2011 年度　修士論文

スポーツコーチングを目指した
スポーツ映像データベースシステムの構築

早稲田大学　大学院スポーツ科学研究科

スポーツ科学専攻　身体運動科学研究領域

5010A026-3

川畑　直也

研究指導教員：誉田　雅彰　教授

目次

1 緒言

1.1 研究の背景

スポーツコーチングにおいて映像を用いることの利点はチームの方向性を共有することである。つまりスポーツコーチングの場面で映像という可視的なツールは選手との意思の疎通を図ることに役立つのである。さらに映像の中であるシーンの統計を取ることは指導方針を支える根拠にもなる。日本女子バレーボールではどの選手のスパイク成功率が高いか、またどのエリアにスパイクしたときに高確率で決まるかなどをリアルタイムで戦術アナリストが監督に逐一教えて監督が選手にフィードバックするという活用も見られるようになった。その日本女子バレーの戦術アナリスト渡辺は次のように述べている。

「日本の女子バレーはサーブ、レシーブ、ミスの少なさなどで世界一になることを目指している。情報活用でも世界のトップになればさらに強みを発揮できる」という。優位性を見つけ、理論に裏付けられた戦術を実行すれば、身体能力の差をひっくり返すことも不可能ではない［1]。

このようにスポーツは身体能力や技能を競うのはもちろん情報活用を競う時代になってきたと推察できる。

この情報活用の方法としてデータベースというものがある。これは You tube などに挙げられるように自分の見たい動画を検索して見ることができるようなシステムである。これをスポーツにも応用すると自分の所望の選手やプレイを見て指導に役立てることができる。このシステムの利点は所望のシーンをすばやく見ることができることである。スポーツ映像という整理されていないものの場合、所望のシーンを見るには手間がかかる。そこでスポーツにおける映像データベースではタグ付けという工夫をする。タグ付けとはシーンの仕分け作業のようなもので、データベース作成の際に自分の所望のシーンにある名称をつけていく作業である。このようにすると後でその所望のシーンの名称を検索ワードとして検索するとそのシーンの候補が列挙されて自分の所望のシーンをすばやく見ることができる。

スポーツにおける映像データベースといっても競技特性により多様である。たとえばラグビーではシーンをすばやく再生することに加えて、全体の中であるシーンの割合を統計処理をするデータベースが見られる。この統計を取る意図として全体の中で割合が高いシーンがあった場合、そのシーンを想定した練習メニューに重点を置くなどが考えられる。サッカーでは選手の走った軌跡から移動距離を算出できるトラックパフォーマンス [2]と呼ばれるデータベースシステムもある。これは選手交代の基準の一つとして選手の移動距離を設けていると考えられる。バレーボールでは図 1.1 のようにスパイクの角度とスパイクが放たれた頻度を表すデータベースなどがある。

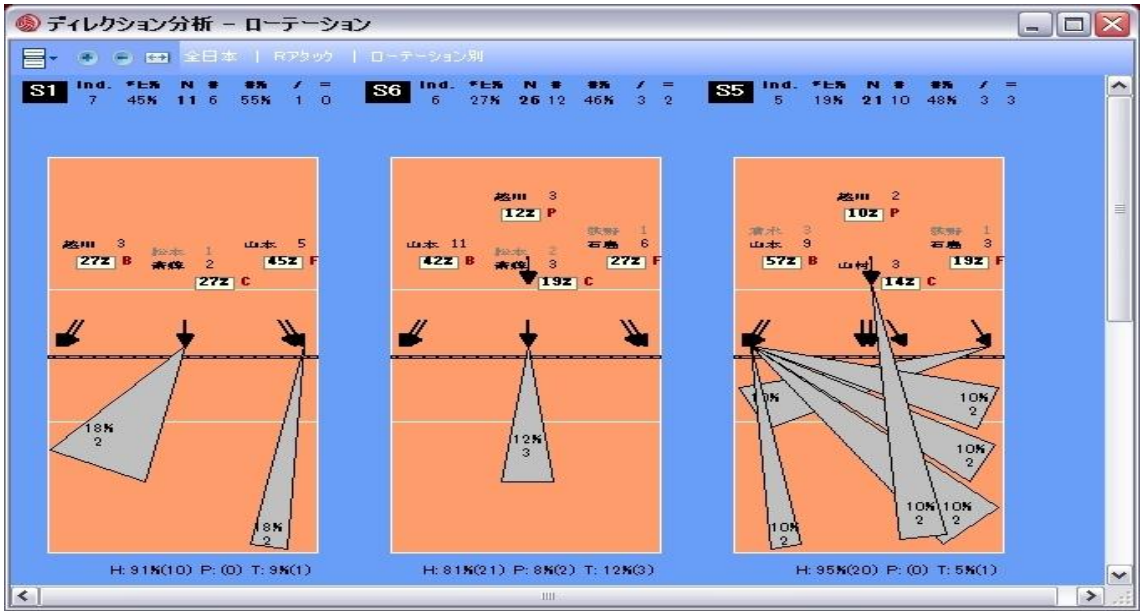


図 1.1　バレーのデータベースの一例
ホームページ unlimited.volleyball.ne.jp より

しかしながらスポーツにおける映像データベース作成の問題点として日本経済新聞では次のような指摘がされている。

ただ、こうした情報戦が進むと、人間の目で逐一プレイ内容を投入する情報だけでは追いつかなくなる。一部の競技では、データ入力を機械に任せる技術の導入が始まっている［1]。

このように映像データベースを構築する上で最大の問題は、タグ付けに膨大な手間がかかるということである。一般的に、タグ付けの作業は人が映像を見て映像シーンを特定し，そのプレイ内容を手動で入力するものである。このような手作業によってタグ付けを行う場合、実時間の数十倍から数百倍の時間を要することになり、所要時間を短縮するにはタグ付けに携わる作業者を増やすしか方法がなかった。

1.2 本研究のねらい

そこで本研究のねらいは映像データベースの作成におけるタグ付けを、コンピュータの情報処理技術を用いて自動化する手法を開発する。このようなシステムが構築できれば、映像データベースをごく短時間に自動的に構築できるようになる。

まずタグ付けを自動化するには、コンピュータがあるシーンの特徴を自動検出することが前提となる。コンピュータが映像データから自動検出するための手掛かりにするものとして一般的に画像情報と音情報がある。スポーツ映像の画像情報には、選手や観客やレフ

ェリーの画像情報が含まれている。一方、スポーツ映像の音情報には、歓声やアナウンサーの声(放送映像に限る)や選手の声、レフェリーの笛などか含まれる。これらの情報からシーン特徴を検出するには、画像情報であれば画像から色や形を取り出す画像処理技術が用いられ、音情報であれば、音の音響的特徴を取り出す音響信号処理技術が用いられる。画像情報には、プレイに関わる多くの情報を含んでおり、多彩な検出をすることができる反面、画像情報からプレイ内容を特定するには、膨大かつ複雑な画像処理が必要となる。一方音情報は、画像情報に比べてプレイに関わる多様な情報を含んでおらず、検出できる処理に限りがあるが、画像処理に比べてはるかにシーン検出に要する処理時間が短くてすむ。

また、本研究ではラグビーを対象としてレフェリーの笛検出を取り上げるが、ラグビーにおけるレフェリーの笛情報は次のようなキーとなるプレイの前に笛が吹かれる場合が多く、ラグビーのプレイシーンを検出する有効な手掛かりとなる。たとえば笛の鳴るシーンとして

①ミス→スクラム
②反則→ペナルティー
③トライキックオフ
④怪我人が出たときなどである

まずラグビーのデータベースとしてタグの自動化のニーズがあるのは得点シーンである。得点を競うゲームなので、このシーンは必要不可欠である。したがって笛検出によるトライシーンとペナルティーのシーンのタグづけはスポーツコーチングに有効である。スクラムは練習でしたことが試合にダイレクトに反映されやすく、自分の指導や選手の練習成果を評価する際に一つの評価観点として用いることができる。このように笛の鳴るシーンはプレイの全てではないが重要なシーンであることが多い。

1.3 音情報を用いた映像検索の先行研究

音情報から映像検索する研究は、これまでにもいくつか見られる。例えば、サッカーのゴールシーンにおいて歓声があがったとき、コンピュータがその歓声のあがった時間区間を認識し、映像に何らかの変化をもたらすことでテレビ視聴の雰囲気を変えるような装置である。また、他の研究では、ある動画から音声を取りだして音声を自動認識した後、話された内容の字幕を自動で付与するようなシステムの構築も見られる。そして、ユーザーが所望のシーンについて字幕の文字情報から該当する映像を再生できるということが目指されている。しかしながら、これまでの研究では、レフリーの笛の音からスポーツ映像を自動検索するような研究はあまり行われていない。

2 実験方法

2.1 放送映像における音情報

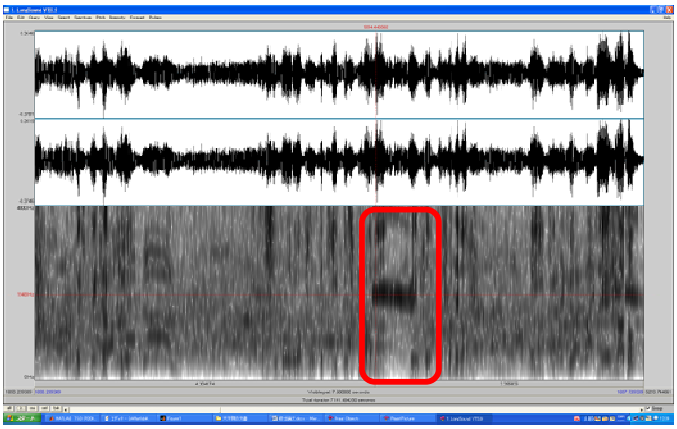
ラグビーの放送映像における音情報には、アナウンサーの声、歓声、レフェリーの声や笛、選手の声などが混在している。本研究では、このように混在する音情報の中からレフェリーの笛を自動検出することが必要になる。そのためには、まずレフェリーの笛特有の音響的特徴から考える必要がある。

2.2 笛の音響的特徴

2.2.1 音のスペクトル

音は空気の中を振動して対象物に届かせる。音の性質は一般的に①大きさ（音圧）、②高さ（周波数 Hz）、③音色（波形）に分けることができる。音が空気を通る際にその音の振動と周りの空気には圧力差が生じる。この圧力差を音圧と言い、音の大きさに関係してくる。音は振動して対象物に伝わり、その音の振動を音波という。音は波として一定の周期を持ち、この周期の逆数を周波数と呼ぶ。この周波数が高いと高い音になり、低いと低い音になる。

音色とは音信号に含まれる周波数成分とその強度の分布によって特徴づけられる。このような周波数強度分布をスペクトルと呼ぶ。図 2.1 は、音信号のスペクトルをスペクトログラムと呼ばれる表示法で示したものである。図の上段の 2 つの波形はステレオ録音されたただの音信号波形であり、下段がスペクトログラムを表している。スペクトログラムの横軸は時間、縦軸は周波数を表し、グレースケール画像の濃淡が各周波数と各時間における音信号のスペクトル強度を表している。このように、スペクトログラムは音色が時間的に変化する音信号の特徴を表すことに適している。図 2.1 に示した音信号には笛音やアナウンサー音声を含まれているが、スペクトログラム上に赤枠で囲まれた区間が濃い部分が笛音に対応している。



2.1 音波形とスペクトログラム

2.2.2 スペクトルピーク

笛の音響的特徴は、ラグビーの試合にもよるが、2000Hz 周辺に強い周波数成分が存在する。さらに図 2.1 はスペクトル強度を画像の濃淡で表しているが、図 2.2 は横軸を周波数、縦軸をスペクトル強度としてスペクトルを表示したものである。下図の例では、2251Hz の周波数成分が最も強い。ここでは、スペクトル強度が最も強くなる周波数成分をスペクトルピークと呼び、それに対応する周波数をスペクトルピーク周波数（略してピーク周波数）と呼ぶことにする。

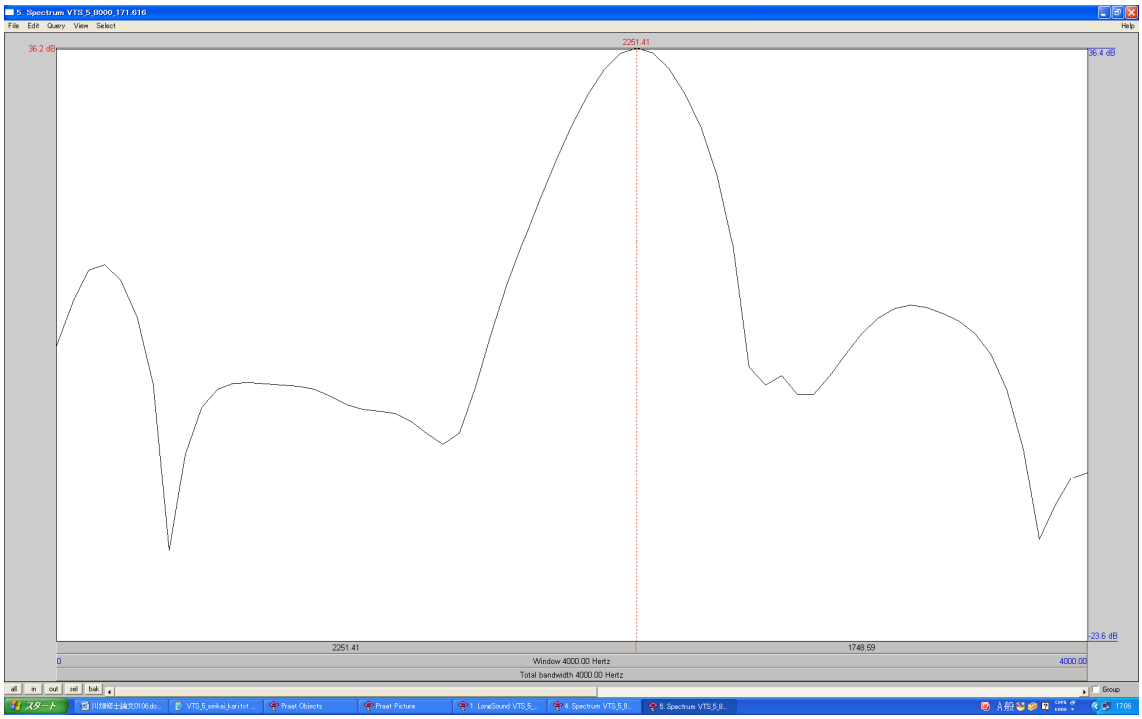


図 2.2 音スペクトルとスペクトルピーク

図 2.3 は、笛音の音波形とスペクトログラムを表し、スペクトログラム上の赤線はピーク周波数を表している。図 2.3 に示されるように、笛の場合にはピーク周波数の時間的変化は、一定の時間間隔で周期的に変動していることがわかる。このようなピーク周波数の時間的変化成分を FM(Frequency Modulation)成分と呼ぶことにする。一方、図 2.4 は、アナウンサー音声に対する音波形とスペクトログラムを示している。音声のスペクトルは、図中の黒い縞模様として見られるように、いくつかの周波数で強度が強くなり、特定の周波数成分のみが強い笛音の特徴とは異なっているのがわかる。また、図中の赤線で示されるピーク周波数の時間的変化には、ゆっくりとした変化成分と微細なランダム変化成分は見られるが、笛音のような周期的な FM 成分は見られない。なお、歓声のスペクトル特徴は音声のスペクトル特徴に近い。

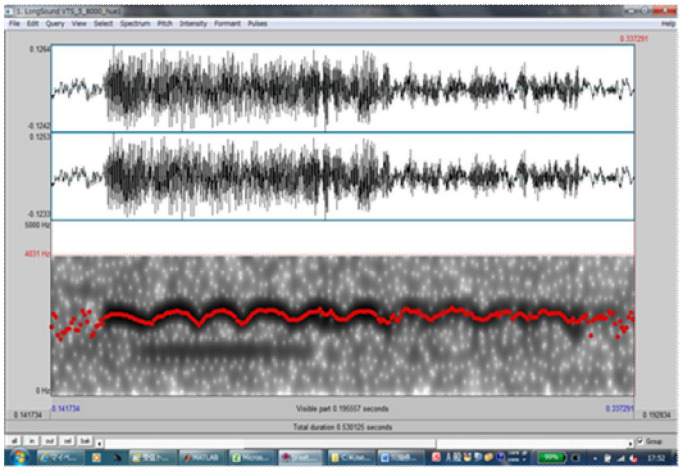


図 2.3 笛音のピーク周波数

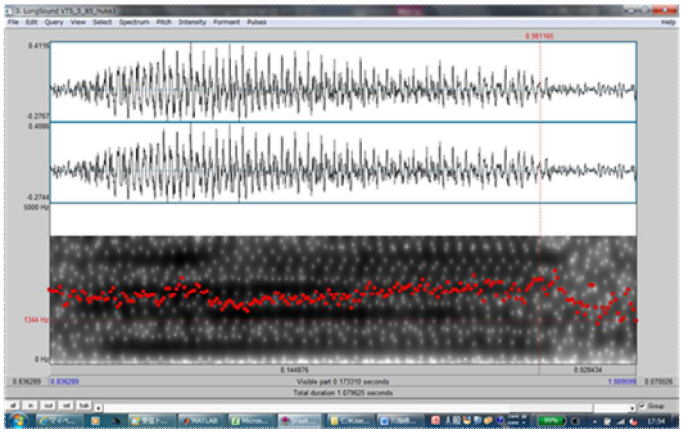


図 2.4 アナウンサー音声のピーク周波数

以上述べたように、笛音のスペクトルは、2000Hz 付近に集中したスペクトル強度と周期的に変化するピーク周波数（FM 成分）によって特徴づけられ、音声あるいは歓声とは際立った相違がみられる。

2.3 笛の特徴抽出

前節で述べた笛音の音響的特徴をもとに、笛音にアナウンサーの音声や歓声が混在した音から、コンピュータ処理によって笛音を自動的に検出する方法を検討する。

笛音の自動検出処理は、大きく笛音の音響的特徴を抽出する処理と、それらの音響的特徴を基づいて笛音区間を検出する検出処理からなる。ここでは、まず笛の特徴抽出処理について述べる。笛の特徴抽出処理として、以下の処理を行う。

① バンドパスフィルタを用いた笛の周波数分析処理

② FM 成分を抽出する処理

2.3.1 バンドパスフィルタ処理

フィルタとは信号の周波数成分を選択的に通過させる仕組みのことである。これを音響分析に応用すると、信号に含まれる特定の周波数帯域成分を通して、それ以外の帯域の周波数成分を遮断することができる。高域の周波数成分を通過させるタイプをハイパスフィルタ、低い周波数成分を通過させるタイプをローパスフィルタという。そしてある指定された周波数帯域区間だけを通してそれ以外を遮断するフィルタをバンドパスフィルタという。今回の場合、笛の検出を目的としているので、その周波数帯域は笛にもよるが、ほぼ2000Hz～2500Hz の周波数帯域になる。この帯域の周波数成分を通過させることが目的なのでバンドパスフィルタを用いる。

図 2.5 にバンドパスフィルタ処理のブロック図を示す。 音信号をバンドパスフィルタに入力し、その出力信号の振幅を二乗してその時間平均を取ったものをバンドパスフィルタ出力として算出する。今回の実験では、時間平均の時間窓長を 30ms に設定した。また、バンドパスフィルタの帯域に関しては、フィルタの中心周波数 fc とバンド幅 fb に関して、検出精度の面から実験的に検討した。

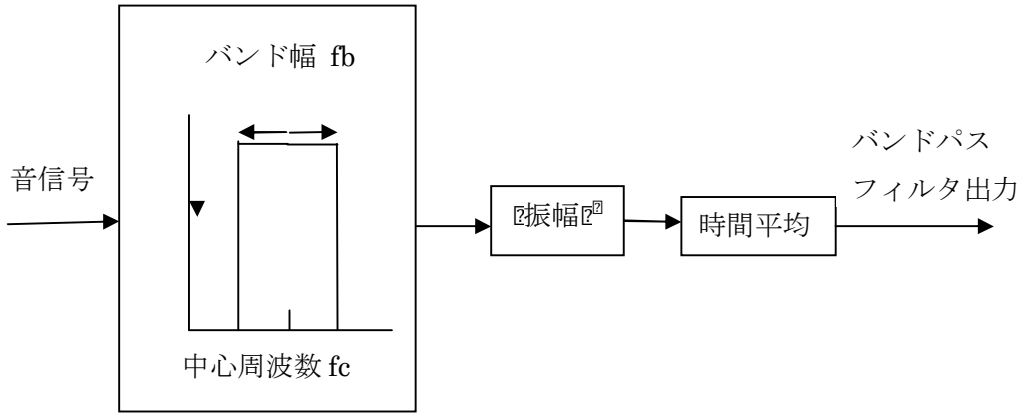


図 2.5 バンドパスフィルタ処理のブロック図

2.3.2 FM 成分の分析

図 2.6 は FM 成分の分析処理のブロック図を示したものである。まず、音信号をバンドパスフィルタに通した後、フーリエ変換を用いてスペクトルを算出し、スペクトルのピーク値を求め、ピーク周波数を決定する。このようなピーク検出（図 2.6 におけるピーク検出 I）によって、図 2.3 の赤線で示されたピーク周波数の時系列が求められる。

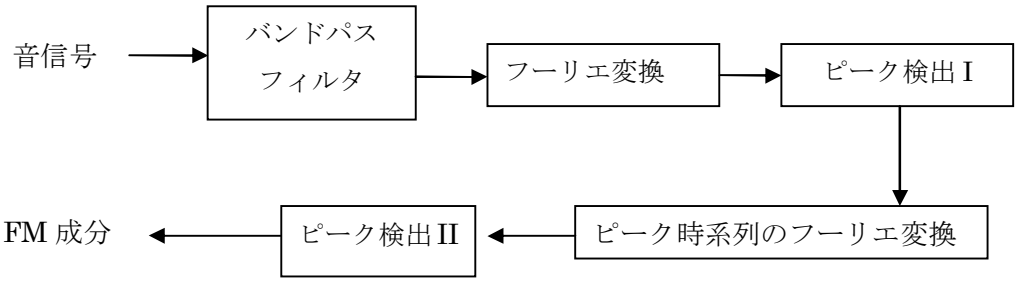


図 2.6 FM 成分の抽出処理

次に、笛音の場合は、図 2.3 に示されるように、ピーク周波数の時間的変化は周期的に変化する FM 成分によって特徴づけられる。この FM 成分を抽出するため、フーリエ変換を用いてピーク周波数時系列に対するスペクトルを算出し、そのスペクトルのピーク値を求める（図 2.6 のピーク検出 II)。このピーク値をここでは FM 成分と呼ぶことにする。笛音に対するピーク周波数時系列に対するスペクトルと FM 成分を図 2.7 の左図に示す。図に示されるように、笛音ではピーク周波数時系列のスペクトルには鋭いピークが存在することがわかる。このピーク値に対応する周波数（逆数としての周期）は、ピーク周波数の時間的変化の周期に対応している。

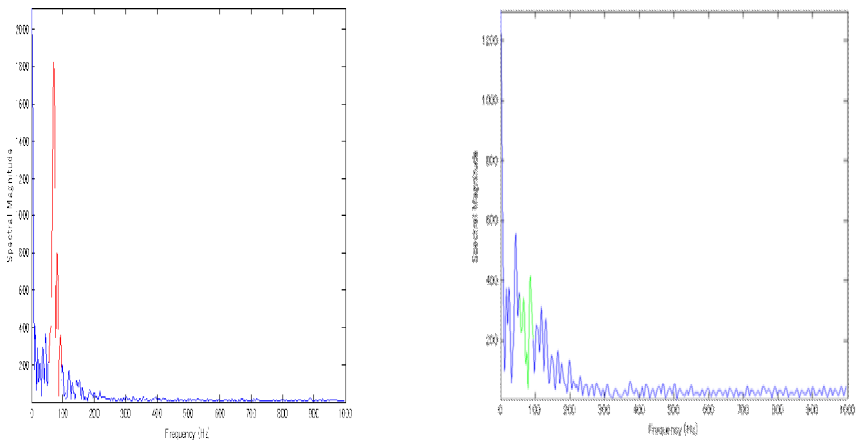


図 2.7 笛音（左）とアナウンサー音声（右）に対する
ピーク周波数時系列のスペクトルと FM 成分

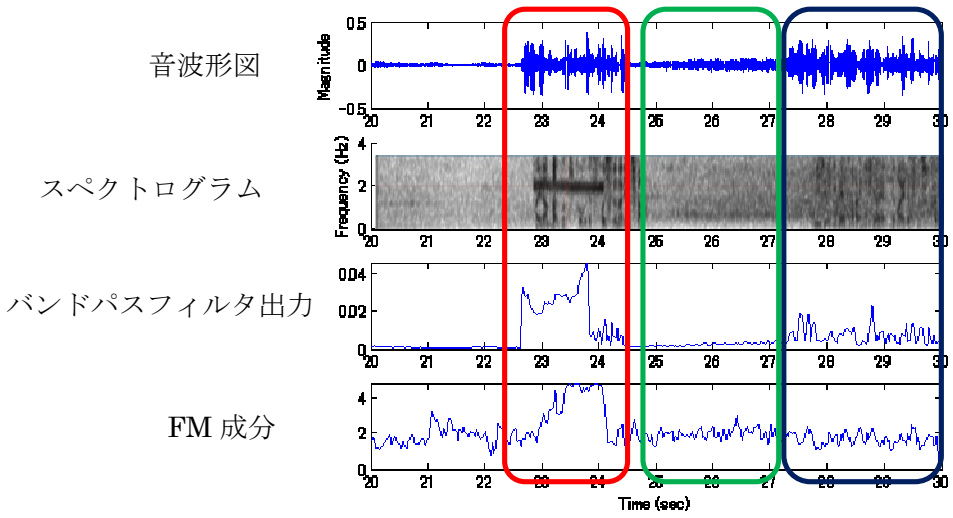


図 2.8　音の特徴抽出処理の例
赤枠：笛区間、緑枠：歓声、青枠：アナウンサー音声

一方、アナウンサー音声に対する FM 成分は、図 2.7 の右図に示すように、鋭いピークを示さない。これは、図 2.3 で示したように、ピーク周波数時系列が周期的に変化していないことに対応している。なお、上記 FM 成分の分析条件は、ピーク周波数を求めるスペクトル分析における分析窓長を　6ms、分析フレーム長を 0.5ms とした。また、ピーク周波数時系列のスペクトル分析における分析窓長を 128ms、分析フレーム長を 30m に設定した。

図 2.8 に音の特徴抽出処理の分析例を示す。図の上から、音波形、スペクトログラムバンドパスフィルタ出力、FM 成分を表している。また、図中の赤枠は笛＋歓声の区間、緑枠は歓声の区間、青枠はアナウンサー音声＋歓声の区間である。これより、笛が存在する区間では、バンドパスフィルター出力と FM 成分が共に他の区間と比較して大きな値になっているいることがわかる。本研究ではこの 2 つの笛区間の特徴を検出する。

2.4 笛区間の自動検出

2.4.1 笛区間の検出処理

ここでは、前節で述べた音響的特徴を用いて笛区間を検出する処理について述べる。図 2.9 は音信号から笛区間検出までの全体の処理の流れを示したものである。

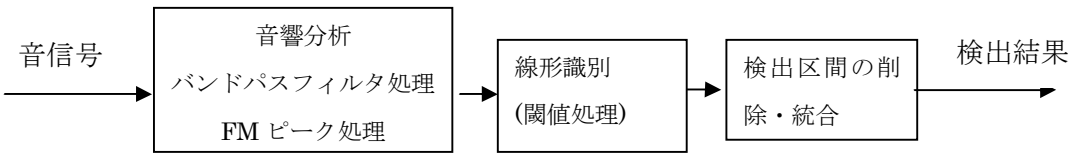


図 2.9 笛区間の自動検出処理のブロック図

図 2.10 のような笛検出処理において用いられる線形識別や閾値処理について説明する。例えばある基準の値を超える音信号を笛音、それを下回る音信号を笛以外の音にする二値化処理において、この基準の値を閾値と呼ぶ。具体的には、バンドパスフィルタ出力を閾値と比較して笛音かどうかをコンピュータに識別させる。図 2.10 に閾値処理の仕組みを示す。〇で表されているものが笛音×で表されているものが笛以外の音ということにする。

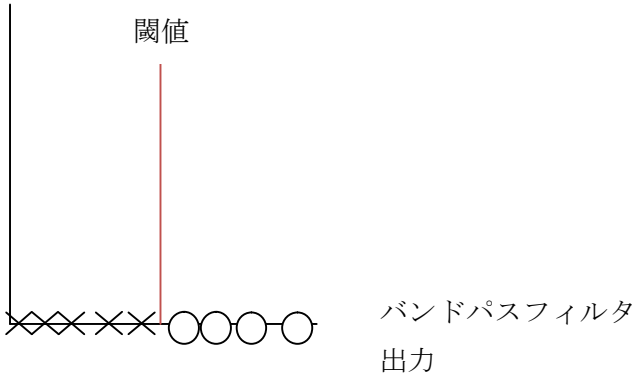


図 2.10 閾値処理による笛区間検出

音響特徴が 1 つの場合はその閾値より大きいか小さいかによって、笛区間とそれ以外の区間を識別することができるが、音響特徴が 2 つ以上の場合は、単純な閾値判定ではなく、図 2.11 のような直線を設定し、直線の上側か下側かによって識別する方法が用いられる。この方法を線形識別と呼び、直線を線形識別関数と呼ぶ。この線形識別関数は一般的に y＝ax□ bと表現することができる。図 2.11 において、横軸はバンドパスフィルタ出力、縦軸は FM 成分を表している。分析によって得られたバンドパスフィルタ出力を x、FM 成分を y とした時、y□ □□□ □の場合を笛音、y□ □□□ □の場合を笛以外の音としてある音信号を識別することができる。□と□ の値は、様々な値の組み合わせに関して検出実験を行い、検出性能を比較評価し最も性能が良いときの□と□の値に設定した。

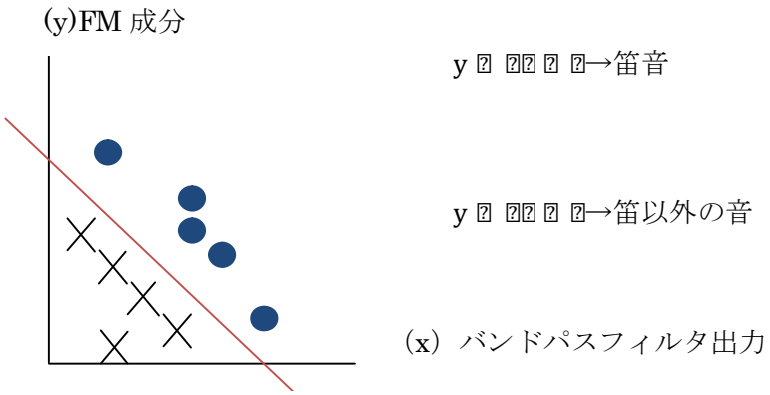


図 2.11 バンドパスフィルタ出力と FM ピークを用いる場合の線形識別処理

2.4.2 検出区間と削除・結合処理

前節で述べた笛識別処理によって得られる区間には、極めて時間長が短い区間が含まれる。また、離接する区間の間に極めて短いギャップ区間が存在する場合もある。実際に吹かれる笛の時間長には長短の違いは存在するものの、最も時間が短い場合でも 0.07 秒程度であり、それより短い時間長の区間は笛区間とは認められない。

検出区間の削除とは、笛音として検出した区間がある程度の時間長でなければ削除するというものである。これは笛以外の笛に似た音響的特徴をもった音信号が一瞬現れ、識別処理で笛と判定されてしまう場合への対策である。

検出区間の統合とは、笛音が断続して鳴らされた場合にその断続的に吹かれた笛を一つの笛として結合する処理のことである。この処理は正解データを作る際に断続的な笛音を一つに統合して正解区間として作成したことに対応したものである。正解区間を一つに統合すると笛検出もその正解区間にあわせる必要があった。このように合わせなければ 3 回断続的に吹かれた笛音に対して 3 つ検出され、正解データの正解が 1 つしかないと 100%を超えてしまう可能性があった。したがって 3 回断続的に吹かれた場所でもそれを 1 つの笛として検出させる処理が必要となった。具体的には、隣接する検出笛区間の間の時間長が設定値以下の場合は、2 つの区間を 1 つの区間に結合するようにした。

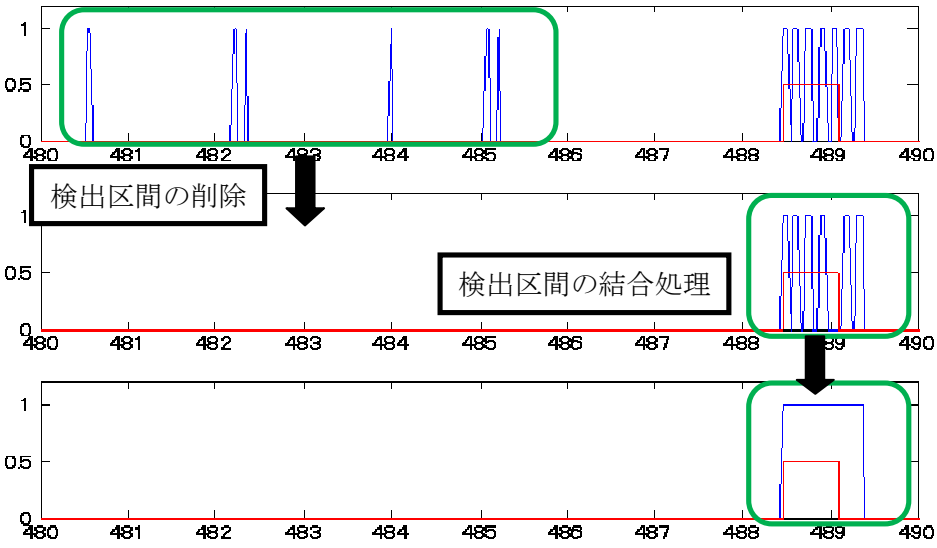


図 2.12 検出区間と削除・結合処理

2.5 映像データベースの自動作成

2.5.1 ELAN の機能

本研究では映像データベースを作成する際、ELAN というフリーソフトウェアを使用する。図 2.13 は，ELAN を用いた映像データベース作成の例を示したものである。ELAN では，コンピュータに取り込まれた映像データを表示しながら，スクロールバーを用いてシーンの区間を特定し，各シーンに対してプレイ内容を文字で手入力する。

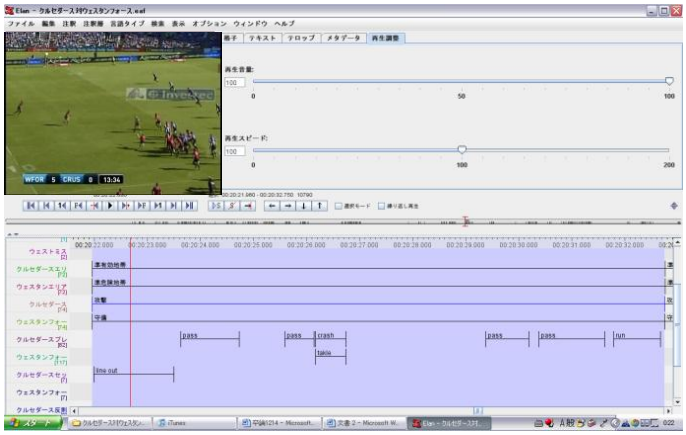


図 2.13 ELAN 映像データベース

このタグ付け映像データベースから所望の映像シーンを検索することが ELAN の主要機能である。

図 2.14 はクルセダーズというチームの攻撃シーンを列挙している。この列挙された候補の中から所望の映像シーンを検索することができる。

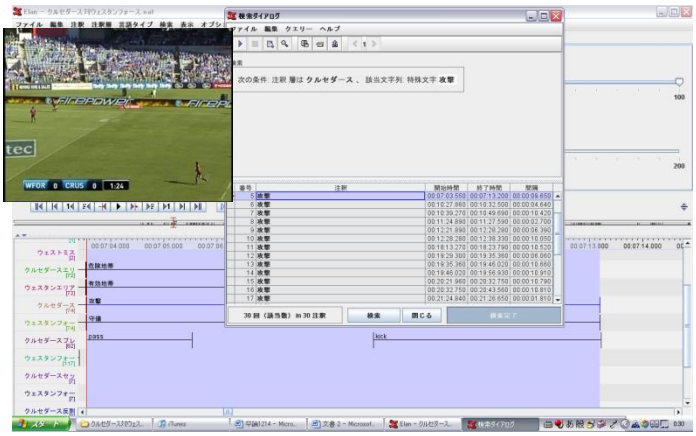


図 2.14 ELAN による映像シーンの検索

2.5.2 ELAN スクリプトについて

ELAN による映像データベースを作成する際は基本的に手動になるので、笛音の検出結果データを ELAN のみで自動で反映させることできない。したがって ELAN と笛音の検出結果データをリンクできるプログラムが必要になる。ELAN はスクリプト上に書かれた情報が反映されているため、一部のスクリプト情報を自動で書き換えることができれば、ELAN の映像データベース自動作成は可能となる。図 2.15 のようにサクラエディタと呼ばれるテキストファイルで ELAN スクリプトをみた場合、スクリプトの一部分に ELAN のタグ情報が反映されていることがわかる。

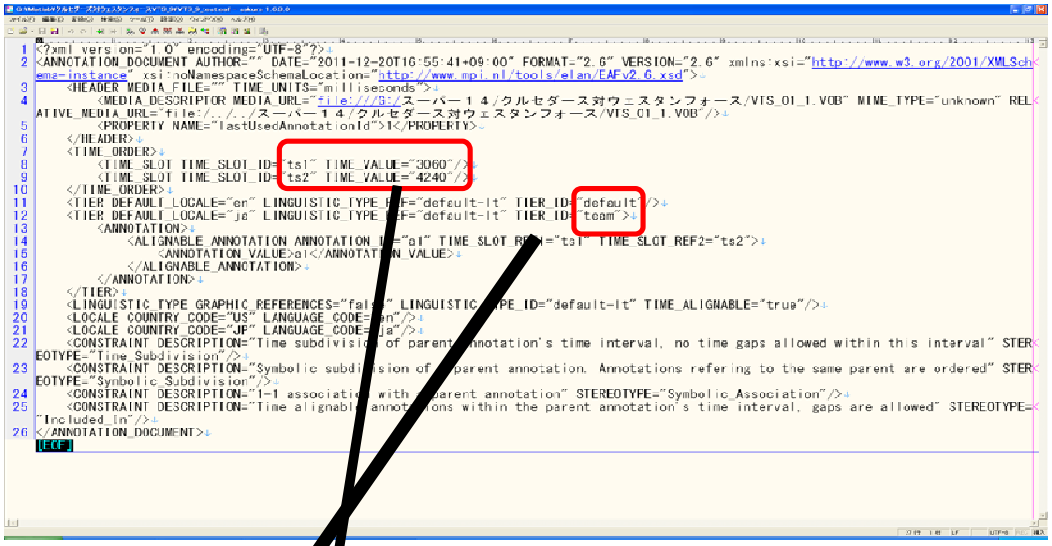


図 2.15 ELAN スクリプト

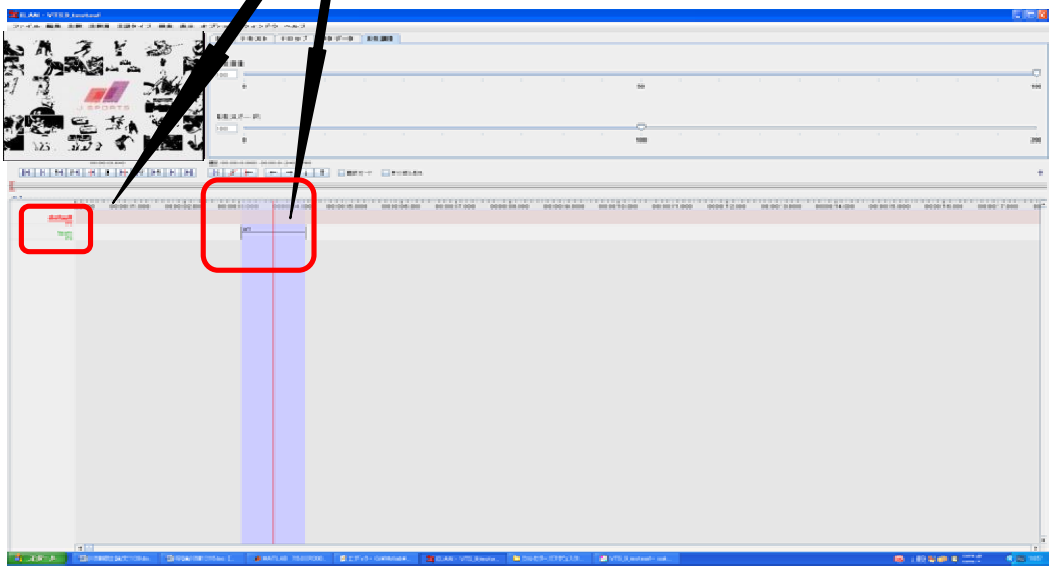


図 2.16　図 2.15 の ELAN スクリプトに対応する ELAN 画面

2.5.3 ELAN スクリプトの自動作成

このサクラエディタで見たような ELAN スクリプトの一部分を自動で書き換えられるようなプログラムを作成することで、ELAN 本体に検出データを反映させることができる。以下、ELAN スクリプトの自動作成方法について述べる。

まず、ELAN スクリプトにおいて映像ファイルを定義している部分の詳細を以下に示す。

```
<?xml version="1.0" encoding="UTF-8"?>
<ANNOTATION_DOCUMENT AUTHOR="" DATE="2011-11-01T12:15:26+09:00" FORMAT="2.7"
VERSION="2.7" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.mpi.nl/tools/elan/EAFv2.7.xsd">
<HEADER MEDIA_FILE="" TIME_UNITS="milliseconds">
    <MEDIA_DESCRIPTOR MEDIA_URL="file:///A" MIME_TYPE="video/*"
RELATIVE_MEDIA_URL="file://B"/>
    <PROPERTY NAME="lastUsedAnnotationId">1</PROPERTY>
</HEADER>
```

スクリプトの A の箇所にパスネームとファイルネーム、B の箇所にファイルネームを挿入する。次に、スクリプトにおいてタグ区間の開始時点と終了時点を定義している部分の詳細を以下に示す。

```
<TIME_ORDER>
    <TIME_SLOT TIME_SLOT_ID="C" TIME_VALUE="D"/>
    <TIME_SLOT TIME_SLOT_ID="E" TIME_VALUE="F"/>
</TIME_ORDER>
```

C に ts+シリアル番号、E に te+シリアル番号、D に開始時刻（単位は ms）、F に終了時刻(単位は ms)をそれぞれ挿入する。最後に、注釈層を定義している部分とタグを定義している部分の詳細を以下に示す。

```
<TIER LINGUISTIC_TYPE_REF="default-lt" TIER_ID="G">
    <ANNOTATION>
        <ALIGNABLE_ANNOTATION ANNOTATION_ID="a1" TIME_SLOT_REF1="H"
TIME_SLOT_REF2="I">
            <ANNOTATION_VALUE>??</ANNOTATION_VALUE>
        </ALIGNABLE_ANNOTATION>
    </ANNOTATION>
```

G に開始時刻番号 ts+シリアル番号、H に終了時刻番号 te+シリアル番号、I にタグ名をそれぞれ挿入する。

以上が ELAN スクリプトの内容である。MATLAB プログラムを用い、自動検出した笛の開始・終了時点のデータを ELAN スクリプトに挿入することにより、ELAN スクリプトを自動作成する。

3 実験

3.1 映像データ

実験で用いたのはラグビーのクラブチームの放送試合映像で 4 試合分を分析した。ハイランダーズ対レッズを VTS_5、ハリケーンズ対チフスを VTS_7、ブルス対ストーマーズを VTS_8、クルセダーズ対ウェスタンフォースを VTS_9 と表記する。

3.2 検出実験

映像データから 1 試合分の音データを抽出し、その音信号に対して笛検出を行う。音信号の音響分析処理および笛の検出処理は、MATLAB プログラムを作成し実行した。笛の検出実験では、笛の音響特徴として

① バンドパスフィルタ出力のみを用いる場合

② バンドパスフィルタ出力に加えて FM 成分を用いる場合

の 2 通りについて検討した。

また、検出性能に関係する以下のパラメータを用いて検出実験を行い、検出性能を比較評価した。

① バンドパスフィルタの中心周波数とバンド幅

③ バンドパスフィルタ出力に対する閾値

④ バンドパスフィルタ出力と FM 成分を音響特徴とする場合の線形識別関数

⑤ 笛区間削除処理における笛の最小時間継続長

⑥ 笛区間の最小ギャップ時間長

3.3 評価方法

3.3.1 笛区間正解データの作成

音分析ソフトウェア Praat を用い、音信号の聴取とスペクトログラムの視察によって笛区間の開始時点と終了時点を決定し、図 3.1 に示すような笛区間の正解データを含むテキストファイルに作成した。検出性能は、自動検出された笛区間を正解区間と照合することによって行う。

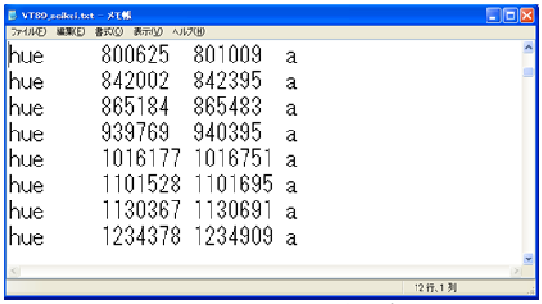


図 3.1 笛区間の正解データ

（2 列目が笛の開始時点、3 列目は笛の終了時点（単位は ms）

3.3.2 評価方法

図 3.2 は、笛区間の検出評価方法を示している。検出区間と正解区間に重なりがある場合は正しい検出（図 3.2 の左）とし、図正解区間が検出できない場合をミス誤り（図 3.2 の中央）とし、検出区間と正解区間に重なりがない場合は付加誤り（図 3.2 の右）とした。すなわち、検出誤りには、誤って正解区間でない区間を検出してしまう付加誤りと、本来の笛区間を検出し損なうミス誤りの 2 種類がある。検出率は、成功区間数＋付加誤り区間数＋ミス誤り区間数（総回数）に対する成功区間数の比率として算出した。また、付加誤り率は総区間数に対する付加誤り区間数の比率、ミス誤り率は総区間数に対するミス誤り区間数の比率として算出した。

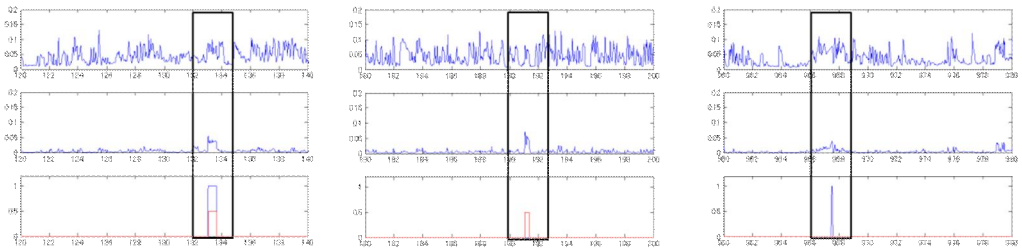


図 3.2 笛区間の検出評価方法　下段の赤線区間は正解区間、青線区間は検出区間
左：正しく検出された場合　　中央：ミス誤り　　右：付加誤り

4 実験結果

4.1 笛区間の特徴

4.1.1 笛の鳴らされた回数の統計

各試合と笛の数は以下のようになった。試合によってレフェリーが笛を吹く回数は異なるが、100 回前後笛を吹くことがわかる。

表 4.1 各試合の鳴らされた笛の回数

Game	VTS_5	VTS_7	VTS_8	VTS_9
number of whistle	129	101	96	93

4.1.2 笛によるシーン分類

図 4.1 は笛のシーンの原因の割合を表している。最も多いシーンは濃い紫色の etc（その他）である。etc（その他）は何らかの理由でプレイを中断するときやそこから再開する笛である。その理由は副審のアピール、特定の選手やキャプテンに対する警告、ポイントと違う場所でプレイを再開させようとしている選手の制止、けんかの仲裁などである。そして薄い紫は penalty（反則行為）である。三番目に多いのは knock on（落球）である。この三つのシーンで平均 7.3%を占めている。予想よりも etc が多く、レフィリーが中断させているケースがあることがわかった。そしてその割合は各試合によっても異なっている。

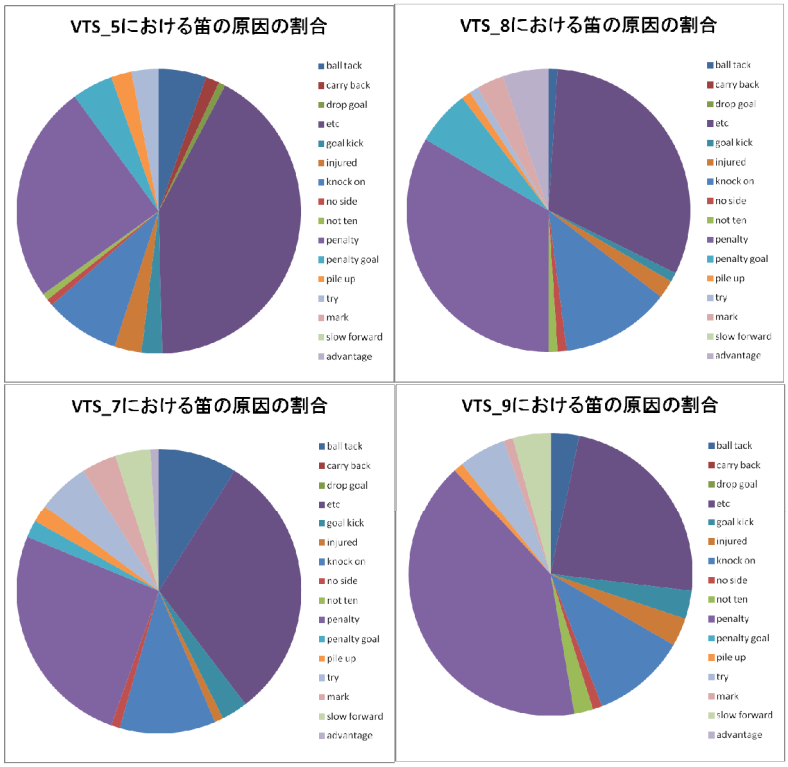


図 4.1 各試合の笛の原因のシーンの割合

図 4.2 は、笛とその結果起こるシーンを割合で示したものである。一番多いのは scrum（スクラム）となっている。二番目に多いのは tap kick である。tap kick とは penalty をされた場合に選択するプレイである。ボールを 30cm けり上げることで成立するプレイである。

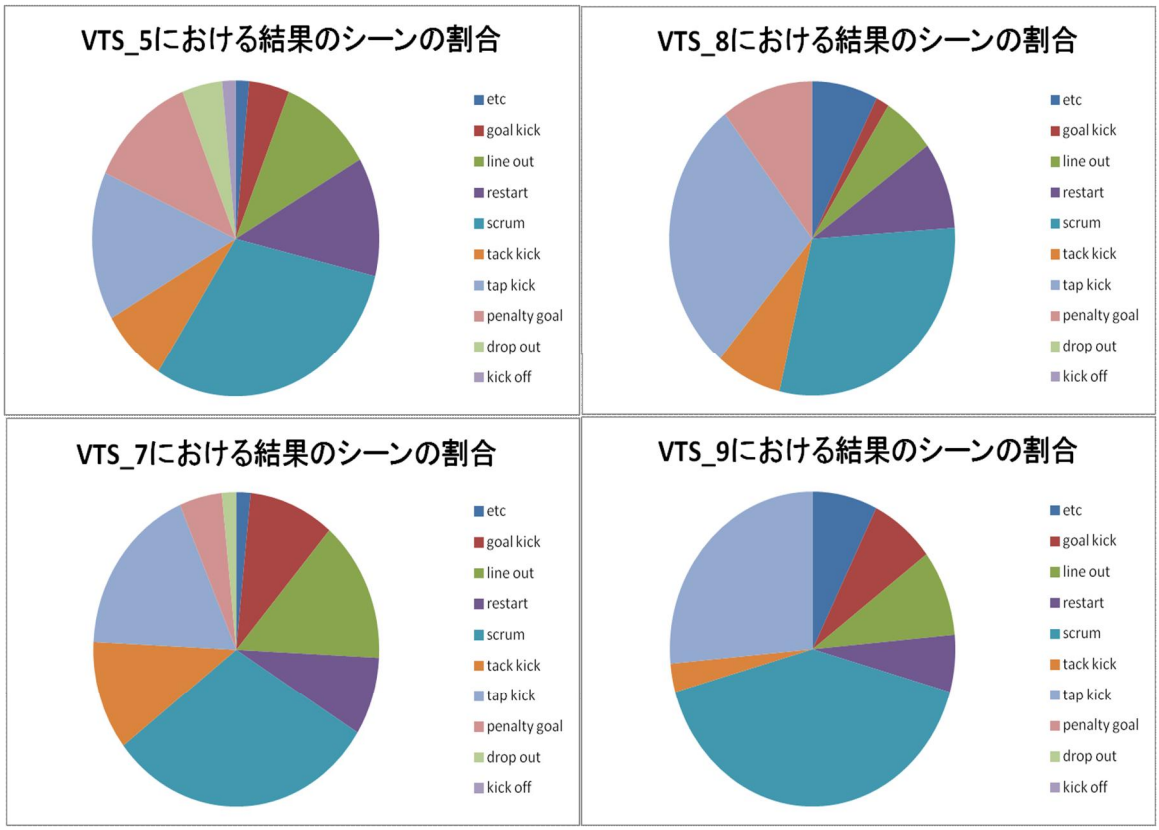


図 4.2 各試合の笛の結果のシーンの割合

図 4.3 は、試合毎の笛の継続時間のヒストグラムを示したものである。横軸は継続時間長、縦軸は各継続時間長の笛が吹かれる頻度である。横軸は、0~2.5 秒までの範囲を 20 分割している。レフェリー（試合）によって笛の継続時間長には差が見られる。また同じレフェリーでも笛の継続時間長にはばらつきがあることがわかる。VTS_5 の笛継続時間長の平均は 0.43 秒で標準偏差は 0.31 秒だった。VTS_7 の笛継続時間長の平均は 0.35 秒で標準偏差は 0.48 秒だった。VTS_8 の笛継続時間長の平均は 0.55 秒で標準偏差は 0.3 秒だった。VTS_9 の笛継続時間長の平均は 0.47 秒で標準偏差は 0.23 秒だった。

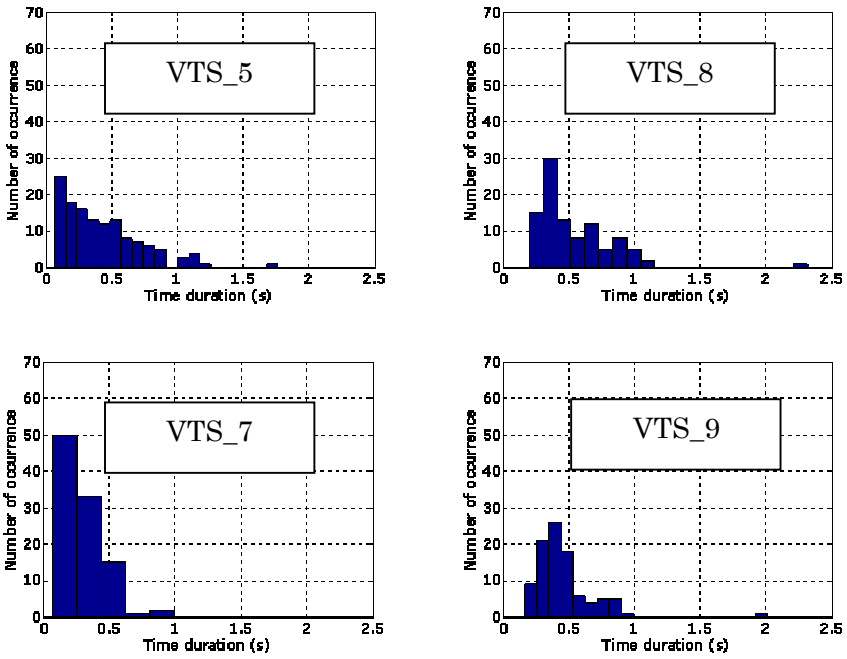


図 4.3 笛区間のヒストグラム

4.2 笛区間の検出率

本節では、笛検出に用いる音響特徴と検出率の関係、およびバンドパスフィルタの中心周波数（Fc(kHz)）、バンド幅（Fb(kHz)）、識別関数の傾きと切片(a と b)（閾値(Pth)、検出区間の削除と結合の最小時間（Ton(sec)と Toff(sec)）のパラメータと検出率の関係を示す。

4.2.1 笛の音響特徴と検出率の関係

表 4.2 に、バンドパスフィルタを音響特徴とし、検出性能が最も高くなるように各パラメータ条件を調整した時の検出率とパラメータ条件を示す。検出率は、試合によって異なるが 70.5~85.8%の範囲であり、ミス誤り率は 7.7~10.5%、付加誤り率は 6.4~21.4%の範囲にあることがわかる。試合の中では、VTS_5 において最も高い検出性能が得られている。また、各試合の音データに対する最適なパラメータ値に関しては、閾値 Pth とバンド幅 Fb は試合による差はあまりないが、バンドパスフィルタの中心周波数 fc は試合によって大きく異なることがわかる。表 4.3 は、バンドパスフィルタ出力と FM 成分を音響特徴とし、検出性能が最も高くなるように各パラメータ条件を調整した時の検出率とパラメータ条件を示す。表 4.2 に示した検出率と比較すると、FM 成分を音響特徴に加えることにより、4 試合中 3 試合において検出率が改善しているのがわかる。その改善率は最大で 3.8%であった。試合間で最適なパラメータ条件を比較すると、バンドパスフィルタの中心周波数は各試合によって検出率の良い条件が異なることがわかる。また、線形識別関数 y=ax+b の傾き a の値はどの試合でも同じ値になっているが、切片 b の値は試合によって異なっていることがわかる。

表 4.2 各試合の最も検出性能が高かったときの各パラメータ条件

試合	Toff	Ton	Pth	Fc	Fb	正解率(%)	ミス誤り率(%)	付加誤り率(%)
VTS_5	0.3	0.08	0.035	2.25	0.25	85.8	7.8	6.4
VTS_7	0.08	0.03	0.05	2.6	0.25	81.2	7.7	11.1
VTS_8	0.2	0.08	0.04	1.9	0.3	70.5	8.1	21.4
VTS_9	0.4	0.08	0.04	2.0	0.25	73.7	10.5	15.8

表 4.3 バンドパスフィルタ出力と FM 成分を用いた場合の笛検出性能

試合	Toff	Ton	b	a	Fc	Fb	正解率（%）	ミス誤り率(%)	付加誤り率(%)
VTS_5	0.7	0.08	-190	8	2.25	0.25	89.4	2.8	7.7
VTS_7	0.4	0.04	-120	8	2.25	0.25	85.0	7.1	7.1
VTS_8	0.2	0.08	-180	8	1.7	0.25	70.6	9.9	16.2
VTS_9	0.8	0.08	-140	8	1.7	0.25	72.8	20.4	6.7

4.2.2 バンドパスフィルタの中心周波数およびバンド幅と検出率の関係

表 4.4 は、試合 1(VTS_5)の音データを用い、バンドパスフィルタ出力を音響特徴とした場合の笛検出実験において、バンドパスフィルタの中心周波数 fc を 200Hz 間隔で変化させた場合の検出率の変化を表したものである。最も高い正解率は 81.2%であり、その時の中心周波数は 2.6 kHz であった。表 4.5 は、同様な笛検出実験において、バンドパスフィルタのバンド幅を 50Hz 間隔で変化させた場合の検出率を表している。最も検出率が高くなるのでバンド幅が 250Hz の時であり、250Hz を境にバンド幅を狭めるとミス誤りが増加し、付加誤りが減少する。逆にバンド幅を広くするとミス誤りは減少し、付加誤りが増加する。

表 4.4 バンドパスフィルタの中心周波数と笛検出率の関係

Toff	Ton	Pth	Fc	Fb	検出率（%）	ミス誤り（%）	付加誤り率（%）
0.08	0.03	0.05	2.2	0.25	54.0	38.1	8.0
0.08	0.03	0.05	2.4	0.25	80.5	11.5	8.0
0.08	0.03	0.05	2.6	0.25	81.2	7.7	11.1
0.08	0.03	0.05	2.8	0.25	30.8	58.1	11.1
0.08	0.03	0.05	3	0.25	0	86.7	13.3

表 4.5 バンドパスフィルタのバンド幅と笛検出率の関係

Toff	Ton	Pth	Fc	Fb(kHz)	正解率 (%)	ミス誤り率(%)	付加誤り率(%)
0.8	0.03	0.05	2.6	0.1	49.1	49.1	1.9
0.8	0.03	0.05	2.6	0.15	61.7	35.5	2.8
0.8	0.03	0.05	2.6	0.2	73.9	19.8	6.3
0.8	0.03	0.05	2.6	0.25	81.9	7.8	10.3
0.8	0.03	0.05	2.6	0.3	75.2	5.4	19.4
0.8	0.03	0.05	2.6	0.35	67.8	4.9	27.3

4.2.3 閾値（線形識別関数）と検出率の関係

表 4.6 は、閾値だけを変えたときの検出率の変化を表している。最も高い検出率は閾値が 0.05 の時である。閾値が 0.05 をより小さく設定すると、ミス誤りが減少するが付加誤りが増加する。逆に、閾値が 0.05 より大きく設定すると、ミスが増えるが付加誤りは減少していくのがわかる。

表 4.6 閾値と笛検出率の関係

Toff	Ton	Pth	Fc	Fb	正解率(%)	ミス誤り率(%)	付加誤り率 (%)
0.08	0.03	0.02	2.6	0.25	6.3	0.1	93.5
0.08	0.03	0.03	2.6	0.25	23.6	0.9	75.4
0.08	0.03	0.04	2.6	0.25	58.1	4.2	37.7
0.08	0.03	0.05	2.6	0.25	81.2	7.7	11.1
0.08	0.03	0.06	2.6	0.25	70.1	27.1	2.8
0.08	0.03	0.07	2.6	0.25	44.2	55.8	0
0.08	0.03	0.08	2.6	0.25	21.1	78.8	0

4.2.4 検出区間のギャップ時間長と検出率の関係

表 4.7 は、検出区間のギャップ時間長と検出率の関係を示している。断続的な笛が吹かれた場合にその笛を結合させる処理を行うが、その断続的な笛の間隔がどのくらいまで許容できるか基準を設ける必要がある。この笛の間隔をギャップ時間長と定義する。たとえば笛区間のギャップ時間長が 0.01 秒の場合は、断続的な笛の間隔が 0.01 秒以内の場合は笛検出が可能という意味である。表 4.7 の場合、0.04 秒を境に検出率が変わっている。

表 4.7 検出区間ギャップ時間長と笛検出率の関係

Toff	Ton	Pth	Fc	Fb	正解率(%)	ミス誤り率(%)	付加誤り率(%)
0.01	0.03	0.05	2.6	0.25	79.8	7.6	12.6
0.02	0.03	0.05	2.6	0.25	79.8	7.6	12.6
0.03	0.03	0.05	2.6	0.25	79.8	7.6	12.6
0.04	0.03	0.05	2.6	0.25	79.8	7.6	12.6
0.05	0.03	0.05	2.6	0.25	81.2	7.7	11.1
0.06	0.03	0.05	2.6	0.25	81.2	7.7	11.1
0.07	0.03	0.05	2.6	0.25	81.2	7.7	11.1
0.08	0.03	0.05	2.6	0.25	81.2	7.7	11.1
0.09	0.03	0.05	2.6	0.25	81.2	7.7	11.1

4.2.5 検出区間長と検出率の関係

表 4.8 は、笛の検出区間長を 0.01 秒から 0.09 秒まで変えていったときの検出率の推移を表している。0.04 秒までは検出率は変わらないが、それ以上の値を設定した場合は正解率が下がり、ミス誤り率は増加し、付加誤りは減少している。

表 4.8 検出区間ギャップ時間長と笛検出率の関係

Toff	Ton	Pth	Fc	Fb	正解率(%)	ミス誤り率(%)	付加誤り率(%)
0.08	0.01	0.05	2.6	0.25	81.2	7.7	11.1
0.08	0.02	0.05	2.6	0.25	81.2	7.7	11.1
0.08	0.03	0.05	2.6	0.25	81.2	7.7	11.1
0.08	0.04	0.05	2.6	0.25	81.2	7.7	11.1
0.08	0.05	0.05	2.6	0.25	62.6	34.6	2.8
0.08	0.06	0.05	2.6	0.25	62.6	34.6	2.8
0.08	0.07	0.05	2.6	0.25	62.6	34.6	2.8
0.08	0.08	0.05	2.6	0.25	55.7	42.5	1.9
0.08	0.09	0.05	2.6	0.25	55.7	42.5	1.9

4.3 映像データベースの自動作成

検出された笛の区間（開始時点と終了時点）のデータをもとに ELAN スクリプトを自動生成した。図 4.4 は、自動生成された ELAN スクリプトによる映像データベースの一例を示す。図に示すように、笛の検出結果が ELAN のデータベースに反映されていることがわかる。それぞれ正解のタグは c（correct）ミス誤りは m(Miss error)、付加誤りは a(Add error)でとして付与されている。このように ELAN の映像データベースを自動作成することができる。左の図は c38 とついているこれは正解と検出区間が合致した 38 番目のシーンを出している。真ん中の図は m4 で正解区間がある場所に検出がされていないシーンの 4 番目のシーンである。右の図は a3 で正解区間がない場所に検出がされているシーンが 3 番目とい

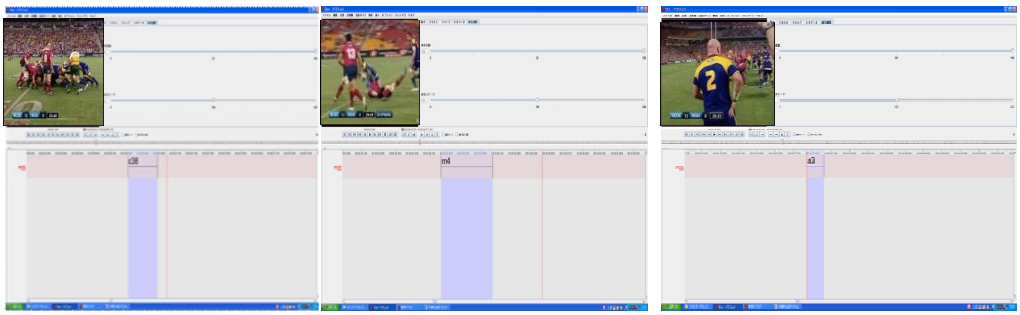


図 4.4 ELAN による映像データベース自動作成

うことを表している。本来、自動データベースを実用するなら正解区間があらかじめわかっていることがないので、正解やミス誤りや付加誤りというタグは存在せず検出結果をタグに反映させるのだが、現段階では実用ではなく研究段階なのでこのようなタグがついている。このデータベースの作成は以下の①～③の処理を MATLAB プログラムにより実行することで行う。

① 検出区間のデータファイル、タグ付け対象の映像ファイルの指定
② 作成する ELAN ファイル名の指定
③ ①で指定したファイルの情報を反映して ELAN スクリプトを生成

5 考察

本研究で用いられる笛検出の性能は、様々なパラメータの設定によって変化する。ここでは、各パラメータが検出率に与える影響について考察する。

5.1 バンドパスフィルタの中心周波数が検出率に与える影響

バンドパスフィルタ出力を用いた場合の笛検出性能を見たところ、試合によって検出率が異なることがわかった。この原因として中心周波数が低いと笛音以外の音がバンドパスフィルタ出力に入り込む可能性が高くなる。したがって VTS_8 や VTS_9 のように付加誤り率が増加し、正解率が低くなる。

また表 4.4 や 4.5 の結果では笛音のピーク周波数の中心周波数にバンドパスフィルタの中心周波数を設定することが望ましいことを示していた。バンドパスフィルタの中心周波数を笛音のピーク周波数以上や以下にすると、笛音の特徴である周波数成分が検出できなくなるために笛の検出ミスが急激に増大し、検出率が低下する。

このようにバンドパスフィルタ出力の特性上設定した周波数に笛の音信号が入ってこないと検出率に影響を与える。そしてその中心周波数に大量の笛以外の音信号が入ると検出率に影響を与えるということである。つまりバンドパスフィルタ出力を用いて笛検出をする際に中心周波数の設定は重要な要素であることがわかった。

5.2 バンド幅が検出率に与える影響

表 4.5 で見たように最も検出率が高くなるバンド幅が 250Hz の時だった。そしてこの 250Hz を境にバンド幅を狭めるとミス誤りが増加し、付加誤りが減少する。逆にバンド幅を広くするとミス誤りは減少し、付加誤りが増加する。これはバンド幅を狭めると抽出される音信号の周波数成分がより限定されるため、笛の音信号がこのバンド幅より外に存在する場合に笛を検出できなくなるためにミス誤りが増加するが、笛音でない音信号も遮断できるので付加誤りは減少する。逆にバンド幅を広くすると狭めた場合と逆に、笛信号は検出しやすくなるが、笛以外のアナウンサー音声や歓声を笛と間違って検出してしまうため、付加誤りが増大することになる。

5.3 閾値が検出率に与える影響

表 4.6 のとおり、この試合の場合、最も高い正解率は閾値が 0.05 の時である。閾値が 0.05 をより小さく設定すると、ミス誤りが減少するが付加誤りが増加する。逆に、閾値が 0.05 より大きく設定すると、ミスが増えるが付加誤りは減少していくのがわかる。閾値を下げるということは笛判定の基準を甘くすることを意味する。したがって基準を甘くすると検出しにくかった音信号を笛音として検出することに成功するが、笛ではない音信号を笛音として誤認識してしまう。逆に、閾値を上げて笛判定の基準を厳しくすると、笛音を笛で

はないと誤認識し、笛ではない音信号は検出されなくなる。このようなことが検出結果に反映されている。

5.4 笛検出区間長が検出率に与える影響

表 4.8 を見るとこの試合の場合、0.05 を境に正解率、ミス誤り、付加誤りの値が違っているのがわかる。つまり継続長を短くするということはバンドパスフィルタ出力が閾値を超えた場合に笛音以外の音信号でも笛検出される可能性が高くなる。したがって 0.05 秒以下のパラメータ設定の場合、笛音として検出されにくい音信号を拾うことができるためミス誤りは減少するが、逆に笛音ではない音信号を笛検出してしまうので付加誤りが増えてしまう。一方 0.05 秒以上のパラメータ設定の場合は、0.05 秒以下に比べミス誤りが増加しているが、付加誤りがそれ以上に減少しているため、正解率に差が出ている。

5.5 検出誤りに関する分析

図 5.1 のようにミス誤りを引き起こしている笛音は多くの場合非常に笛の継続長が短い

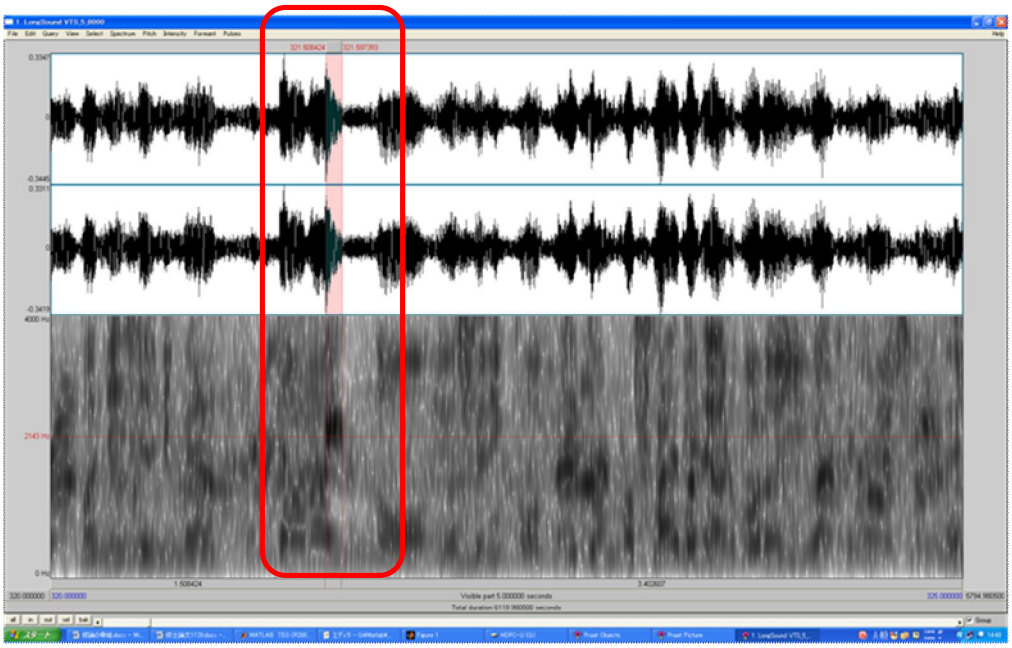


図 5.1 ミス誤りを起こす笛の音声波形の例

ことがわかった。

この笛の継続長は約 0.05 秒である。VTS_5 のミス誤り 5 か所中 4 か所は 0.06 秒以下であり、1 秒も笛が吹かれていない。プログラム上、継続長も笛検出の基準にしているので、短い笛音は笛と認識してくれないと考えられる。しかしながら前述したように、笛の最小継続長を短く設定すればこれらのミス誤りは解消されるが、逆に付加誤りが増えてしまう。

したがって、継続長の閾値を 0.04 秒などにするとこれらの短い笛音は検出されるが、他のアナウンサーの声などが笛として検出されて結局全体の検出率は落ちてしまう。

5.6 検出性能と処理量の関係

本研究では笛特徴抽出の方法として

①バンドパスフィルタを用いた笛の周波数分析を行う方法

②バンドパスフィルタを用いた笛の周波数分析＋FM 成分の分析を行う方法

の 2 つを用いた。

これらの特徴量の選択を考える場合、検出率を高めることが一義的ではあるが、処理量の問題を考慮する必要がある。表 4.3 に示したように、2 種類の特徴を用いる②の方が検出率は良い。しかし検出処理速度が数百倍変わってくる。バンドパスフィルタ出力のみを用いる①の方法では、試合にもよるが 1 試合全体の検出処理が約 2 分弱で完了することができるのに対し、②の方法では 4 時間～5 時間かかる。検出率と処理速度の双方を考えると、①の方が有用性はあるように思われる。

6 結論

本研究のねらいは映像データベースの作成におけるタグ付けを、コンピュータの情報処理技術を用いて自動化する手法を開発することであり、そのためにラグビーを対象としてレフェリーの笛検出を取り上げた。笛検出実験を行った結果として

①バンドパスフィルタを用いた笛の周波数分析を行う方法でもっとも良い正解率は 85%で笛のシーン数 132 シーン中 112 シーンが検出できた。

④ ンドパスフィルタを用いた笛の周波数分析＋ FM 成分の分析を行う方法で最も良い正解率は 89%で、笛のシーン数 132 シーン中 117 シーンが検出できた。

の 2 つがわかった。ただし試合によって検出率には開きがあり、また最適なパラメータ条件も試合によって異なっていた。

本研究では笛検出率を 100%にすることはできなかった。その理由は継続長が短い笛を検出しようと試みると他の音を誤検出してしまうからである。また試合によって最適なパラメータ条件が異なるという点である。どの試合でもパラメータ条件を固定し、ある一定の検出率を出すような工夫が今後必要である。したがってまずソフトウェアからの改善が考えられる。例えばこの笛検出と画像処理を併用する方法などが考えられる。たとえば笛検出のパラメータ条件をあまく設定し、その箇所について画像処理をするという方法である。このように併用することで画像処理の問題点である処理時間をカバーすることができる。つまり音情報で目当てのところを素早く検出し、その箇所により質の高い画像検出をかけるということである。

もうひとつはハード機器の改善が考えられる。たとえば撮影の際に笛だけを拾うことができる集音声の高い機器を用いるなどである。これにより笛音とそれ以外の音のコントラストが鮮明になるため継続長の処理をプログラムに組み込む必要が無くなる可能性がある。さらに継続長の問題を解決することができる。このように笛検出率を 100%にするためにはソフトウェアとハード機器の両面からの改善が求められるだろう。

最後に、スポーツコーチングに利用する視点から見た今後の課題について述べる。データベースシステムをスポーツコーチングに活用するためには手順を簡素化する必要がある。現段階の手順は、放送映像を録画→音信号の抽出→MATLAB による検出→ELAN へ検出結果の書き込み、のようになり複雑である。特に MATLAB と ELAN の互換性に問題がある。たとえば文字コードなどのトラブルで文字変換がうまくできないこともある。この手順を何らかの形でスムーズにする必要がある。この手順の簡素化により、リアルタイムでの自動ゲーム分析も可能になるだろう。

しかし現段階で本研究のデータベースシステムをスポーツコーチングに実用するなら次のようなビジョンが考えられる。まずコンピュータに笛音をすばやく取りこみ、より詳細な分析は視察によって行い、指導に活かすというスタイルである。すなわち、すばやく検出

できるコンピュータの利点とプレイの専門知識を生かしてプレイを細かく評価できる人による利点を融合して効果的な指導に生かすという形である。

謝辞

本論文作成にあたり、ご指導いただきました誉田先生には心よりお礼申し上げます。またお忙しい中、快く副査を引きうけてくださった吉永先生、宝田先生には感謝しています。また誉田研究室の方々にも多くの面でご助力いただき本当にありがとうございました。最後になりましたが、本論文作成に関わってくださった方々に厚くお礼申しあげます。

文献目録

1. 日本経済新聞. iPad と動画が変えた戦術 女子バレー飛躍の舞台裏. (オンライン) 2011 年 8 月 16 日 . http://www.nikkei.com/news/headline/article/g=96958A9C93819499E2EAE2E19E8DE2EAE2EAE0E2E3E3E2E2E2E2E2E2.

2. Sports Code. (オンライン) (引用日: 2011 年 12 月 5 日.) http://www.fitnessapollo.jp/new/sportstec/trak.html.

3. HellwigBirgit. ELAN Language Archiving Technology. (オンライン) (引用日: 2012 年 1 月 11 日.) http://www.lat-mpi.eu/tools/elan.

4. 國枝　孝之. MPEG-7 と映像検索. ：CQ 出版社, 2004.

5. 池原　雅章、奥田正浩、長井隆行. だれでもわかる MATLAB 即戦力ツールブック.: 培風館, 2006.

6. HonXuedong Huang、Alex Acero、Hsiad－wuen. SPOKEN LANGUAGE PROCCESSING A Guide to Theory,Algorithm,and System Development. : Carnegie Mellon University, 2001.

7. Alan V. OppenheimW. Schafer, John R. BuckRonald. DISCRETE-TIME SIGNAL PROCESSING. United States of America : PRENTICE HALL INTERNATIONAL,INC, 1998.

8. ディジタル信号処理. 萩原　将文. ：森北出版株式会社, 2001 年.

9. 音声のディジタル信号処理(上)、(下). 鈴木　久喜. ：コロナ社, 1983 年.

10. レイ・D・ケント、チャールズ・リード. 音声の音響分析. ：海文堂, 1996.

11. 新版音響用語辞典. 日本音響学会. ：コロナ社, 1988 年.

12. 宮崎　友孝. 電子機器の動作制御装置. 2008 年 1 月 17 日.

13. 小林　一行. MATLAB ハンドブック. ：秀和システム, 2004.

14. 田中　和紀. 音声認識による映像検索 Web システム「Dumpling」の構築.

付録

①バンドパスフィルタを用いた笛の周波数分析を行うプログラム（MATLAB）

```
clc
clear all
close all

%グラフ表示モード
display=1;
%音のサンプリング周波数
fs=8000;
%fs=16000;
%フィルタ次チャネル数
nch=1;
%nch=2;
%フィルタ中心周波数
f1=2250;
f2=4800;
%フィルタバンド幅
df=250;
%フィルタ次数
ntap=64;
%ブロック時間長(秒)
nsec=60;
%笛区間を結合する時間長（秒）
dth=0.3;
%笛区間の最小時間長（秒）
dth_pow=0.04;
%パワー計算の窓長（秒）
nwsec=0.03;

%音ファイルの入力
[filename,pathname]=uigetfile('./*.wav','Open WAV File...');
siz=wavread([pathname,filename],'size')

%ブロックサンプル数
div=fs*nsec;
%ブロックのトータル個数
total_blk=fix(siz(1)/div);

% x  正解データを読み込む
[filename2,pathname2]=uigetfile('./*.txt','Open TXT File...');
[buf]=textread([pathname2,filename2],'%s');
nsei=length(buf)/4;
seikai(:,1)=str2num(char(buf(2:4:nsei*4)))/1000;
seikai(:,2)=str2num(char(buf(3:4:nsei*4)))/1000
wsl=char(buf(4:4:nsei*4));;

%分析条件のループ
ncon=0;
for dth=[0.3]
for dth_pow=[0.08]
for pth=[0.035]
for f1=[2250]
for f2=[4800]
for df=[250]
ncon=ncon+1;

%フィルタの設計
fc=[f1,f2];
for i=1:nch
```

```
flow=(fc(i)-df)/(fs/2);
fhigh=(fc(i)+df)/(fs/2);
b(:,i)=fir1(ntap,[flow fhigh],'bandpass');
end
a(1)=1;

ndet=0;
%音ブロックのループ
for iblk=1:total_blk
start=(iblk-1)*div+1;
last=iblk*div;


%音データの読み込み
[input,fs]=wavread([pathname,filename],[start last]);
%wavplay(input,fs)

%フィルタリング処理
output=zeros(length(input(:,1)),1);
for i=1:nch
output=output+filtfilt(b(:,i),a,input(:,1));
end

%パワー計算
nw=nwsec*fs;
npow=0;
for i=1:nw:length(input(:,1))
npow=npow+1;
pow_input(npow)=sqrt(dot(input(i:i+nw-1,1),input(i:i+nw-1,1))/nw);
pow_output(npow)=sqrt(dot(output(i:i+nw-1),output(i:i+nw-1))/nw);
end
%パワーレベルと継続時間による検出
%パワーの二値化
pow_logical=logical(pow_output>pth);

%継続時間が短い区間を削除
pow_logical2=pow_logical;
dth_on=round(dth_pow/nwsec);
for i=1:npow
if pow_logical2(i)==1
on=on+1;
elseif pow_logical2(i)==0 & on>=1 & on<dth_on
pow_logical2(i-on:i-1)=0;
on=0;
else
on=0;
end
end

%継続時間が短い検出区間の結合
pow_logical3=pow_logical2;
dth_off=round(dth/nwsec);
off=0;
for i=1:npow
if pow_logical3(i)==0
off=off+1;
elseif pow_logical3(i)==1 & off>=1 & off <dth_off
pow_logical3(i-off:i-1)=1;
off=0;
else
off=0;
end
end
 %検出時点の算出
check=0;
for i=1:npow-1
if (pow_logical3(i)==0 && pow_logical3(i+1)==1) | (i==1 & pow_logical3(i)==1)
check=1;
istart=i;
elseif check==1 & pow_logical3(i)==1 && pow_logical3(i+1)==0
ndet=ndet+1;
```

```
detect(ndet,1)=istart*nwsec+(iblk-1)*nsec;
detect(ndet,2)=i*nwsec+(iblk-1)*nsec;
check=0;
elseif check==1 & i==npow-1 & pow_logical3(npow)==1
ndet=ndet+1;
detect(ndet,1)=istart*nwsec+(iblk-1)*nsec;
detect(ndet,2)=npow*nwsec+(iblk-1)*nsec;
check=0;
end
end
 %波形表示（入力信号パワー、フィルタ出力パワー、検出区間、正解区間）
if display==1
time=[0:npow-1]*nwsec+(iblk-1)*nsec;
plot(time,pow_input)
axis([(iblk-1)*nsec,iblk*nsec, 0,0.2])

seikai_kukan(1:div)=0;
for i=1:nsei
if seikai(i,1)>=(iblk-1)*nsec && seikai(i,1)<=iblk*nsec
for j=1:div
t=j/div*nsec+(iblk-1)*nsec;
if t>=seikai(i,1) && t<=seikai(i,2)
seikai_kukan(j)=0.5;
end
end
end
pause
end
end
 %検出率の計算
clear index_detect
index_detect(1:ndet)=0;
index_seikai(1:nsei)=0;
for i=1:nsei
for j=1:ndet
if (seikai(i,1)> detect(j,1) & seikai(i,1) > detect(j,2)) | ...
 (seikai(i,2)< detect(j,1) & seikai(i,2) < detect(j,2))
else
index_detect(j)=1;
index_seikai(i)=1;
end
end
end
nseikai_detect=sum(index_detect);
nseikai_seikai=sum(index_seikai);
nerror_miss=nsei-nseikai_seikai;
nerror_add=ndet-nseikai_detect;
 ntotal=nseikai_seikai+nerror_miss+nerror_add;
seikai_rate(ncon)=nseikai_seikai/ntotal*100;
error_miss_rate(ncon)=nerror_miss/ntotal*100;
error_add_rate(ncon)=nerror_add/ntotal*100;seikai_hyo(ncon,1)=dth;
seikai_hyo(ncon,2)=dth_pow;
seikai_hyo(ncon,3)=pth;
seikai_hyo(ncon,4)=f1/1000;
seikai_hyo(ncon,5)=f2/1000;
seikai_hyo(ncon,6)=df/1000;
seikai_hyo(ncon,7)=seikai_rate(ncon);
seikai_hyo(ncon,8)=error_miss_rate(ncon);
seikai_hyo(ncon,9)=error_add_rate(ncon);
seikai_hyo(ncon,1:9)

%ミス誤りと付加誤りのインデックス
j=0;
for i=1:nsei
if index_seikai(i)==0
j=j+1;
error_miss(ncon,j)=seikai(i,1);
end
end
j=0;
for i=1:ndet
if index_detect(i)==0
j=j+1;
```

```
error_add(ncon,j)=detect(i,1);
end
end
end
end
end
end
end
end
seikai_hyo
save 'tone_detect_resultVTS_2' seikai_hyo error_miss error_add
```

②バンドパスフィルタを用いた笛の周波数分析 ＋ FM成分の分析を行うプログラム

```
%短いギャップを埋めてから、短い区間を削除
clc
clear all
close all

%グラフ表示モード
display=0;
miss_sample=[0.3215    0.6566    3.4140    3.6841    5.2479]*1000;
add_sample=[1.4595    2.0778    2.9931    2.9951    3.7873    4.4033    4.4160    5.1947 5.5817
5.6421]*1000;
%音のサンプリング周波数
fs=8000;
%fs=16000;
%フィルタ次チャネル数
nch=1;
%バンドパスフィルタ中心周波数
f1=2250;
f2=4800;
%バンドパスフィルタバンド幅
df=250;
%フィルタ次数
ntap=64;
%ブロック時間長(秒)
nsec=60;
%笛区間を結合する時間長（秒）
dth=0.3;
%笛区間の最小時間長（秒）
dth_pow=0.04;
%パワー計算の窓長（秒）
nwsec=0.03;

%音ファイルの入力
[filename,pathname]=uigetfile('./*.wav','Open WAV File...');
siz=wavread([pathname,filename],'size');

%ブロックサンプル数
div=fs*nsec;
%ブロックのトータル個数
total_blk=fix(siz(1)/div);

% x  正解データを読み込む
[filename2,pathname2]=uigetfile('./*.txt','Open TXT File...');
[buf]=textread([pathname2,filename2],'%s');
nsei=length(buf)/4;
seikai(:,1)=str2num(char(buf(2:4:nsei*4)))/1000;
```

```
seikai(:,2)=str2num(char(buf(3:4:nsei*4)))/1000;
wsl=char(buf(4:4:nsei*4));

%分析データの読み込み
[filename3,pathname2]=uigetfile('./*.mat','分析データ...');
data=load([pathname2,filename3]);
pow_data=data.pow_output;
fcpeak_data=data.fcpeak;
fcpeak_max=max(fcpeak_data);
npow=data.nfcpeak;

%分析条件のループ
ncon=0;
for dth=[0.2:0.2:1.0]
for dth_pow=[0.04 0.06 0.08]
for powth=[0.02]
for pth1=[-190]
for pth2=[8]
for f1=[2250]
for f2=[4800]
for df=[250]
for ishift=[1]
ncon=ncon+1;
ndet=0;

%音ブロックのループ
for iblk=1:total_blk
pow_output=pow_data((iblk-1)*npow+1:iblk*npow);
%FM変調成分の分析(pow_outputがpowthより大きい場合のみ分析)
if max(pow_output)>powth
fcpeak=fcpeak_data((iblk-1)*npow+1:iblk*npow);
if ishift>0
fcpeak(1:npow-ishift)=fcpeak(ishift+1:npow);
fcpeak(npow-ishift+1:npow)=0;
elseif ishift<0
fcpeak(-ishift+1:npow)=fcpeak(1:npow+ishift);
fcpeak(1:-ishift)=0;
end
%パワーレベルと継続時間による検出
%パワーの二値化
pow_logical=logical(pth1*pow_output+pth2<=fcpeak);
else
pow_logical=logical(pow_output>powth);
end
%継続時間が短い検出区間の結合
pow_logical2=pow_logical;
dth_off=round(dth/nwsec);
off=0;
for i=1:npow
```

```
if pow_logical2(i)==0
off=off+1;
elseif pow_logical2(i)==1 & off>=1 & off <dth_off
pow_logical2(i-off:i-1)=1;
off=0;
else
off=0;
end
end
%継続時間が短い区間を削除
pow_logical3=pow_logical2;
dth_on=round(dth_pow/nwsec);
on=0;
for i=1:npow
if pow_logical3(i)==1
on=on+1;
elseif pow_logical3(i)==0 & on>=1 & on<dth_on
pow_logical3(i-on:i-1)=0;
on=0;
else
on=0;
end
end
```

```
%検出時点の算出
check=0;
for i=1:npow-1
if (pow_logical3(i)==0 && pow_logical3(i+1)==1) | (i==1 & pow_logical3(i)==1)
check=1;
istart=i;
elseif check==1 & pow_logical3(i)==1 && pow_logical3(i+1)==0
ndet=ndet+1;
detect(ndet,1)=istart*nwsec+(iblk-1)*nsec;
detect(ndet,2)=i*nwsec+(iblk-1)*nsec;
check=0;
elseif check==1 & i==npow-1 & pow_logical3(npow)==1
ndet=ndet+1;
detect(ndet,1)=istart*nwsec+(iblk-1)*nsec;
detect(ndet,2)=npow*nwsec+(iblk-1)*nsec;
check=0;
end
end

%波形表示（入力信号パワー、フィルタ出力パワー、検出区間、正解区間）
if display~=0
check=0;
if length(miss_sample)~=0
for i=1:length(miss_sample)
```

```
if miss_sample(i)>=(iblk-1)*nsec & miss_sample(i)<=iblk*nsec
check=1;
end
end
end
if length(add_sample)~=0
for i=1:length(add_sample)
if add_sample(i)>=(iblk-1)*nsec & add_sample(i)<=iblk*nsec
check=1;
end
end
end
if display==1 | check==1
subplot(5,1,1)
time=[0:npow-1]*nwsec+(iblk-1)*nsec;
plot(time,pow_output)
axis([(iblk-1)*nsec,iblk*nsec, 0,0.2])
subplot(5,1,2)
plot(time,fcpeak)
axis([(iblk-1)*nsec,iblk*nsec, 0,fcpeak_max])
subplot(5,1,3)
plot(time,pow_logical)
axis([(iblk-1)*nsec,iblk*nsec,0,1.2])
hold on
subplot(5,1,4)
plot(time,pow_logical2)
axis([(iblk-1)*nsec,iblk*nsec,0,1.2])
hold on
subplot(5,1,5)
plot(time,pow_logical3)
axis([(iblk-1)*nsec,iblk*nsec,0,1.2])
hold on

seikai_kukan(1:div)=0;
for i=1:nsei
if seikai(i,1)>=(iblk-1)*nsec && seikai(i,1)<=iblk*nsec
for j=1:div
t=j/div*nsec+(iblk-1)*nsec;
if t>=seikai(i,1) && t<=seikai(i,2)
seikai_kukan(j)=0.5;
end
end
end
end
for i=3:5
subplot(5,1,i)
plot([0:div-1]/div*nsec+(iblk-1)*nsec,seikai_kukan,'r')
axis([(iblk-1)*nsec,iblk*nsec,0,1.2])
```

```
hold off
end
pause
end
end
end

%分析ブロックにまたがる区間の統合
ndetbuf=ndet;
for i=1:ndet-1
if detect(i,2)+nwsec==detect(i+1,1)
detect(i,2)=detect(i+1,2);
detect(i+1:ndetbuf-1,:)=detect(i+2:ndetbuf,:);
ndetbuf=ndetbuf-1;
end
end
ndet=ndetbuf;

%検出率の計算
clear index_detect
index_detect(1:ndet)=0;
index_seikai(1:nsei)=0;
for i=1:nsei
for j=1:ndet
if (seikai(i,1)> detect(j,1) & seikai(i,1) > detect(j,2)) | ...
(seikai(i,2)< detect(j,1) & seikai(i,2) < detect(j,2))
else
index_detect(j)=1;
index_seikai(i)=1;
end
end
end
nseikai_detect=sum(index_detect);
nseikai_seikai=sum(index_seikai);
nerror_miss=nsei-nseikai_seikai;
nerror_add=ndet-nseikai_detect;
ntotal=nseikai_detect+nerror_miss+nerror_add;
seikai_rate(ncon)=nseikai_seikai/ntotal*100;
error_miss_rate(ncon)=nerror_miss/ntotal*100;
error_add_rate(ncon)=nerror_add/ntotal*100;
seikai_hyo(ncon,1)=dth;
seikai_hyo(ncon,2)=dth_pow;
seikai_hyo(ncon,3)=pth1;
seikai_hyo(ncon,4)=pth2;
seikai_hyo(ncon,5)=f1/1000;
seikai_hyo(ncon,6)=f2/1000;
seikai_hyo(ncon,7)=df/1000;
seikai_hyo(ncon,8)=powth;
```

```
seikai_hyo(ncon,9)=seikai_rate(ncon);
seikai_hyo(ncon,10)=error_miss_rate(ncon);
seikai_hyo(ncon,11)=error_add_rate(ncon);
seikai_hyo(ncon,1:11)
%ミス誤りと付加誤りのインデックス
j=0;
for i=1:nsei
if index_seikai(i)==0
j=j+1;
error_miss(ncon,j)=seikai(i,1);
end
end
j=0;
for i=1:ndet
if index_detect(i)==0
j=j+1;
error_add(ncon,j)=detect(i,1);
end
end
end
end
end
end
end
end
end
end
seikai_hyosave 'tone_detect_resultVTS_5' seikai_hyo error_miss error_add
```