦幅
3738:   int lnSDWidth;    // 横幅
3739:   int lnSDHeight;    // 縦幅
3740:   int i;
3741:   double left=0;
3742:   int fromx, fromy;    // 始点
3743:   int tox, toy;      // 終点
3744:
3745:   int printMode = 0;  // 横幅をリズムと合わせる...0 手本と同
様...1
3746:
3747:   // デバイスコンテクストの指定
3748:   CWnd* barVelo = GetDlgItem(IDC_MEANVELOCITY);
3749:   CRect veloRect;
3750:   barVelo->GetClientRect(veloRect);
3751:   CClientDC noteDC(barVelo);
3752:   // 前の結果を塗りつぶし
3753:   noteDC.FillSolidRect(veloRect, RGB(255,255,255));
3754:   // ベロシティ偏差
3755:   CWnd* strsdVelo = GetDlgItem(IDC_SDVELOCITY);
3756:   CRect sdVeloRect;
3757:   strsdVelo->GetClientRect(sdVeloRect);
3758:   CClientDC sdVeloDC(strsdVelo);
3759:   // 前の結果を塗りつぶし
3760:   sdVeloDC.FillSolidRect(sdVeloRect, RGB(192,192,192));
3761:
3762:   // フォント
3763:   CFont barFont;
3764:   barFont.CreateFont(16, 0, 0, 0, FW_BOLD,
3765:     0,0,0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
3766:     CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH|
3767:     FF_DONTCARE, "Century");
3768:
3769:   // PEN
3770:   CPen blackLine(PS_SOLID,1,RGB(0,0,0));
3771:   CPen* notePen = noteDC.SelectObject(&blackLine);
3772:
3773:   // BRUSH
3774:   CBrush* p_noteBrush;
3775:   p_noteBrush = new CBrush[g_nNotePerBar];
3776:
3777:   // 小節全体の四角形
3778:   CRect allBar;
3779:   barVelo->GetClientRect(allBar);
3780:
3781:   // 縦幅、横幅の取得
3782:   lnWidth = (int)allBar.Width();
3783:   lnHeight = (int)allBar.Height();
3784:   lnSDWidth = (int)sdVeloRect.Width();
3785:   lnSDHeight = (int)sdVeloRect.Height();
3786:
3787:   // 各音符分の四角形
3788:   CRect *noteRect;
3789:   noteRect = new CRect[g_nNotePerBar];
3790:   // 各音符分の四角形
3791:   CRect *sdRect;
3792:   sdRect = new CRect[g_nNotePerBar];
3793:
3794:   CString msgString;
3795:   // 四角形を一つずつ記述
3796:   for(i=0;i<g_nNotePerBar;i++)
3797:   {
3798:     fromx = (int)left;
3799:     fromy = lnHeight - (int)((double)lnHeight
 * g_meanVn[i]/(double)127);
3800:     if(printMode==0)
3801:     {
3802:       tox = (int)(left + g_dTemplateNotes[i]
 * (double)lnWidth * ((double)1 + (double)g_meanSHIFT[i]/100))
 - 1;
3803:     }
3804:     else if(printMode==1)
3805:     {
3806:       tox = (int)(left + g_dTemplateNotes[i]
 * (double)lnWidth * ((double)1)) - 1;
3807:     }
3808:     if(i==g_nNotePerBar-1)
3809:     {
3810:       tox = lnWidth;
3811:     }
3812:     toy = lnHeight;
3813:
3814:     // 色の設定
```

```
3815:     p_noteBrush[i].CreateSolidBrush(RGB(0,0,128));
3816:     noteDC.SetTextColor(RGB(255,255,255));
3817:
3818:     msgString.Format("%.1f",g_meanVn[i]);
3819:     noteDC.SelectObject(p_noteBrush[i]);
3820:
3821:     // 四角形の描画
3822:     noteRect[i].SetRect(fromx, fromy, tox, toy);
3823:     noteDC.Rectangle(noteRect[i]);
3824:     noteDC.SelectObject(notePen);
3825:     noteDC.SelectObject(&barFont);
3826:     noteDC.SetBkMode(TRANSPARENT);// 背景を透明に
3827:     noteDC.DrawText(msgString, &noteRect[i], DT_CENTER);
3828:
3829:     // ベロシティ偏差の表示の範囲を指定
3830:     sdRect[i].SetRect(fromx, 0, tox, lnSDHeight);
3831:     sdVeloDC.SelectObject(&barFont);
3832:     sdVeloDC.SetBkMode(TRANSPARENT);// 背景を透明に
3833: //    msgString.Format("%.1f",g_sdVn[i]);
3834:     msgString.Format("%.1f%%",g_sdVn[i] / g_meanVn[i]
 * 100);
3835:     sdVeloDC.DrawText(msgString, &sdRect[i], DT_CENTER|
DT_VCENTER);
3836:
3837:     left = tox + 1;
3838:   }
3839:
3840:   return 0;
3841: }
3842:
3843: // リズムパターンを演奏
3844: int CRecordPlayMidiDlg::playRhythm(int nBar)
3845: {
3846:   int i,j,k;
3847:   MMRESULT ret;
3848:   DWORD dwMsg, dwMsg_click;
3849:   // 開始時間
3850:   struct _timeb start;
3851:   time_t starttime;
3852:   int startms;
3853:   double clockstart;
3854:   // 瞬間の時間
3855:   struct _timeb now;
3856:   time_t nowtime;
3857:   int nowms;
3858:   double clocknow;
3859:   double dNextNote=0;
3860:   int *nTmpVelocity;
3861:   double *dTmpNotes;
3862:   double tmp=0;
3863:
3864:   UpdateData(TRUE);
3865:
3866: //  return 0;
3867:
3868:   // 打拍用の音長データを作成
3869:   dTmpNotes = new double[g_nNotePerBar];
3870:   nTmpVelocity = new int[g_nNotePerBar];
3871:   for(k=0;k<g_nNotePerBar;k++)
3872:   {
3873:     dTmpNotes[k]=tmp;
3874:     tmp+=g_dTemplateNotes[k];
3875:
3876:     // ベロシティの作成
3877:     if(m_isSetAccent==0 || m_nAccentMode==0)
3878:     {// アクセントなし
3879:       nTmpVelocity[k]=63;
3880:     }
3881:     else if(g_accent[k]==1)
3882:     {// アクセントあり
3883:       nTmpVelocity[k]=93;
3884:     }
3885:     else if(g_accent[k]==0)
3886:     {
3887:       nTmpVelocity[k]=53;
3888:     }
3889:     else
3890:     {
3891:       MessageBox("fatal error in playRhythm().");
3892:       exit(0);
3893:     }
3894:   }
3895:
3896:   // テンポを取得
3897:   CButton* tempo90big=(CButton*)GetDlgItem(
IDC_TEMPO90_BIG);
3898:   CButton* tempo120big=(CButton*)GetDlgItem(
IDC_TEMPO120_BIG);
3899:   if(tempo90big->GetCheck()) {g_tempo=90;
g_nVelocity=63;}
3900:   if(tempo120big->GetCheck()) {g_tempo=120;
g_nVelocity=63;}
3901:   dwMsg_click = (g_nVelocity << 16) + 0x2599;// クリックの音

3902: //  dwMsg = (g_nVelocity << 16) + 0x1f99;// スネアの音
3903:
3904:   if(g_nState == IDLE)
3905:   {
3906:     // メトロノームボタンと開始ボタンを使用不可に
3907:     CButton* metro=(CButton*)GetDlgItem(IDC_TEMPO_PLAY);
3908:     CButton* kaishi=(CButton*)GetDlgItem(IDC_BUTTON_TEST);
3909:     metro->EnableWindow(false);
3910:     kaishi->EnableWindow(false);
3911:
3912:
3913:     // 開始時間取得
3914:     _ftime(&start);
3915:     starttime=start.time;
3916:     startms=start.millitm;
3917:     clockstart = (double)startms / 1000;
3918:
3919:     // nBar 小節の打拍
3920:     for(i=0;i<nBar+1;i++)
3921:     {
3922:       if(i==0)
3923:       {
3924:         // クリック４回
3925:         for(j=0;j<4;j++)
3926:         {
3927:           dNextNote = (double)j * 60 / g_tempo;
3928:
3929:           while(1)
3930:           {
3931:             _ftime(&now);
3932:             nowtime=now.time;
3933:             nowms=now.millitm;
3934:
3935:             clocknow = (double)(nowtime-starttime)
 + (double)nowms / 1000 - clockstart;
3936:
3937:             if(clocknow >= dNextNote)
3938:             {
3939:               // 音を鳴らしてブレイク
3940:               ret = midiOutShortMsg(gs_hMidiOut,
 dwMsg_click);
3941:               break;
3942:             }
3943:           }
3944:         }
3945:       }
3946:       else
3947:       {
3948:         // 音符
3949:         for(j=0;j<g_nNotePerBar;j++)
3950:         {
3951:           // 音符の長さ
3952:           dNextNote = ((double)i + dTmpNotes[j]) * 4 * 60
 / g_tempo;
3953:           // スネアの音
3954:           dwMsg = (nTmpVelocity[j] << 16) + 0x1f99;
3955:
3956:           while(1)
3957:           {
3958:             _ftime(&now);
3959:             nowtime=now.time;
3960:             nowms=now.millitm;
3961:
3962:             clocknow = (double)(nowtime-starttime)
 + (double)nowms / 1000 - clockstart;
3963:
3964:             if(clocknow >= dNextNote)
3965:             {
3966:               // 音を鳴らしてブレイク
3967:               ret = midiOutShortMsg(gs_hMidiOut, dwMsg);
3968:               break;
3969:             }
3970:           }
3971:         }
3972:       }
3973:     }
3974:
3975:     // 最後の１発
3976:     dNextNote = (double)i * 4 * 60 / g_tempo;
3977:     // スネアの音
3978:     dwMsg = (nTmpVelocity[0] << 16) + 0x1f99;
3979:     while(1)
3980:     {
3981:       _ftime(&now);
```

```
3982:       nowtime=now.time;
3983:       nowms=now.millitm;
3984:
3985:       clocknow = (double)(nowtime-starttime)
 + (double)nowms / 1000 - clockstart;
3986:
3987:       if(clocknow >= dNextNote)
3988:       {
3989:         // 音を鳴らしてブレイク
3990:         ret = midiOutShortMsg(gs_hMidiOut, dwMsg);
3991:         break;
3992:       }
3993:     }
3994:     // 再び使用可能に
3995:     metro->EnableWindow(true);
3996:     kaishi->EnableWindow(true);
3997:   }
3998:
3999:   return 0;
4000: }
4001:
4002: void CRecordPlayMidiDlg::OnBarAdvice()
4003: {
4004:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加して
ください
4005:   int k;
4006:   CString fname;
4007:
4008:   if(g_bIsAfter!=1)
4009:   {
4010:     MessageBox("打拍した後にこちらのボタンを押すと、リズムに対
しての、音声による診断を聞くことができます。");
4011:     return;
4012:   }
4013:
4014:   for(k=0;k<g_nNotePerBar;k++)
4015:   {
4016:     if( g_meanSHIFT[k] > 4  ||  g_meanSHIFT[k] < -4 )
4017:     {
4018:       fname.Format(WAVE_PATH + "/%d 番目の.wav",k+1);
4019:       wavePlay(fname);
4020:       if(g_dTemplateNotes[k] == 0.75 ){
4021:         wavePlay(WAVE_PATH + "/f2onpuga.wav");
4022:       } else if(g_dTemplateNotes[k] == 0.5 ){
4023:         wavePlay(WAVE_PATH + "/2onpuga.wav");
4024:       }else if(g_dTemplateNotes[k] == 0.375 ){
4025:         wavePlay(WAVE_PATH + "/f4onpuga.wav");
4026:       }else if(g_dTemplateNotes[k] == 0.25 ){
4027:         wavePlay(WAVE_PATH + "/4onpuga.wav");
4028:       }else if(g_dTemplateNotes[k] == 0.125 ){
4029:         wavePlay(WAVE_PATH + "/8onpuga.wav");
4030:       } else{
4031:         wavePlay(WAVE_PATH + "/onpuga.wav");
4032:       }
4033:     }
4034:     if( g_meanSHIFT[k] > 24 )
4035:     {
4036:       wavePlay(WAVE_PATH + "/kanari.wav");
4037:       wavePlay(WAVE_PATH + "/nagaidesu.wav");
4038:     } else if( g_meanSHIFT[k] > 12 ){
4039:       wavePlay(WAVE_PATH + "/nagaidesu.wav");
4040:     } else if( g_meanSHIFT[k] > 4 ){
4041:       wavePlay(WAVE_PATH + "/yaya.wav");
4042:       wavePlay(WAVE_PATH + "/nagaidesu.wav");
4043:     } else if( g_meanSHIFT[k] < -24 ){
4044:       wavePlay(WAVE_PATH + "/kanari.wav");
4045:       wavePlay(WAVE_PATH + "/mijikaidesu.wav");
4046:     } else if( g_meanSHIFT[k] < -12 ){
4047:       wavePlay(WAVE_PATH + "/mijikaidesu.wav");
4048:     } else if( g_meanSHIFT[k] < -4 ){
4049:       wavePlay(WAVE_PATH + "/yaya.wav");
4050:       wavePlay(WAVE_PATH + "/mijikaidesu.wav");
4051:     } else {
4052:     }
4053:   }
4054: }
4055:
4056: void CRecordPlayMidiDlg::OnFormAdvice()
4057: {
4058:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加して
ください
4059:   int k;
4060:   CString fname;
4061:   double instableFuri;
4062:
4063:   if(g_bIsAfter!=1)
4064:   {
4065:     MessageBox("打拍した後にこちらのボタンを押すと、右手の振り
方に対しての、音声による診断を聞くことができます。");
4066:     return;
4067:   }
4068:
4069:   for(k=0;k<g_nNotePerBar;k++)
4070:   {
4071:     // g_meanEn に対する音声フィードバック
4072:     if(g_meanEn[k] > 0.53)
4073:     {
4074:       fname.Format(WAVE_PATH + "/%d 番目の.wav",k+1);
4075:       wavePlay(fname);
4076:       if(g_dTemplateNotes[k] == 0.75 ){
4077:         wavePlay(WAVE_PATH + "/付点二分音符の.wav");
4078:       } else if(g_dTemplateNotes[k] == 0.5 ){
4079:         wavePlay(WAVE_PATH + "/二分音符の.wav");
4080:       }else if(g_dTemplateNotes[k] == 0.375 ){
4081:         wavePlay(WAVE_PATH + "/付点四分音符の.wav");
4082:       }else if(g_dTemplateNotes[k] == 0.25 ){
4083:         wavePlay(WAVE_PATH + "/四分音符の.wav");
4084:       }else if(g_dTemplateNotes[k] == 0.125 ){
4085:         wavePlay(WAVE_PATH + "/八分音符の.wav");
4086:       } else{
4087:         wavePlay(WAVE_PATH + "/onpuno.wav");
4088:       }
4089:       wavePlay(WAVE_PATH + "/打拍の後の.wav");
4090:       wavePlay(WAVE_PATH + "/手の振り方が.wav");
4091:       if(g_meanEn[k] > 2)
4092:       {
4093:         wavePlay(WAVE_PATH + "/kanari.wav");
4094:         wavePlay(WAVE_PATH + "/ぶれています.wav");
4095:       }else if(g_meanEn[k] > 1)
4096:       {
4097:         wavePlay(WAVE_PATH + "/ぶれています.wav");
4098:       }else
4099:       {
4100:         wavePlay(WAVE_PATH + "/yaya.wav");
4101:         wavePlay(WAVE_PATH + "/ぶれています.wav");
4102:       }
4103:     }
4104:   }
4105:
4106:
4107:   for(k=0;k<g_nNotePerBar;k++)
4108:   {
4109:     instableFuri = g_sdSn[k] / g_meanSn[k] * 100;
4110:
4111:     if(instableFuri > 7.41)
4112:     {
4113:       fname.Format(WAVE_PATH + "/%d 番目の.wav",k+1);
4114:       wavePlay(fname);
4115:       if(g_dTemplateNotes[k] == 0.75 ){
4116:         wavePlay(WAVE_PATH + "/付点二分音符の.wav");
4117:       } else if(g_dTemplateNotes[k] == 0.5 ){
4118:         wavePlay(WAVE_PATH + "/二分音符の.wav");
4119:       }else if(g_dTemplateNotes[k] == 0.375 ){
4120:         wavePlay(WAVE_PATH + "/付点四分音符の.wav");
4121:       }else if(g_dTemplateNotes[k] == 0.25 ){
4122:         wavePlay(WAVE_PATH + "/四分音符の.wav");
4123:       }else if(g_dTemplateNotes[k] == 0.125 ){
4124:         wavePlay(WAVE_PATH + "/八分音符の.wav");
4125:       } else{
4126:         wavePlay(WAVE_PATH + "/onpuno.wav");
4127:       }
4128:       wavePlay(WAVE_PATH + "/打拍の後の.wav");
4129:     }
4130:     // g_sdVn に対する音声フィードバック
4131:     if( instableFuri > 22.23 )
4132:     {
4133:       wavePlay(WAVE_PATH + "/手のふり幅が.wav");
4134:       wavePlay(WAVE_PATH + "/kanari.wav");
4135:       wavePlay(WAVE_PATH + "/不安定です.wav");
4136:     } else if( instableFuri > 14.82 ){
4137:       wavePlay(WAVE_PATH + "/手のふり幅が.wav");
4138:       wavePlay(WAVE_PATH + "/不安定です.wav");
4139:     } else if( instableFuri > 7.41 ){
4140:       wavePlay(WAVE_PATH + "/手のふり幅が.wav");
4141:       wavePlay(WAVE_PATH + "/yaya.wav");
4142:       wavePlay(WAVE_PATH + "/不安定です.wav");
4143:     } else{
4144:     }
4145:   }
4146: }
4147:
4148: void CRecordPlayMidiDlg::OnVeloAdvice()
4149: {
4150:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
4151:   int k;
4152:   double instableVelo;
4153:   CString fname;
```

```
4154:
4155:   if(g_bIsAfter!=1)
4156:   {
4157:     MessageBox("打拍した後にこちらのボタンを押すと、打拍の強さ
に関しての音声による診断を聞くことができます。");
4158:     return;
4159:   }
4160:
4161:   for(k=0;k<g_nNotePerBar;k++)
4162:   {
4163:     instableVelo = g_sdVn[k] / g_meanVn[k] * 100;
4164:
4165:     if(instableVelo > 7.41)
4166:     {
4167:       fname.Format(WAVE_PATH + "/%d 番目の.wav",k+1);
4168:       wavePlay(fname);
4169:       if(g_dTemplateNotes[k] == 0.75 ){
4170:         wavePlay(WAVE_PATH + "/付点二分音符の.wav");
4171:       } else if(g_dTemplateNotes[k] == 0.5 ){
4172:         wavePlay(WAVE_PATH + "/二分音符の.wav");
4173:       }else if(g_dTemplateNotes[k] == 0.375 ){
4174:         wavePlay(WAVE_PATH + "/付点四分音符の.wav");
4175:       }else if(g_dTemplateNotes[k] == 0.25 ){
4176:         wavePlay(WAVE_PATH + "/四分音符の.wav");
4177:       }else if(g_dTemplateNotes[k] == 0.125 ){
4178:         wavePlay(WAVE_PATH + "/八分音符の.wav");
4179:       } else{
4180:         wavePlay(WAVE_PATH + "/onpuno.wav");
4181:       }
4182:     }
4183:     // g_sdVn に対する音声フィードバック
4184:     if( instableVelo > 22.23 )
4185:     {
4186:       wavePlay(WAVE_PATH + "/音の強さが.wav");
4187:       wavePlay(WAVE_PATH + "/kanari.wav");
4188:       wavePlay(WAVE_PATH + "/不安定です.wav");
4189:     } else if( instableVelo > 14.82 ){
4190:       wavePlay(WAVE_PATH + "/音の強さが.wav");
4191:       wavePlay(WAVE_PATH + "/不安定です.wav");
4192:     } else if( instableVelo > 7.41 ){
4193:       wavePlay(WAVE_PATH + "/音の強さが.wav");
4194:       wavePlay(WAVE_PATH + "/yaya.wav");
4195:       wavePlay(WAVE_PATH + "/不安定です.wav");
4196:     }
4197:   }
4198: }
4199:
4200:
4201: int CRecordPlayMidiDlg::drawAccent()
4202: {
4203:   int i,allWidth;
4204:
4205:   printBarPict();
4206:   // アクセントの音符の場所に「>」を提示
4207:   CWnd* accentBar = GetDlgItem(IDC_BAR);
4208:   CRect accentRect;
4209:   accentBar->GetClientRect(accentRect);
4210:   CClientDC accentDC(accentBar);
4211:   accentDC.SetBkMode(TRANSPARENT);// 背景を透明に
4212:   allWidth = accentRect.Width();
4213:
4214:   CString strAccent = ">";
4215:
4216:   for(i=0;i<g_nNotePerBar;i++)
4217:   {
4218:     if(g_accent[i])
4219:     {
4220:       accentDC.TextOut(
(int)(g_rateAccent[m_nSelectedPattern-1][i]
 * (double)allWidth),0,strAccent);
4221:     }
4222:   }
4223:
4224:   return 0;
4225: }
4226:
4227: // アクセントセットメイン
4228: void CRecordPlayMidiDlg::setAccent()
4229: {
4230:   if(m_nAccentMode == 0)
4231:   {
4232:     // アクセントなし
4233:     return;
4234:   }
4235:   else if(m_nAccentMode == 1)
4236:   {
4237:     // 静的アクセント
4238:     setStaticAccent();
4239:   }
4240:   else if(m_nAccentMode == 2)
4241:   {
4242:     // ランダムアクセント
4243:     OnPutAccent();
4244:   }
4245:   else
4246:   {
4247:     MessageBox("error in setAccent()");
4248:     exit(0);
4249:   }
4250: }
4251:
4252: // 静的アクセントを表示
4253: int CRecordPlayMidiDlg::setStaticAccent()
4254: {
4255:   int i;
4256:
4257:   if(m_isSetAccent==1)
4258:   {
4259:     delete(g_accent);
4260:   }
4261:   g_accent = new bool[g_nNotePerBar];
4262:   srand( (unsigned)time( NULL ) );
4263:
4264:   // アクセントの場所を決定
4265:   for(i=0;i<g_nNotePerBar;i++)
4266:   {
4267:     g_accent[i] = g_setAccent[i];
4268:   }
4269:   drawAccent();
4270:   m_isSetAccent=1;
4271:
4272:   return 0;
4273: }
4274:
4275: void CRecordPlayMidiDlg::OnPutAccent()
4276: {
4277:   // TODO: この位置にコントロール通知ハンドラ用のコードを追加し
てください
4278:   int i,random;
4279:
4280:   UpdateData(true);
4281:
4282:   if(m_isSetAccent==1)
4283:   {
4284:     delete(g_accent);
4285:   }
4286:   g_accent = new bool[g_nNotePerBar];
4287:   srand( (unsigned)time( NULL ) );
4288:
4289:   // アクセントの場所を決定
4290:   for(i=0;i<g_nNotePerBar;i++)
4291:   {
4292:     random = rand()%g_nNotePerBar + 1;//1〜4
4293:     if(random == 1)
4294:     {
4295:       g_accent[i] = 1;
4296:     }
4297:     else
4298:     {
4299:       g_accent[i] = 0;
4300:     }
4301:   }
4302:   // 描画
4303:   drawAccent();
4304:   m_isSetAccent=1;
4305: }
4306:
4307: // y 成分のみの Rn を計算
4308: double CRecordPlayMidiDlg::calcOneRyn(char *here)
4309: {
4310:   positionForCalc * lp_here = (positionForCalc *)here;
4311:
4312:   // 座標用
4313:   double dBarYn;
4314:   double tmpSum;
4315:   double delta;
4316:
4317:   int i;
4318:   // Bar の計算
4319:   dBarYn=0;
4320:
4321:   for(i=0;i<lp_here->nData;i++)
4322:   {
4323:     dBarYn += (double)lp_here->posi_y[i];
4324:   }
4325:   dBarYn = dBarYn / lp_here->nData;
4326:
4327:   tmpSum=0;
```

```
4328:   delta=0;
4329:
4330:   for(i=0;i<lp_here->nData;i++)
4331:   {
4332:     // y 座標のみ
4333:     delta = pow(lp_here->posi_y[i] - dBarYn, 2);
4334:     tmpSum += delta;
4335:   }
4336:   tmpSum /= lp_here->nData;
4337:   tmpSum = sqrt(tmpSum);
4338:   // 各 Rn を計算
4339:   return tmpSum;
4340: }
4341:
4342: // リズムパターン選択ボタンをすべて使用可能／不可能にする
4343: void CRecordPlayMidiDlg::enableRhythmButton(BOOL bEnable)
4344: {
4345:   int i;
4346:   CButton* tmp;
4347:
4348:   for(i=IDC_MINI1;i<=IDC_MINI8;i++)
4349:   {
4350:     tmp = (CButton*)GetDlgItem(i);
4351:     tmp->EnableWindow(bEnable);
4352:   }
4353: }
4354:
4355: // formCustomize で作成されたフォームデータを保存
4356: void CRecordPlayMidiDlg::autoSaveCForm()
4357: {
4358:   FILE*  fp;
4359:   int    i;
4360:   time_t  tim;
4361:   char lfilename[100];
4362:
4363:   time(&tim);
4364:   CTime  ctim(tim);
4365:
4366:   sprintf(lfilename,"%s/%s/%d_%d_%dcustomform",
 DATA_PATH, m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1]);
4367:   fp = fopen(lfilename, "w");
4368:
4369:   fprintf(fp, "DATA: Customized DiffTime x y z\n");
4370:   for(i=0;i<m_rec.cycl * 4;i++)
4371:   {
4372:     fprintf(fp, "%f %d %d %d\n",m_simple.dDiffTime[i],
 m_simple.x[i], m_simple.y[i], m_simple.z[i]);
4373:   }
4374:   fclose(fp);
4375: }
4376:
4377: //
4378: void CRecordPlayMidiDlg::autoSaveTotal()
4379: {
4380:   FILE* fp;
4381:   // 音符ごとの平均値
4382:   double t_meanSHIFT=0;
4383:   double t_sdSHIFT=0;
4384:   double t_meanEn=0;
4385:   double t_sdEn=0;
4386:   double t_meanRn=0;
4387:   double t_sdRn=0;
4388:   double t_meanSn=0;
4389:   double t_sdSn=0;
4390:   double t_meanTn=0;
4391:   double t_sdTn=0;
4392:   double t_meanVn=0;
4393:   double t_sdVn=0;
4394:   char lfilename[100];
4395:
4396:   int i;
4397:
4398:   // 音符当たりの平均値を取る
4399:   for(i=0;i<g_nNotePerBar;i++)
4400:   {
4401:     t_meanSHIFT += fabs(g_meanSHIFT[i]);
4402:     t_sdSHIFT += g_sdSHIFT[i];
4403:     t_meanEn += g_meanEn[i];
4404:     t_sdEn += g_sdEn[i];
4405:     t_meanRn += g_meanRn[i];
4406:     t_sdRn += g_sdRn[i];
4407:     t_meanSn += g_meanSn[i];
4408:     t_sdSn += g_sdSn[i];
4409:     t_meanTn += g_meanTn[i];
4410:     t_sdTn += g_sdTn[i];
4411:     t_meanVn += g_meanVn[i];
4412:     t_sdVn += g_sdVn[i]/g_meanVn[i]*100;
4413:   }
4414:   t_meanSHIFT /= g_nNotePerBar;
4415:   t_sdSHIFT /= g_nNotePerBar;
4416:   t_meanEn /= g_nNotePerBar;
4417:   t_sdEn /= g_nNotePerBar;
4418:   t_meanRn /= g_nNotePerBar;
4419:   t_sdRn /= g_nNotePerBar;
4420:   t_meanSn /= g_nNotePerBar;
4421:   t_sdSn /= g_nNotePerBar;
4422:   t_meanTn /= g_nNotePerBar;
4423:   t_sdTn /= g_nNotePerBar;
4424:   t_meanVn /= g_nNotePerBar;
4425:   t_sdVn /= g_nNotePerBar;
4426:
4427:   sprintf(lfilename,"%s/totaldat.txt", DATA_PATH);
4428:   fp = fopen(lfilename, "a");
4429:   fprintf(fp,"%s\t%d\t%d\t%d\t%.6f\t%.6f\t%.6f\t%.6f
\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\n",
4430:     m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1],
4431:     t_meanSHIFT, t_sdSHIFT,
4432:     t_meanEn, t_sdEn,
4433:     t_meanRn, t_sdRn,
4434:     t_meanSn, t_sdSn,
4435:     t_meanTn, t_sdTn,
4436:     t_meanVn, t_sdVn);
4437:   fclose(fp);
4438: }
4439:
4440: // 個人データログ
4441: void CRecordPlayMidiDlg::autoSaveSelfLog()
4442: {
4443:   FILE* fp;
4444:   int k;
4445:   char lfilename[100];
4446:   CString fname;
4447:   double instableVelo;
4448:   double instableFuri;
4449:
4450:   sprintf(lfilename,"%s/%s/%d_%d_%dself", DATA_PATH,
 m_sUserName, g_tempo, m_nSelectedPattern,
 g_nPerformTime[m_nSelectedPattern-1]);
4451:   fp = fopen(lfilename, "w");
4452:
4453:   // Rhythm フィードバック
4454:   for(k=0;k<g_nNotePerBar;k++)
4455:   {
4456:     if( g_meanSHIFT[k] > 4  ||  g_meanSHIFT[k] < -4 )
4457:     {
4458:       fprintf(fp,"## %dbanmeno\n",k+1);
4459:       if(g_dTemplateNotes[k] == 0.75 ){
4460:         fprintf(fp,"## f2onpuga\n");
4461:       } else if(g_dTemplateNotes[k] == 0.5 ){
4462:         fprintf(fp,"## 2onpuga\n");
4463:       }else if(g_dTemplateNotes[k] == 0.375 ){
4464:         fprintf(fp,"## f4onpuga\n");
4465:       }else if(g_dTemplateNotes[k] == 0.25 ){
4466:         fprintf(fp,"## 4onpuga\n");
4467:       }else if(g_dTemplateNotes[k] == 0.125 ){
4468:         fprintf(fp,"## 8onpuga\n");
4469:       } else{
4470:         fprintf(fp,"## onpuga\n");
4471:       }
4472:     }
4473:     if( g_meanSHIFT[k] > 24 )
4474:     {
4475:       fprintf(fp,"## kanari\n");
4476:       fprintf(fp,"## ldesu\n");
4477:     } else if( g_meanSHIFT[k] > 12 ){
4478:       fprintf(fp,"## ldesu\n");
4479:     } else if( g_meanSHIFT[k] > 4 ){
4480:       fprintf(fp,"## yaya\n");
4481:       fprintf(fp,"## ldesu\n");
4482:     } else if( g_meanSHIFT[k] < -24 ){
4483:       fprintf(fp,"## kanari\n");
4484:       fprintf(fp,"## sdesu\n");
4485:     } else if( g_meanSHIFT[k] < -12 ){
4486:       fprintf(fp,"## sdesu\n");
4487:     } else if( g_meanSHIFT[k] < -4 ){
4488:       fprintf(fp,"## yaya\n");
4489:       fprintf(fp,"## sdesu\n");
4490:     } else {
4491:       //
4492:     }
4493:   }
4494:
4495:
4496:   // g_meanEn に対する音声フィードバック
4497:   for(k=0;k<g_nNotePerBar;k++)
4498:   {
```

```
4499:     if(g_meanEn[k] > 0.53)
4500:     {
4501:       fprintf(fp,"## %dbanmeno\n",k+1);
4502:       if(g_dTemplateNotes[k] == 0.75 ){
4503:         fprintf(fp,"## f2onpuno\n");
4504:       } else if(g_dTemplateNotes[k] == 0.5 ){
4505:         fprintf(fp,"## 2onpuno\n");
4506:       }else if(g_dTemplateNotes[k] == 0.375 ){
4507:         fprintf(fp,"## f4onpuno\n");
4508:       }else if(g_dTemplateNotes[k] == 0.25 ){
4509:         fprintf(fp,"## 4onpuno\n");
4510:       }else if(g_dTemplateNotes[k] == 0.125 ){
4511:         fprintf(fp,"## 8onpuno\n");
4512:       } else{
4513:         fprintf(fp,"## onpuno\n");
4514:       }
4515:       fprintf(fp,"## dahakunoatono\n");
4516:       fprintf(fp,"## tenofurikataga\n");
4517:       if(g_meanEn[k] > 2)
4518:       {
4519:         fprintf(fp,"## kanari\n");
4520:         fprintf(fp,"## bureteimasu\n");
4521:       }else if(g_meanEn[k] > 1.06)
4522:       {
4523:         fprintf(fp,"## bureteimasu\n");
4524:       }else
4525:       {
4526:         fprintf(fp,"## yaya\n");
4527:         fprintf(fp,"## bureteimasu\n");
4528:       }
4529:     }
4530:   }
4531:
4532:   // ふり幅のフィードバック
4533:   for(k=0;k<g_nNotePerBar;k++)
4534:   {
4535:     instableFuri = g_sdSn[k] / g_meanSn[k] * 100;
4536:
4537:     if(instableFuri > 7.41)
4538:     {
4539:       fprintf(fp,"## %dbanmeno\n",k+1);
4540:       if(g_dTemplateNotes[k] == 0.75 ){
4541:         fprintf(fp,"## f2onpuno\n");
4542:       } else if(g_dTemplateNotes[k] == 0.5 ){
4543:         fprintf(fp,"## 2onpuno\n");
4544:       }else if(g_dTemplateNotes[k] == 0.375 ){
4545:         fprintf(fp,"## f4onpuno\n");
4546:       }else if(g_dTemplateNotes[k] == 0.25 ){
4547:         fprintf(fp,"## 4onpuno\n");
4548:       }else if(g_dTemplateNotes[k] == 0.125 ){
4549:         fprintf(fp,"## 8onpuno\n");
4550:       } else{
4551:         fprintf(fp,"## onpuno\n");
4552:       }
4553:       fprintf(fp,"## dahakunoatono\n");
4554:     }
4555:     // g_sdVn に対する音声フィードバック
4556:     if( instableFuri > 22.23 )
4557:     {
4558:       fprintf(fp,"## tenofurihabaga\n");
4559:       fprintf(fp,"## kanari\n");
4560:       fprintf(fp,"## fuanteidesu\n");
4561:     } else if( instableFuri > 14.82 ){
4562:       fprintf(fp,"## tenofurihabaga\n");
4563:       fprintf(fp,"## fuanteidesu\n");
4564:     } else if( instableFuri > 7.41 ){
4565:       fprintf(fp,"## tenofurihabaga\n");
4566:       fprintf(fp,"## yaya\n");
4567:       fprintf(fp,"## fuanteidesu\n");
4568:     } else{
4569:     }
4570:   }
4571:
4572:
4573:   // SD(Velocity) のフィードバック
4574:   for(k=0;k<g_nNotePerBar;k++)
4575:   {
4576:     instableVelo = g_sdVn[k] / g_meanVn[k] * 100;
4577:
4578:     if(instableVelo > 7.41)
4579:     {
4580:       fprintf(fp,"## %dbanmeno\n",k+1);
4581:       if(g_dTemplateNotes[k] == 0.75 ){
4582:         fprintf(fp,"## f2onpuno\n");
4583:       } else if(g_dTemplateNotes[k] == 0.5 ){
4584:         fprintf(fp,"## 2onpuno\n");
4585:       }else if(g_dTemplateNotes[k] == 0.375 ){
4586:         fprintf(fp,"## f4onpuno\n");
4587:       }else if(g_dTemplateNotes[k] == 0.25 ){
4588:         fprintf(fp,"## 4onpuno\n");
4589:       }else if(g_dTemplateNotes[k] == 0.125 ){
4590:         fprintf(fp,"## 8onpuno\n");
4591:       } else{
4592:         fprintf(fp,"## onpuno\n");
4593:       }
4594:     }
4595:     // g_sdVn に対する音声フィードバック
4596:     if( instableVelo > 22.23 )
4597:     {
4598:       fprintf(fp,"## otonotsuyosaga\n");
4599:       fprintf(fp,"## kanari\n");
4600:       fprintf(fp,"## fuanteidesu\n");
4601:     } else if( instableVelo > 14.82 ){
4602:       fprintf(fp,"## otonotsuyosaga\n");
4603:       fprintf(fp,"## fuanteidesu\n");
4604:     } else if( instableVelo > 7.41 ){
4605:       fprintf(fp,"## otonotsuyosaga\n");
4606:       fprintf(fp,"## yaya\n");
4607:       fprintf(fp,"## fuanteidesu\n");
4608:     } else{
4609:     }
4610:   }
4611:
4612:
4613:   // データ記録
4614:   for(k=0;k<g_nNotePerBar;k++)
4615:   {
4616:     fprintf(fp,"# meanSHIFT[%d] sdSHIFT[%d] %f %f\n",
4617:       k,k,g_meanSHIFT[k], g_sdSHIFT[k]);
4618:   }
4619:   for(k=0;k<g_nNotePerBar;k++)
4620:   {
4621:     fprintf(fp,"# meanEn[%d] sdEn[%d] %f %f\n",
4622:       k,k,g_meanEn[k], g_sdEn[k]);
4623:   }
4624:   for(k=0;k<g_nNotePerBar;k++)
4625:   {
4626:     fprintf(fp,"# meanRn[%d] sdRn[%d] %f %f\n",
4627:       k,k,g_meanRn[k], g_sdRn[k]);
4628:   }
4629:   for(k=0;k<g_nNotePerBar;k++)
4630:   {
4631:     fprintf(fp,"# meanSn[%d] sdSn[%d] %f %f\n",
4632:       k,k,g_meanSn[k], g_sdSn[k]);
4633:   }
4634:   for(k=0;k<g_nNotePerBar;k++)
4635:   {
4636:     fprintf(fp,"# meanTn[%d] sdTn[%d] %f %f\n",
4637:       k,k,g_meanTn[k], g_sdTn[k]);
4638:   }
4639:   for(k=0;k<g_nNotePerBar;k++)
4640:   {
4641:     fprintf(fp,"# meanVn[%d] sdVn[%d] %f %f\n",
4642:       k,k,g_meanVn[k], g_sdVn[k]);
4643:   }
4644:
4645:   fclose(fp);
4646: }
4647:
4648:
4649: // ふり幅を計算
4650: double CRecordPlayMidiDlg::calcOneFurihaba(char *here)
4651: {
4652:   positionForCalc * lp_here = (positionForCalc *)here;
4653:
4654:   // 座標用
4655:   double minYn;
4656:   double maxYn;
4657:   double tmpSum;
4658:
4659:   int i;
4660:   // Bar の計算
4661:   minYn = (double)lp_here->posi_y[0];
4662:   maxYn = (double)lp_here->posi_y[0];
4663:
4664:   for(i=1;i<lp_here->nData;i++)
4665:   {
4666:     if((double)lp_here->posi_y[i] < minYn)
4667:     {
4668:       minYn = (double)lp_here->posi_y[i];
4669:     }
4670:     if((double)lp_here->posi_y[i] > maxYn)
4671:     {
4672:       maxYn = (double)lp_here->posi_y[i];
4673:     }
4674:   }
4675:
4676:   tmpSum = maxYn - minYn;
```

```
4677:   // 各 Rn を計算
4678:   return tmpSum;
4679: }
4680:
4681: void CRecordPlayMidiDlg::drawFurihaba()
4682: {
4683:   int lnWidth;    // 横幅
4684:   int lnHeight;    // 縦幅
4685:   int lnSDWidth;    // 横幅
4686:   int lnSDHeight;    // 縦幅
4687:   int i;
4688:   double left=0;
4689:   int fromx, fromy;    // 始点
4690:   int tox, toy;      // 終点
4691:
4692:   int printMode = 0;  // 横幅をリズムと合わせる...0 手本と同
様...1
4693:
4694:   // デバイスコンテクストの指定
4695:   CWnd* barVelo = GetDlgItem(IDC_MEANFURIHABA);
4696:   CRect veloRect;
4697:   barVelo->GetClientRect(veloRect);
4698:   CClientDC noteDC(barVelo);
4699:   // 前の結果を塗りつぶし
4700:   noteDC.FillSolidRect(veloRect, RGB(255,255,255));
4701:   // ベロシティ偏差
4702:   CWnd* strsdVelo = GetDlgItem(IDC_SDFURIHABA);
4703:   CRect sdVeloRect;
4704:   strsdVelo->GetClientRect(sdVeloRect);
4705:   CClientDC sdVeloDC(strsdVelo);
4706:   // 前の結果を塗りつぶし
4707:   sdVeloDC.FillSolidRect(sdVeloRect, RGB(192,192,192));
4708:
4709:   // フォント
4710:   CFont barFont;
4711:   barFont.CreateFont(16, 0, 0, 0, FW_BOLD,
4712:     0,0,0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
4713:     CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH|
4714:     FF_DONTCARE, "Century");
4715:
4716:   // PEN
4717:   CPen blackLine(PS_SOLID,1,RGB(0,0,0));
4718:   CPen* notePen = noteDC.SelectObject(&blackLine);
4719:
4720:   // BRUSH
4721:   CBrush* p_noteBrush;
4722:   p_noteBrush = new CBrush[g_nNotePerBar];
4723:
4724:   // 小節全体の四角形
4725:   CRect allBar;
4726:   barVelo->GetClientRect(allBar);
4727:
4728:   // 縦幅、横幅の取得
4729:   lnWidth = (int)allBar.Width();
4730:   lnHeight = (int)allBar.Height();
4731:   lnSDWidth = (int)sdVeloRect.Width();
4732:   lnSDHeight = (int)sdVeloRect.Height();
4733:
4734:   // 各音符分の四角形
4735:   CRect *noteRect;
4736:   noteRect = new CRect[g_nNotePerBar];
4737:   // 各音符分の四角形
4738:   CRect *sdRect;
4739:   sdRect = new CRect[g_nNotePerBar];
4740:
4741:   CString msgString;
4742:   // 四角形を一つずつ記述
4743:   for(i=0;i<g_nNotePerBar;i++)
4744:   {
4745:     fromx = (int)left;
4746:     // 30cm を基準にする
4747:     fromy = lnHeight - (int)((double)lnHeight
 * g_meanSn[i]/(double)300);
4748:     if(fromy < 0) fromy=0;
4749:     if(printMode==0)
4750:     {
4751:       tox = (int)(left + g_dTemplateNotes[i]
 * (double)lnWidth * ((double)1
 + (double)g_meanSHIFT[i]/100)) - 1;
4752:     }
4753:     else if(printMode==1)
4754:     {
4755:       tox = (int)(left + g_dTemplateNotes[i]
 * (double)lnWidth * ((double)1)) - 1;
4756:     }
4757:     if(i==g_nNotePerBar-1)
4758:     {
4759:       tox = lnWidth;
4760:     }
4761:     toy = lnHeight;
4762:
4763:     // 色の設定
4764:     p_noteBrush[i].CreateSolidBrush(RGB(0,0,128));
4765:     noteDC.SetTextColor(RGB(255,255,255));
4766:
4767:     msgString.Format("%.1f",g_meanSn[i]/10);
4768:     noteDC.SelectObject(p_noteBrush[i]);
4769:
4770:     // 四角形の描画
4771:     noteRect[i].SetRect(fromx, fromy, tox, toy);
4772:     noteDC.Rectangle(noteRect[i]);
4773:     noteDC.SelectObject(notePen);
4774:     noteDC.SelectObject(&barFont);
4775:     noteDC.SetBkMode(TRANSPARENT);// 背景を透明に
4776:     noteDC.DrawText(msgString, &noteRect[i], DT_CENTER);
4777:
4778:     // ベロシティ偏差の表示の範囲を指定
4779:     sdRect[i].SetRect(fromx, 0, tox, lnSDHeight);
4780:     sdVeloDC.SelectObject(&barFont);
4781:     sdVeloDC.SetBkMode(TRANSPARENT);// 背景を透明に
4782:     msgString.Format("%.1f",g_sdSn[i]);
4783:     msgString.Format("%.1f%%",g_sdSn[i] / g_meanSn[i]
 * 100);
4784:     sdVeloDC.DrawText(msgString, &sdRect[i], DT_CENTER|
DT_VCENTER);
4785:
4786:     left = tox + 1;// グラフの間に隙間を
4787:   }
4788: }
```