情報処理学会論文誌 Vol.60 No.2 397–410 (Feb. 2019)

Picognizer：電子音の認識のための JavaScript ライブラリの開発と評価

栗原 一貴[1,a)] 植村 あい子[2] 板谷 あかり[1] 北原 鉄朗[2] 長尾 確[3]

受付日 2018年4月20日, 採録日 2018年11月7日

概要：我々は今や様々な電子音に囲まれて生活しており，それらを検出・認識し情報処理を行うことは Maker 文化の発展，ディジタルゲーム拡張およびゲーミフィケーションの構成のうえで有意義であるが，エンドユーザプログラマが個々の検出・認識対象について教師付き学習により認識器を構築するのは現状ではまだ容易ではない．そこで再生ごとの音響的変動が小さいという電子音の特徴を活かし，Dynamic Time Warping などの伝統的なテンプレートベースのパターンマッチングアルゴリズムにより電子音の検出・認識を行う JavaScript ライブラリを実装する．基礎的な性能評価を行うとともに，様々なユースケースを例示することでその有用性を示す．

キーワード：音響イベント検出，テンプレートマッチング，ディジタルゲーム，ゲーミフィケーション

Picognizer: Development and Evaluation on a JavaScript Library for Detecting and Recognizing Synthesized Sounds

Kazutaka Kurihara[1,a)] Aiko Uemura[2] Akari Itaya[1] Tetsuro Kitahara[2] Katashi Nagao[3]

Received: April 20, 2018, Accepted: November 7, 2018

Abstract: In this paper, we describe and evaluate Picognizer, a JavaScript library that detects and recognizes user-specified synthesized sounds using a template-matching approach. In their daily lives, people are surrounded by various synthesized sounds, so it is valuable to establish a way to recognize such sounds as triggers for invoking information systems. However, it is not easy to enable end-user programmers to create custom-built recognizers for each usage scenario through supervised learning. Thus, by focusing on a feature of synthesized sounds whose auditory deviation is small for each re-play, we implemented a JavaScript library that detects and recognizes sounds using traditional pattern-matching algorithms. We evaluated its performance quantitatively and show its effectiveness by proposing various usage scenarios such as an autoplay system of digital games, and the augmentation of digital games including a gamification.

Keywords: audio event detection and recognition, template matching, digital games, gamification

1. はじめに

我々はディジタルゲームの効果音をはじめとして，日々，交通誘導，マナー啓発，広告手段，家電や情報機器の状態通知などの様々な電子音に囲まれて生活している．ここで電子音とは計算機などの機械によって再生された，毎回の音響的変動の小さい音と定義する．人間の音声を録音し再生した電子音声も，電子音の一種である．これらを人間へのシグナルとしてだけではなく，電子音検出・認識を通じて計算機システムへの入力として再利用する手段を提供することで，まだ IoT 化されていない生活環境の要素を IoT の世界に接続することが可能になり，そこを起点として，生活を豊かにする様々なアイディアが実現できるであろう．これは HCI・EC 研究コミュニティ，および近年さか

[1] 津田塾大学
Tsuda College, Kodaira, Tokyo 187–8577, Japan
[2] 日本大学
Nihon University, Setagaya, Tokyo 156–8550, Japan
[3] 名古屋大学大学院情報学研究科
Graduate School of Informatics, Nagoya University, Nagoya, Aichi 464–8601, Japan
[a)] kurihara@tsuda.ac.jp

んになっている Maker 文化の発展に貢献が期待できる研究領域である．

本研究では，電子音の中でディジタルゲームの効果音の検出・認識に注目する．効果音の認識を起点にしたディジタルゲーム拡張は，既存ディジタルゲームに新たなエンタテインメント価値を付与したり，非ゲーム的目的，たとえば社会善の達成を実現したりするような拡張的システム開発を容易にすることができる．後者は栗原 [1] がこれまでに提唱した，ゲーミフィケーションの周辺概念である Toolification of Games を指す．既存ディジタルゲームを再利用した拡張的システム開発は，これまで知的財産権の問題，およびソースコードの入手経路の問題から実現が難しかった．電子音検出・認識を用いることにより，既存ゲームを改変することなく，外付けする形でのゲームの内部状態の取得とそれに基づく情報処理が可能になる．

音の認識を構成する基礎技術として，近年，パターン認識技術は飛躍的な進歩をとげている．その中でも，深層学習を用いた教師付き機械学習手法は特に目覚ましい成果をあげている [2] が，大規模なデータ収集と正解ラベル付け作業，および膨大な計算資源に基づいた学習が必要になる．画像認識や音声認識などをエンドユーザプログラマが活用しようと思った場合は，そのような資源を持つ企業・研究機関などが構築した汎用認識器や API を活用することが多いのが現状である．

一方，ゲーム内の効果音をはじめとする電子音に対象を絞れば，これらを検出するうえで高度なパターン認識技術は必ずしも必要ない．再生される音源は基本的に毎回同じであり，音響的特徴の変動が十分に近いと期待できるため，テンプレートベースの古典的な手法でも実用上十分な精度が得られると考えられる．しかし，こういった古典的なテンプレートマッチングによる電子音の検出を Web 上で簡単に実現するためのライブラリは存在しなかった．

本研究では，そのような用途で活用が期待できる，電子音の検出・認識を手軽に実現する JavaScript ライブラリ，Picognizer を提案する．既知の検出対象の音響信号とマイクから得られた音響信号からパワースペクトルなどの特徴量を抽出して，Dynamic Time Warping [3] などのマッチングアルゴリズムで求めた距離を比較し，閾値以下であった場合に当該音が検出されたと見なす．「認識」とは複数の候補から尤もらしいものを選ぶことだが，Picognizer においては複数の検出対象とマイク入力音との距離を算出し，それらを比較選別すればよい．以後は記述を簡便にするため，Picognizer の機能について言及する際，「検出・認識」ではなく「検出」と表記する．

検出判定後はたとえば myThings [4] や Sony MESH [5] などのインターネットサービス・IoT デバイスなどへ通知することができ，多様な応用が可能である（**図 1**）．ライブラリは JavaScript のみで記述されているため，Web ブラウザがあればインストール不要で動作し，また現在開発中である Node.js 版を用いれば Raspberry Pi などの小型の組み込みコンピュータでの運用が可能となり，電子音を扱うガジェット作成にも活用が期待される．

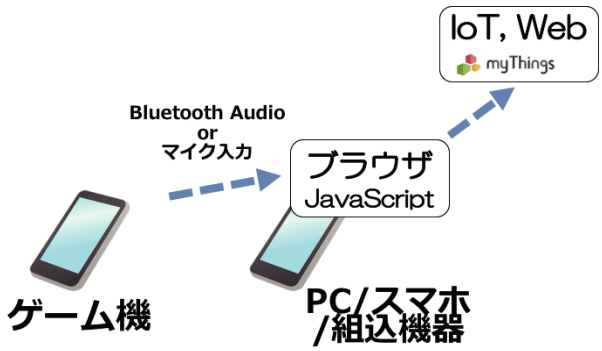

図 1 Picognizer の概要
Fig. 1 Overview of Picognizer.

本論文では，プロトタイプの実装，および音源の単純さとゲーミフィケーション構成への応用可能性のバランスが良い検出対象として，任天堂ファミリーコンピュータのようないわゆる 8 bit 音源を対象にした検出器の評価実験について報告する．

本論文の構成は以下のようになっている．まず次章で関連研究について述べる．次に提案システムの実装の詳細，および性能評価についての基礎的検討について示す．次に多様な応用シナリオを記述する．最後に現状の課題と今後の展望について議論する．

2. 関連研究

2.1 音響イベント検出

音声や音楽以外の音の認識は，従来は環境音認識（environmental sound recognition）という名称で進められてきたが，近年は，音響イベント検出（audio event detection/sound event detection）と呼ばれ，様々な研究が進められている．Cai らは，映像要約の文脈の下，ハイライトな音の検出（具体的には笑い声，拍手，歓声）の検出を隠れマルコフモデルを用いて行った [6]．Pikrakis らは，映画における暴力シーンの同定という文脈の下，映画のオーディオストリームに対してガンショット音を検出する問題を扱った [7]．実際にはガンショット音のほか，音楽，音声，叫び声などを含む 8 クラスを識別するベイジアンネットワークを学習した．Harma らは，生活環境における自動監視のために音を用いる研究を行った [8]．また，Lozano らは，聴覚障碍者への支援の観点から，電話の呼び出し音，アラーム音，ドアの施錠音などの検出に取り組んだ [9]．

これらは，スペクトル，MFCC，zero crossing rate など一般的に広く用いられている特徴量を用いている．そのほかにも，音のシーンを階層的にモデル化したり [10]，非負値行列因子分解を用いて音響イベントのスパース表現を抽

出したりするもの [11] など様々な研究が行われている．また，他の関連分野と同様に，深層学習を用いた研究も増えつつある [12], [13].

こういった研究事例の増加にともない，音響イベント検出の精度をコンテスト形式で競う試みも始められている．IEEE の AASP（Audio and Acoustic Signal Processing）Technical Commiitee は，DCASE（Detection and Classification of Acoustic Scene and Events）というものを始めた．ただし，基本的には実世界の様々な音響イベントをロバストかつ高精度に検出することを目的としており，我々のように電子音に限定し，特定の電子音のみを簡便な処理で検出するという立場ではない．

2.2 Web 上の音の検出・認識プラットフォーム

一方，音声認識に関しては，近年 Web 上の API が数多く公開され，これらを活用して音声認識対応の Web アプリケーションなどを簡単に作成できる状況が確立されつつある．たとえば, Bing Speech API [14], Google Cloud Speech API [15]，IBM Watson Speech to Text [16]，docomo 音声認識 API [17] などが有名である．これらは，大規模な学習データと最新の機械学習技術を用いて一般の人の音声を認識することを目的としており，特定の音の検出を目的としているわけではない．

また，Shazam [18] のように，スマートフォンのマイク越しに音楽を入力すると，その曲名やアーティスト名を答えてくれるサービスも登場している．近年の音楽のネット配信の普及にともない，こうしたサービスの需要は増しており，今のところ Web サービスに組み込むための API は公開されていないが，API が公開され，様々な Web サービスの開発が始まる可能性は十分にある．

2.3 音の検出・認識機能の IoT デバイスへの適用

音の検出や認識機能を組み込んだ IoT デバイスも登場している．著名なものとして，Amazon Echo [19]，Google Home [20]，Listnr [21] などがあげられる．これらは生活環境に置かれた IoT マイクデバイスが常時周囲の音を取得し，クラウド型の認識エンジンを経て，結果に応じた IoT 機器の連携などを行うものである．主に音声認識を想定したアプリケーションが提案されているが，ユーザが独自の認識エンジンを追加することが可能になれば，本研究の提案システムも導入しこれらの機器を活用することは可能であり，共存できるものと考えられる．

MagicKnock [22] もハードウェア構成としては IoT マイクデバイスと同様だが，置かれた面へのユーザのノック動作の振動パターンを集音，認識し，IoT 機器への入力として扱うシステムである．

2.4 プログラミング環境

Sikuli [23] は，マウスで特定のアイコンをクリックする，などの GUI オートメーションを実現する Python プログラミング環境である．アイコン画像などをスクリーンショット機能で撮影し，それをプログラムコードに直接貼り付けることで，「この画像を対象にする」という指示が行える．実行時は画像のテンプレートマッチングにより照合が行われる．音声を扱う本研究とは，映像を扱っている点で相違点があるが，既存システムを改変することなく，外付けすることで機能拡張するという点で共通点があり，Sikuli と本研究の提案システムを併用することで映像と音響を併用した既存システム拡張を行うことが可能となる．Kato らは Sikuli を発展させ，人間の身体動作やロボットの姿勢などのスナップショットデータをマッチング対象とする拡張を行っている [24].

IFTTT [25] および myThings は，IoT 機器や各種 web サービスなどを「起動条件」と「行う動作」の組である IF-THEN ルールにより記述できる，簡便なプログラミングを可能にしたサービスである．本研究もこのようにエンドユーザプログラマが日常生活を気軽に拡張できることを指向したものであり，IDCF クラウドサービスの meshblu サーバを用いることにより，myThings への接続を可能にしている．

srt.js [26] は，YouTube の動画の字幕情報として，実行時間を指定した JavaScript を埋め込むことで動画コンテンツを拡張することを簡便に行うことができるフレームワークである．既存コンテンツを外付けの形で拡張する JavaScript フレームワークという点で本研究と関連がある．

3. Picognizer

3.1 実装

Picognizer は JavaScript のみで記述された簡易な電子音検出ライブラリである．getUserMedia によりマイク入力を扱える Web ブラウザで動作する．PC では Mac および Windows の Firefox と Chrome で，スマートフォンでは Android の Firefox で動作を確認している．

Picognizer の処理フローは図 **2** のようになっている．検出対象となる「正解」の電子音源を wav 形式または mp3 形式で入力すると，音響特徴量計算ライブラリである Meyda [27]

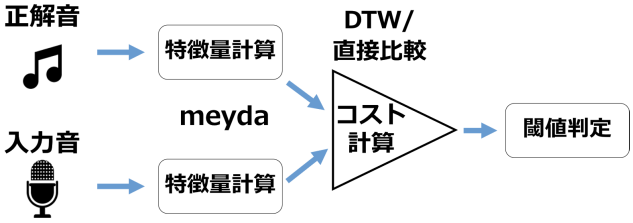

図 **2** Picognizer の処理フロー
Fig. 2 Workflow of Picognizer.

によりパワースペクトル，MFCC など，17 種類の特徴量の組合せからなる特徴量ベクトルが抽出される．その際，正規化やローパスフィルタなどの一般的な処理も行われる．また，検出開始するとマイク入力から音声を取得し，同様に特徴量ベクトルに変換される．

正解の特徴量ベクトルと入力の特徴量ベクトルは Dynamic Time Warping 法，および直接比較法によって比較され，コストが計算される．Dynamic Time Warping 法の実装には，1 次元の Dynamic Time Warping 法の npm パッケージである dtw [28] を多次元に拡張して用いた．

また直接比較法とは，入力音声と正解について，Dynamic Time Warping のように伸縮を考慮せず，ナイーブにフレームごとに比較しコストを算出する手法である．コストが閾値以下であれば，検出されたと見なして事後処理を行う．

なお，Dynamic Time Warping 法，直接比較法のいずれにおいても，各フレームにおけるコストの算出には特徴量ベクトル間の距離関数を定義する必要がある．電子音検出では，「正解の音声が含まれているが他の音も鳴っている」という状況も検出対象としたいため，次式のような非対称な距離関数 $Asym\_Dist$ を定義して用いた．

$$Asym\_Dist(\boldsymbol{target}, \boldsymbol{input}) = Dist(\boldsymbol{target}, \boldsymbol{Mask}(\boldsymbol{target}, \boldsymbol{input}))$$

ここで，$\boldsymbol{target}$ は正解音の特徴ベクトル，$\boldsymbol{input}$ は入力音声の特徴ベクトル，$Dist(\boldsymbol{A}, \boldsymbol{B})$ はベクトル $\boldsymbol{A}$，ベクトル $\boldsymbol{B}$ の間のユークリッド距離を表し，$\boldsymbol{Mask}(\boldsymbol{A}, \boldsymbol{B})$ はベクトル $\boldsymbol{A}$ とベクトル $\boldsymbol{B}$ の要素ごとの最小値からなるベクトルである．すなわちその i 番目の要素 $Mask_i(\boldsymbol{A}, \boldsymbol{B})$ は以下のように定義される．

$$Mask_i(\boldsymbol{A}, \boldsymbol{B}) = min(A_i, B_i)$$

これを図解したものが**図 3** である．

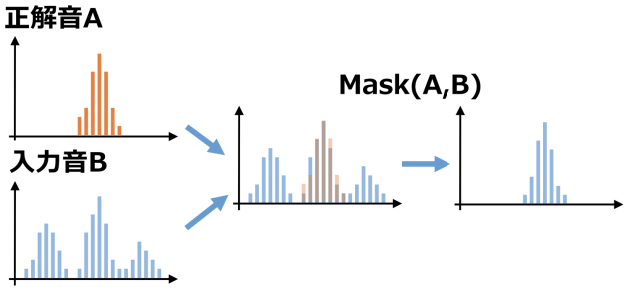

図 3 $\boldsymbol{Mask}(\boldsymbol{A}, \boldsymbol{B})$ の図解．正解音 $\boldsymbol{A}$ と入力音 $\boldsymbol{B}$ を直接比較すると，BGM などの正解音以外の音が混入している際に性能が低下する．そこで $\boldsymbol{Mask}(\boldsymbol{A}, \boldsymbol{B})$ を計算し，正解音 $\boldsymbol{A}$ と $\boldsymbol{Mask}(\boldsymbol{A}, \boldsymbol{B})$ を比較する

Fig. 3 Graphical explanation of $\boldsymbol{Mask}(\boldsymbol{A}, \boldsymbol{B})$. Naïvely comparing target $\boldsymbol{A}$ with input $\boldsymbol{B}$ results in low performance when $\boldsymbol{B}$ contains noise derived from sources such as background music. To exclude the noise, $Asym\_Dist$ calculates the Euclidean distance between $\boldsymbol{A}$ and $\boldsymbol{Mask}(\boldsymbol{A}, \boldsymbol{B})$.

表 1 ブラウザ版 Picognizer に指定するパラメータ

Table 1 Detection parameters for browser-based Picognizer.

パラメータ	説明
src	検出対象音源ファイルのURL (.wav or .mp3)
st,et	検出対象音源中で検出に使う箇所の開始時刻と終了時刻
surl	検出時に実行されるJavaScriptファイルのURL
cri	検出判定を行うコストのしきい値
mode	コスト計算アルゴリズム (“dtw” :DTW or “direct” :直接比較)
wfunc	窓関数(default: "hamming")
ft	Meydaで抽出する特徴量名の配列 (default: [“powerSpectrum”])
frame	特徴量抽出の周期 (default: 0.02 [s])
dur	コスト計算の周期 (default: 1.0 [s])

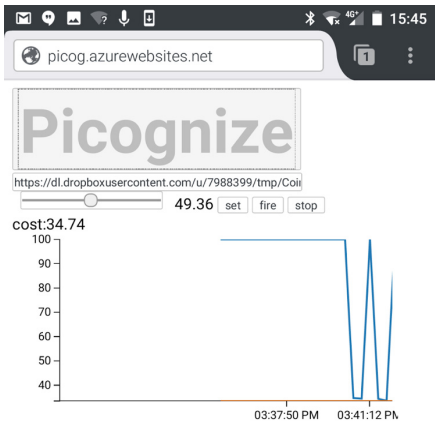

図 4 ブラウザ版の Picognizer の画面

Fig. 4 Screenshot of browser-based Picognizer.

3.2 使用方法

3.2.1 Web ブラウザによる使用

以下に Web ブラウザを念頭に，Picognizer の使用方法を示す．

Picognizer はインストール不要であり，以下のように URL にアクセスするだけで活用可能である．

https://qurihara.github.io/picognizer/script.html?cri=10&surl=http://xxx.net/bg_red.js&src=http://zzz.com/target.mp3

表 1 に，クエリ文字列として指定可能なパラメータの一覧を示す．この例では，src パラメータで指定した音源ファイルを正解とし，cri パラメータをコストの閾値とし，コストが閾値を下回ったときに実行する JavaScript ファイルを surl パラメータで指定している．**図 4** にこの URL にアクセスしたときの画面の様子を示す．「Picognize」ボタンをクリックもしくはタップし，ブラウザにマイク利用の許可を与えると動作が開始される．画面下部にはコストの時系列と閾値がグラフ化されており，スライドバーで適切なレベルに閾値を調整可能である．

以下は検出時に画面の背景を赤く変化させるスクリプト bg_red.js の内容である．

```
setup = function(){
   console.log("Initialized.");
}
onfire = function(){
   document.bgColor = 'red';
}
```

ここで，setup 関数はサイトのロード時に実行され，onfire 関数は検出時に実行される．このように Picognizer は構成した電子音検出に関するすべての情報を URL に組み込めるため，保存および第三者との共有が容易である．

任意の JavaScript を実行可能であることは汎用的である反面，セキュリティ上の懸念がある．そこで，用途をより限定することでセキュリティ対策を図る使用法を開発した．Picognizer の主な活用方法は，何らかの電子音検出を起点として多様な IoT 機器，ネット上のサービスと接続することであろう．その際は検出時に MQTT メッセージブローカである meshblu サーバへの通信機能のみを持つ，以下のような URL を使用する．

https://qurihara.github.io/picognizer/trigger.html?cri=33&src=http://zzz.com/target.mp3&myuuid=XXXXXX&mytoken=YYYYYY&server=192.168.0.1&src=http//zzz.com/target.mp3

のような記法で URL を指定する．server，myuuid，mytoken は meshblu に接続するためのアカウント情報を指定しているクエリ文字列である．これにより，同じ meshblu サーバに接続している各種 IoT 機器との連携，および様々なネット上のサービスどうしの連携を容易に行うことができる myThings との接続が可能である．

また，スマートフォン単体で実世界の「正解」音源の録音から始めるシナリオも考慮した使用法も開発した．

https://qurihara.github.io/picognizer/reco.html?myuuid=XXXXXX&mytoken=YYYYYY&server=192.168.0.1

のような URL を用いるとブラウザ上での録音用 UI が表示される（図 5）．

3.2.2 直接的なコーディング

前項による手法ではなく，直接的に JavaScript をコーディングすることも可能である．特徴量パラメータの設定，複数音源の同時検出（すなわち認識）など，より詳細な挙動の制御が可能である．

```
var Pico = require('picognizer');
var P = new Pico; //インスタンス化
var option = { //検出パラメータ
      windowFunc: 'hamming',
      mode: 'direct',
      feature:[ 'mfcc'],
      framesec:0.1,
      duration:1.0
};
P.init(option); //初期化
P.oncost(['audio1.mp3','audio2.mp3'],
      function(cost){
      //コスト算出時のイベントハンドラ
});
```

のように記載する．

3.2.3 音声入力方法

Picognizer に音声を入力する方法は，検出したい電子音の再生環境に依存する．**図 6**，**図 7**，**図 8** に様々なパターンにおける音声入力方法を列挙する．検出したい電子音が PC 上で再生されている場合は，その音声をマイク入力として扱う仮想的なマイクデバイス（ステレオミキサなどと呼ばれる）を準備すれば，雑音の影響なく同一 PC の Web

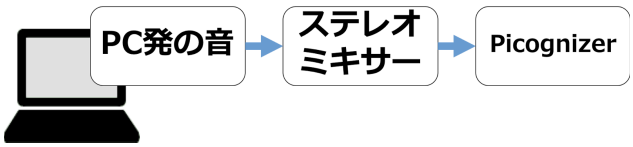

図 6 仮想的マイクデバイスによる接続
Fig. 6 Connection using a virtual microphone device on a PC.

図 7 オーディオケーブルによる有線接続
Fig. 7 Connection with an audio cable.

図 8 Bluetooth audio による無線接続
Fig. 8 Connection with a wireless Bluetooth audio.

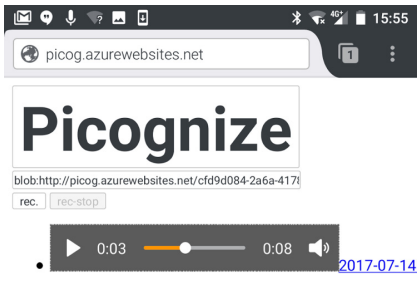

図 5 録音用 UI の画面
Fig. 5 Screenshot of browser-based Picognizer with the recording UI.

ブラウザ上の Picognizer で検出可能となる（図 6）．検出したい電子音が音声出力端子を備えた機器から発せられる場合は，各種オーディオケーブルなどで Picognizer 動作機器に入力することができる．図 7 はモニタのヘッドフォン端子からスマートフォンのマイク入力に有線接続する結線例，および図 8 は Bluetooth オーディオレシーバを用いて無線接続した例である．Picognizer が対象として想定するディジタルゲームの電子音は，通常は図 6，7，8 のいずれかの方法で Picognizer と接続できるため，環境音などのノイズの影響は考慮する必要はない．

4. 基礎性能評価

本論文では研究の初期段階としていわゆる 8 bit 音源のゲームの代表例である「スーパーマリオブラザーズ」の音源を用い評価実験を行った．8 bit 音源のレトロゲームは音源が単純であり検出が容易でありながら，余剰自由度 [1]（外部要素を導入できる余地）が大きく，ゲーミフィケーション構成が比較的容易であるという特徴を持つため，実用的な対象である．

4.1 距離関数とコスト計算アルゴリズムの性能評価

4.1.1 テストデータセット

まず実装したアルゴリズムの性能を比較するため，4 つの実験を行った．各実験の詳細は以下のとおりである．

1. BGM なしで対象の効果音を検出する．
2. BGM が付加された状態で対象の効果音を検出する．
3. BGM なしで対象の効果音を検出する．ただし，検出対象とは別の効果音の重なりを許す．
4. BGM が付加された状態で対象の効果音を検出する．ただし，検出対象とは別の効果音の重なりを許す．

各実験においてテスト用データセットは次のように構築した．検出対象には，コイン取得音（coin），ジャンプ音（jump），1 up きのこ取得音（1 up）の 3 種の効果音を選択した．35 秒のサンプリング周波数 22,050 Hz のモノラル音源を生成し，3 種の効果音のうち 1 種がランダムな箇所に，少なくとも 5 回以上含まれるようにした．また，検出低能低下を人為的に引き起こすために，スーパーマリオブラザーズの 9 種類の効果音から，「非対象音」として無作為に選び，音源に繰り返し混合した．実験 1 と 2 のデータにおいて複数の効果音の重なりはないが，実験 3 と 4 のデータでは効果音の重なりを許した．実験 1 と 3 のデータに対し，3 種の BGM のうち 1 種を 0 dB，5 dB の SN 比で付加し，それぞれを実験 2，4 用とした．なお BGM には，地上ステージ（outdoor），地下ステージ（underground），マリオが無敵になるときの BGM（star）の 3 種を選択した．

そして，評価ツール用に，各入力音源の検出対象音と非対象音すべてについて，オンセット（再生開始）時刻のラベル付けをすべて手動で行った．

4.1.2 評価方法

我々のインフォーマルな予備実験において，様々な音響特徴量の中でパワースペクトルがスーパーマリオブラザーズの効果音に対して最も良い検出性能だったため，音響特徴量にはパワースペクトルを使用した．各実験において，距離関数とコスト計算アルゴリズムに関し，複数の条件の組合せの検出性能を比較した．具体的には距離関数パラメータとしてユークリッド距離と $Asym\_Dist$，コスト計算アルゴリズムとしては DTW と直接比較を使用した．

各実験では，パワースペクトルを 40 ms ごとに音響特徴ベクトルとして抽出し，検出対象の効果音と入力音とのコストを算出する．再現率，適合率，F 値は Python の評価ツール mir_eval [29] を用いて算出した．各効果音で検出されたオンセット時刻の許容誤差は，正解のオンセット時刻から ±50 ms である．

なお，手動で行う閾値の設定については，各実験で最も高い再現率を得られるように定めた．この設定により誤検出は増えるが，ディジタルゲーム拡張の応用を考えた場合，画像照合を扱う Sikuli などの他の技術と Picognizer を統合し，適合率を高めることが可能であると考えたからである．

4.1.3 評価結果

効果音重複のない条件である実験 1 と実験 2 の結果について**表 2**, **表 3**, **表 4**, **表 5** に結果を示す．表中の S/N [dB] について，「clean」は実験 1 の結果に対応し，5 dB と 0 dB は実験 2 の結果に対応する．それぞれ，距離関数（ユークリッ

表 2 実験 1・実験 2 のユークリッド距離・DTW の結果
Table 2 Results of Euclidean distance and DTW condition in experiments 1 and 2.

検出対象	BGM	S/N [dB]	F値	適合率	再現率
coin	outdoor	clean	0.750	0.750	0.750
		5	0.667	0.714	0.625
		0	0.545	1.000	0.375
jump	underground	clean	0.545	1.000	0.375
		5	0.462	0.600	0.375
		0	0.462	0.600	0.375
1up	star	clean	0.500	0.667	0.400
		5	0.500	0.667	0.400
		0	0.500	0.667	0.400

表 3 実験 1・実験 2 のユークリッド距離・直接比較の結果
Table 3 Results of Euclidean distance and direct comparison condition in experiments 1 and 2.

検出対象	BGM	S/N [dB]	F値	適合率	再現率
coin	outdoor	clean	1.000	1.000	1.000
		5	0.875	0.875	0.875
		0	0.667	0.714	0.625
jump	underground	clean	0.889	0.800	1.000
		5	0.842	0.727	1.000
		0	0.842	0.727	1.000
1up	star	clean	1.000	1.000	1.000
		5	1.000	1.000	1.000
		0	1.000	1.000	1.000

表 4 実験 1・実験 2 の *Asym_Dist*・DTW の結果
Table 4 Results of *Asym_Dist* and DTW condition in experiments 1 and 2.

検出対象	BGM	S/N [dB]	F値	適合率	再現率
coin	outdoor	clean	0.750	0.750	0.750
		5	0.625	0.625	0.625
		0	0.400	0.429	0.375
jump	underground	clean	0.545	1.000	0.375
		5	0.400	1.000	0.250
		0	0.222	1.000	0.125
1up	star	clean	0.444	0.500	0.400
		5	0.444	0.500	0.400
		0	0.100	0.067	0.200

表 5 実験 1・実験 2 の *Asym_Dist*・直接比較の結果
Table 5 Results of *Asym_Dist* and direct comparison condition in experiments 1 and 2.

検出対象	BGM	S/N [dB]	F値	適合率	再現率
coin	outdoor	clean	1.000	1.000	1.000
		5	1.000	1.000	1.000
		0	0.933	1.000	0.875
jump	underground	clean	0.941	0.889	1.000
		5	0.889	0.800	1.000
		0	0.889	0.800	1.000
1up	star	clean	1.000	1.000	1.000
		5	1.000	1.000	1.000
		0	1.000	1.000	1.000

表 6 実験 3・実験 4 のユークリッド距離・DTW の結果
Table 6 Results of Euclidean distance and DTW condition in experiments 3 and 4.

検出対象	BGM	S/N [dB]	F値	適合率	再現率
coin	outdoor	clean	0.600	0.857	0.462
		5	0.455	0.556	0.385
		0	0.421	0.667	0.308
jump	underground	clean	0.194	0.167	0.231
		5	0.200	0.176	0.231
		0	0.333	0.600	0.231
1up	star	clean	0.444	0.667	0.333
		5	0.500	1.000	0.333
		0	0.235	0.182	0.333

ド距離と *AsymDist*）とコスト計算アルゴリズム（DTW と直接比較）の影響を比較した．結果から，*Asym_Dist* と直接比較の組合せ（表 5）が最も良い検出性能だったことが分かる．

次に，**表 6**，**表 7**，**表 8**，**表 9** に効果音重複のある条件である実験 3 および実験 4 の結果を示す．表中の S/N [dB] について，「clean」は実験 3 の結果に対応し，5 dB と 0 dB は実験 4 の結果に対応する．再度，距離関数（ユークリッド距離と *Asym_Dist*）とコスト計算アルゴリズム（DTW と直接比較）の影響をそれぞれ比較した．結果から，再び *Asym_Dist* と直接比較の組合せ（表 9）が最も良い検出性能だったことが分かる．

表 7 実験 3・実験 4 のユークリッド距離・直接比較の結果
Table 7 Results of Euclidean distance and direct comparison condition in experiments 3 and 4.

検出対象	BGM	S/N [dB]	F値	適合率	再現率
coin	outdoor	clean	0.870	1.000	0.769
		5	0.636	0.778	0.538
		0	0.636	0.778	0.538
jump	underground	clean	0.553	0.382	1.000
		5	0.867	0.765	1.000
		0	0.867	0.765	1.000
1up	star	clean	1.000	1.000	1.000
		5	1.000	1.000	1.000
		0	1.000	1.000	1.000

表 8 実験 3・実験 4 の *Asym_Dist*・DTW の結果
Table 8 Results of *Asym_Dist* and DTW condition in experiments 3 and 4.

検出対象	BGM	S/N [dB]	F値	適合率	再現率
coin	outdoor	clean	0.636	0.778	0.538
		5	0.571	0.750	0.462
		0	0.333	0.600	0.231
jump	underground	clean	0.261	0.300	0.231
		5	0.267	1.000	0.154
		0	0.143	1.000	0.077
1up	star	clean	0.400	0.500	0.333
		5	0.400	0.500	0.333
		0	0.400	0.500	0.333

表 9 実験 3・実験 4 の *Asym_Dist*・直接比較の結果
Table 9 Results of *Asym_Dist* and direct comparison condition in experiments 3 and 4.

検出対象	BGM	S/N [dB]	F値	適合率	再現率
coin	outdoor	clean	0.870	1.000	0.769
		5	0.818	1.000	0.692
		0	0.762	1.000	0.615
jump	underground	clean	0.963	0.929	1.000
		5	0.963	0.929	1.000
		0	0.929	0.867	1.000
1up	star	clean	1.000	1.000	1.000
		5	1.000	1.000	1.000
		0	1.000	1.000	1.000

4.1.4 考察

実験を通して *Asym_Dist* と直接比較の組合せは，F 値が多くの条件下で 1.0 に近くなり，最小でも 0.762 であったため，Picognizer は実用的な検出性能を持っていると考えられる．なお，本実験では検出性能を乱す要因として，BGM と非対象音からなる「ゲーム由来の雑音」のみを扱い，環境雑音については扱っていない．しかし 3.2.3 項で論じたように，通常は有線接続や無線接続などによって Picognizer へ入力される音源についてはゲームに由来するもののみに限定できるため，この仮定は実際の使用時に即した現実的なものであると考えられる．

距離関数については，検出対象音に加えて BGM と非対象音が混在する条件下において，*Asym_Dist* のマスク処理

が検出対象音以外の影響を防ぐのにうまく機能したといえる．*Asym_Dist* は，検出対象音のスペクトルと非対象音のスペクトルとの重なりが小さい場合，理論的に有効である．今回用いたスーパーマリオブラザーズの音源はこのような特性を持っていた可能性がある．異なる音響特性を持つ他のディジタルゲームについては，さらなる実験が必要である．

コスト計算アルゴリズムについては，再生ごとの音響的変動の小さい電子音の性質に直接比較法が適合し高性能が得られたと推測される．しかし，直接比較法は検出対象音の再生タイミングとコスト計算開始タイミングのずれ，あるいは計算機の処理能力不足によるフレーム欠損があった場合にそれを考慮することができないため，使用にあたっては今日市販されている通常性能のノートブック PC 程度あるいはそれ以上の計算速度を持つマシンを採用し，表 1 の「frame」と「dur」パラメータをより小さく設定する必要がある．これは，そのずれを小さくすべく，頻繁に特徴量抽出とコスト計算を行う設定である．

一方，DTW は直接比較よりも 1 回のコスト計算に多くの計算量を必要とするが，伸縮を許容しながらマッチングを行うアルゴリズムの性質上，先述のような時間的ずれやフレーム欠損に対して一定のロバスト性を備えていることが期待される．よって，ユーザは，直接比較法で膨大な計算する能力がないスマートフォンなどの低速機を活用する際には，「frame」と「dur」パラメータを大きな値に調整して，DTW を使用し平均計算量を削減することができる可能性がある．ただしこの場合レイテンシ（検出対象音の発生から検出までの時間の遅れ）は増加する．マシンパワーに基づくこのようなパラメータ最適化戦略の定量的評価は，今後の課題である．

4.2 レイテンシの評価

4.2.1 Picognizer の処理時間に関する評価

(1) 定義と計測

次にレイテンシについてさらに詳しく評価する．本システムの扱うべきレイテンシには 4 種類の成分がある．検出対象音の発生開始時刻から発生終了時刻までの間隔 a，そこから検出が始まるまでの間隔 b，そこから検出が完了するまでの間隔 c，そしてそこから検出完了後に行うユーザ独自の情報処理の完了までの間隔 d である（**図 9**）．電子音検出システムにおけるレイテンシは一般的には $b+c+d$ で表される．このうち d は応用対象によって様々な値をとるため，$b+c$ を小さくすることを考える．b については単純に Picognizer が検出処理を行うタイマの周期のみに起因するものであり，一様乱数に近いモデル化が可能であると考えられる．c については Picognizer の特徴量抽出およびコスト計算アルゴリズムの計算量に依存するものであり，DTW よりも直接比較法が軽量であることはこれまで示したとおりである．b と c については，高性能な計算機を活用することで c を小さくし，それにあわせてタイマ周期を短くすることで b も小さくさせることが可能である．

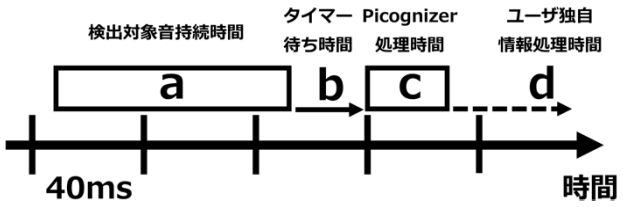

図 9 レイテンシの構成成分
Fig. 9 Components of latency.

前節で述べた実験環境においては，40 ms のタイマ周期でコスト計算を行っているため，b は 0 ms と 40 ms の間の値をとり，平均 20 ms 程度であると予想される．また c について，2 種類の計算機 PC1（Windows 10，Intel Core i7-470K 3.5 GHz CPU，メモリ 16 GB，Firefox 使用）および PC2（Windows 10，Intel Core i3-2367M CPU 1.4 GHz，メモリ 10 GB，Firefox 使用）において 1 ms の時間分解能で 1,500 回の計測を行ったところ，コスト計算に直接比較法を用いた場合に PC1 では平均値 2.80 ms（標準偏差 1.22 ms），PC2 では平均値 6.91 ms（標準偏差 5.14 ms）であった．したがって PC1 を用いた場合，$b+c$ は平均的には 23 ms 程度となる．さらに，現状ではタイマ周期 40 ms に対して c が小さいため，これらが同程度になるまでタイマ周期を短くできる可能性がある．たとえばタイマ周期を 10 ms にすることによって b が平均 5 ms 程度になるため，$b+c$ は平均で 8 ms 程度に削減可能であると予想される．

(2) 考察

Picognizer での電子音検出後にユーザが行いたい情報処理は多様であり，たとえばその電子音のなった回数を記録し後に参照するような用途であれば，レイテンシは 10 分以上かかっても特に実用上問題とならない．現に IoT 機器や各種 web サービスを簡便に接続できるサービスである IFTTT [25] では，機器やサービス間のメッセージ送受信に最大で 15 分程度の遅延があるが，サービスとして成立している．

一方でゲームの電子音に連動してその場でユーザに情報をリアルタイムフィードバックするような情報処理を考えた場合，レイテンシは最小化することが望ましい．ここで，達成すべきレイテンシの基準として，Claypool らによる，オンラインゲームにおける操作入力とその反映までの時間について許容可能なレイテンシの研究を参照する [30]．これによると，ゲームの種類によって許容できるレイテンシには違いがあるが，比較的低遅延を要求する FPS（一人称視点の射撃ゲーム）において，その基準は 100 ms と論じられている．

ゲーム操作入力の反映に対するレイテンシと，ゲームからの出力に基づく付加的情報処理に対するレイテンシを同

列の基準で論じることに議論の余地はあるが，仮にこの基準と同程度のレイテンシを目標とした場合，$b+c$ が平均 8〜23 ms 程度におさえられている現状はまだ 100 ms に対しゆとりがあるため，さらにその後に d で行う処理の内容によっては $b+c+d$ を許容範囲内にとどめることが十分可能であると判断される．

4.2.2 検出対象音の時間長削減と検出性能の関係の評価

次にレイテンシの成分 a について検討する．「与えられた音」を検出するシステムとして a はその音の継続時間そのものであるため，通常は削減できない．しかし十分な検出性能を保ちつつ検出対象音の末尾部を削減（スライス）できれば，a を減少させることが可能である．一般的にはスライスによって対象音源を規定する情報が失われていくため他の音との区別ができにくくなり，適合率は低下することが予想されるが，効果音によって弁別力のある個所は異なるため，複数の効果音に対しスライスが検出性能に与える影響について以下のように評価した．

(1) 評価方法

評価音源は 4.1 節の実験 4 のように，検出対象音が他の効果音と重複しうる状況で，BGM が 0 dB で付加されているものとした．アルゴリズムについては 4.1 節の実験において性能の良かった「*Asym_Dist* および直接比較」を採用した．

検出対象の効果音としてコイン取得音（coin），ジャンプ音（jump），1 up きのこ取得音（1 up）の 3 種，および BGM として地上ステージ（outdoor），地下ステージ（underground），マリオが無敵になるときの BGM（star）の 3 種を組み合わせた．検出対象音はもともとの長さの音源に加えて，冒頭から 0.05，0.1，0.2，0.5，0.7 秒の時点までスライスを行い，特徴ベクトルを算出した．4.1 節の実験と同様に各条件において最も再現率が高くなるように閾値を設定し，F 値を算出した．その結果を**図 10**，**図 11**，**図 12** に示す．

(2) 評価結果と考察

全般的に，ある程度まではスライスをすることによって高い F 値を維持することが可能であることが分かった．本実験においては 0.2 秒にスライスした場合に性能が低下しているのは 1 up きのこ取得音（1 up）と地下ステージ（underground）の組合せのときのみであった．

一方で過剰に短くスライスしすぎると性能は低下する結果となった．特にスライス後の長さが 0.2 秒以下の場合に性能低下は顕著になった．これは現在の 1 回の特徴量検出区間（表 1 における frame パラメータ）が 40 ms であり，わずか数点の時刻による特徴量を用いてマッチングをしなければならないことによる，そもそもの弁別力の低下が原因であると予想される．

興味深いことに，コイン取得音（coin）およびジャンプ音（jump）において，むしろ多少スライスしたほうがいっさいスライスしないよりも性能が向上する結果となった．**図 13** にコイン取得音（coin）のスペクトルを可視化したものを示す．この効果音は残響音が大部分を占めるため，もともとの音源では残響音部分での誤検出が発生しやすかった可能性がある．スライスにより残響音が削減され，より適切に検出が行われるようになったと予想される．ジャンプ音（jump）においても同様の背景があったものと考えられる．

以上の結果から，検出対象音や BGM の音響的性質に依存するものの，検出性能を維持しつつスライスによりレイテンシを減少させる方略は試行の価値があると考えられる．

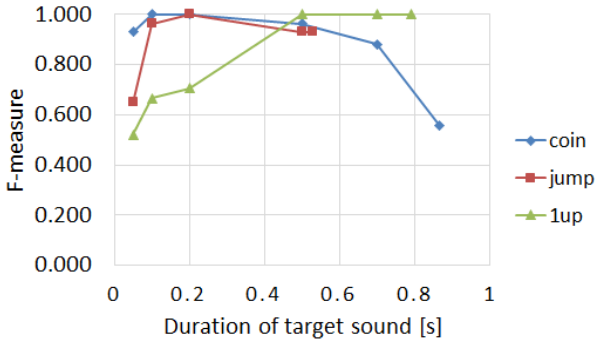

図 11 「地下ステージ BGM」における各検出対象音の F 値
Fig. 11 The F-measure of each audio source in the "underground" BGM.

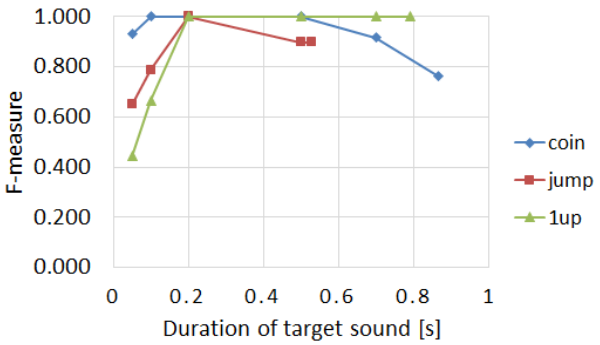

図 10 「地上ステージ BGM」における各検出対象音の F 値
Fig. 10 The F-measure of each audio source in the "outdoor" BGM.

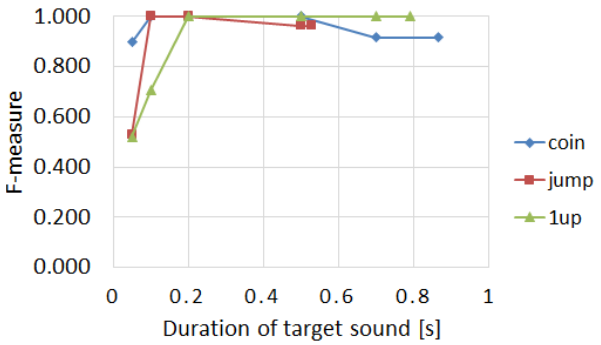

図 12 「無敵時 BGM」における各検出対象音の F 値
Fig. 12 The F-measure of each audio source in the "star" BGM.

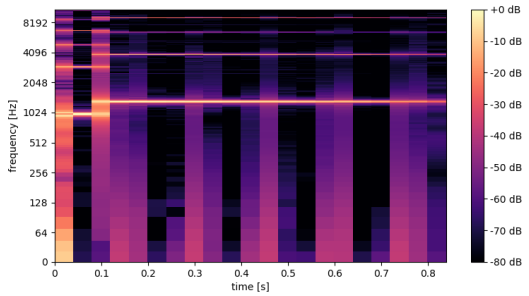

図 13 「コイン取得音」のスペクトルの可視化

Fig. 13 The Spectrum visualization of the “coin” sound effect.

ある効果音について，様々なスライスを自動的に行い，性能を比較することで最適なスライス量をサジェストするような開発者支援ツールの提供は有効かもしれない．その開発は今後の課題である．

5. 応用シナリオ例

本節では Picognizer をディジタルゲーム音検出に適用した場合の応用シナリオ例について述べる．デモ映像（https://youtu.be/-2MtArZAtfg）には実際の動作の様子が収録されているので参照されたい．

5.1 ゲームの拡張

5.1.1 Toolification of Games

図 14 は，スーパーマリオブラザーズのプレイ中にコインを取得した際，コイン取得の効果音を Picognizer で認識し，実世界で硬貨を 1 つ排出するシステムの例である．硬貨の排出は，Sony MESH の GPIO タグに接続したサーボモータを操作することによって実現している．硬貨の排出には，第三者が用意した硬貨であれば賞金，ユーザ自身が用意した硬貨であれば課金もしくは罰金の意味合いを付与可能である．栗原は文献 [1] において，スーパーマリオブラザーズにおけるコイン取得を慈善団体への寄付行為に連動させることで，寄付行為に対する参加障壁を下げる社会的目的を達成する Coins For Two というシステムを提案したが，本例はその実世界版の実装である．イベント会場などで来場者に課金のうえでゲームをプレイしてもらい，排出されたコインを募金箱に収納することで運用する．

本例のように，既存ゲームの拡張を行う際，新たに付与される要素が単純なエンタテインメント価値の増強ではなく，社会善などの非ゲーム的目的の達成に寄与する場合，それを栗原は Toolification of Games と定義している．これはゲーミフィケーションの派生概念であり，Picognizer は宿主となる既存ゲームのソースコードを入手することなく Toolification of Games の構築を簡便に行える基盤技術である．

5.1.2 Hue との連携

図 15 は，スマートフォンのゲーム「モンスターストライク [31]」において，ボス敵の登場時の警告音を検出し，IoT 照明器具 Hue [32] の色合いを変化させ，臨場感を増強するゲーム拡張である．スマートフォンからの音声を Bluetooth audio 経由で Picognizer の動作している PC へと入力しているため，有線接続は不要であり，スマートフォンの把持姿勢やユーザの位置などに制約を課さずにゲームのエンタテインメント価値の増強を図れている．システムをさらに発展させれば，大勢の人があつまるゲームの公開対戦・観戦イベントなどで，本システムと会場の音響や照明などを連動させたコンテンツの作成を行うことができるだろう．

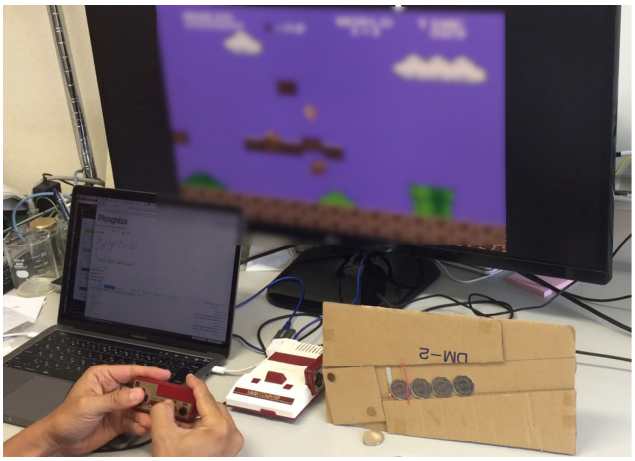

図 14 スーパーマリオブラザーズを用いた Toolification of Games 事例，「Coins For Two」．コイン取得時の効果音を検出し，コインサーバから硬貨が排出される

Fig. 14 An example of Toolification of Games with Super Mario Brothers. Picognizer detects the sound effects of gaining a coin to eject a real coin.

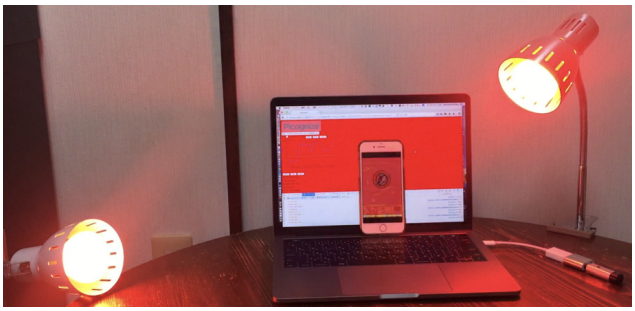

図 15 モンスターストライクと Hue との連携の例．ボス敵登場時の警告音を検出し，Hue の色彩が変化する

Fig. 15 An example of augmentation of the smartphone game Monster Strike. The real-world lighting delivered by an IoT lighting solution Philips Hue turns red when the game alerts the player of the boss enemy’s arrival.

5.2 ゲームの自動操作

図 16 は，Nintendo Switch のゲーム「1-2-Switch [33]」における，ガンマン（QuickDraw）というミニゲームを自動操作するシステムである．これは「Fire」という掛け声

図 16 QuickDraw の自動操作の例．「Fire」の掛け声が検出されると Nintendo Switch のコントローラが IoT モータ Webmo によって回転し，銃を発射する入力が行われる

Fig. 16 An example of an autoplay system for QuickDraw, a game for Nintendo Switch. The game controller is manipulated physically by an IoT motor Webmo.

を聞いた後に対戦相手より早くコントローラを水平にしてボタンを押すことで，勝利となる．本システムは，「Fire」の掛け声を Picognizer で検出し，IoT ステッピングモータの Webmo を用いてゲームコントローラを回転させるものである．筐体とボタン押下用の突起をブロック玩具で作成した．本例では音声のみを手がかりに進行するゲームを扱ったが，画像照合による GUI オートメーション開発環境である Sikuli と組み合わせることで，より複雑なゲームの自動操作が可能であるだろう．

6. 議論

6.1 試運用結果から得られた課題

前章で述べた応用シナリオ例のうち，5.1.1 項の事例の試運用を通じて著者らが経験した Picognizer の課題について述べる．まず，用いる計算機の性能について，4.1 節の性能評価と同様の 40 ms 周期の特徴量抽出と直接比較法によるコスト計算を行ったところ，ノートパソコンである 3.1 GHz Intel Core i5，8 GB メモリ内蔵の MacBook Pro 2016 年モデルでは問題なく動作したが，Android スマートフォンである Nexus 5X（Qualcomm Snapdragon808（MSM8992）1.8 GHz + 1.4 GHz ヘキサコア，メモリ 2 GB）では処理能力不足でブラウザがフリーズし動作しなかった．コスト計算アルゴリズムを DTW にし，特徴量抽出の周期を 200 ms，コスト計算の周期を 500 ms に設定することで，継続的な動作を実現できたものの，レイテンシが大きく，また誤検出を防ぐ閾値の設定は困難だった．2015 年モデルのスマートフォンでの高精度な Picognizer の運用にはまだ処理能力が不足しているという結果となった．

また本運用事例では，ゲーム内でコインを取得した際にコインサーバから硬貨が排出されるが，連続で複数のコインを取得した際の挙動に問題が生じた．1 つには，連続でコインを取得した際に，複数のコイン取得音が重畳し古いものの再生が中断されてしまうことによる問題である．本運用では，コイン取得音が完全に鳴り終わるまでの区間を検出対象としたため，検出の取りこぼしが生じた．このような効果音重複時の再生中断は，同時発音数の制約の厳しいレトロゲームでは起こりがちな現象であると考えられるが，4.2.2 項で評価したように検出対象音をスライスすることである程度は改善可能だと考えられる．ここから，特にレイテンシの軽減が求められない応用対象であっても，スライスを試みることはこの問題を未然に防ぐために有益なプラクティスといえるかもしれない．

もう 1 つには，硬貨排出のためにサーボモータが往復動作する所要時間がコイン取得間隔よりも長くなったことによる問題である．本運用では，コイン排出動作中の新たなコイン排出指示受信を想定していないサーボモータ制御プログラムであったため，適切な枚数のコイン排出ができなかった．コイン排出指示を貯めておく待ち行列を実装することで，解決を図るべきであろう．

6.2 展望

6.2.1 パラメータ設定支援

Picognizer は伝統的なテンプレートマッチング手法により，正解とする音響データと入力信号の間のコスト（近さ）を算出するところまでが機械的に行われ，コストがどの値以下であれば「検出」と判断するかは，ユーザの手作業により設定される．現在はスライドバーとコストのグラフ表示による閾値 UI が提供されているが，よりきめ細かい支援が可能である．たとえば準備作業として正解音響データの発生時と非発生時を実施もしくはエミュレートして，おおよそのコストの変動幅を見積もり，仮の閾値を推薦することなどである．

また，音響学に詳しくないユーザのために，対象となる音響データの性質と使用する計算機の性能，およびニーズに基づき，特徴量や検出パラメータを推薦するウィザードの実装も有意義であろう．すなわち，人の声なのか，ディジタルゲームの効果音なのか，計算負荷を上げてでも再現率を上げたいのか，非力な計算機で扱いたいのか，などを対話により取得することである．これらは今後の課題である．

6.2.2 より複雑な音響特性を持つ電子音への拡張

本論文では，認識対象として任天堂ファミリーコンピュータのようないわゆる 8 bit 音源を対象にした評価について報告した．電子音といっても対象は多様であり，そのすべてに対し高性能の検出器を構成するのは容易ではない．研究の第 1 歩として，音源の単純さとゲーミフィケーション構築への応用可能性のバランスが良い対象として，これを選定した．より現代的なゲーム機が採用する効果音に対応させるには，用いる音響特徴量の改善が必要である．その 1 つの方法が，現在採用している meyda 以外の音響特徴量抽出ライブラリを用いることである．たとえば，JS-Xtract [34] では，meyda では対応していない wavelet 変換や基本周波

数などを使用することができる．また，FFT に特化した JavaScript ライブラリが多数開発され比較されている [35] ので，これらを用いることも考えられる．

一方，近年，variational autoencoder や restricted Boltzmann machine などを用いて，スペクトログラムなどから潜在特徴量を抽出する抽出器を学習する研究が行われている [36], [37]．また，多段の convolutional LSTM により構成した autoencoder で音響イベント検出のための特徴量抽出を試みた研究も存在する [38]（音を含む時系列メディアの教師なし特徴量学習は文献 [39] にまとめられている）．多様な音を収録したデータベースを用いて特徴量抽出器を学習しておき，公開することで，こうした技術の活用も可能になると考えられる．

また，画像認識分野における VGG16 [40] などのように，大規模かつ一般的な音響イベント検出のための深層学習の学習済みモデルが公開されれば，そのモデルの前段を用いて少数の教師データにより転移学習したり，音響特徴量抽出器として活用し DTW や直接比較などを行うことにより，性能向上が可能かもしれない．

6.2.3 セキュリティの改善

Picognizer は URL の通知により，第三者の作成した任意の JavaScript の実行環境を広く共有できる．流通が容易である反面，悪意のあるコードの実行を許す点でセキュリティに問題がある．現状でも Picognizer は，単純に meshblu サーバへのイベント発火通知を行う機能に限定する実装によりある程度のセキュリティ対策はされているが，同様の問題を提起している srt.js を参考にした対策も有効である．すなわち，ローカルリソースへのアクセスを限定した（すなわち sandbox 化されている）[41] などの JavaScript-JavaScript インタプリタを導入する対策，JavaScript の実行をサーバ側で行う対策，および SNS と集合知を利用してコードの安全性を確保する対策などである．

6.2.4 アーキテクチャの検討

Picognizer は，ユーザが使用する端末内で動作するブラウザ上で動作するように設計されている．これは，サーバの準備や運用をすることなく手軽に使えることを目指したためである．別の設計指針としては，端末に入力された音響信号をサーバに送り，サーバ上で検出処理を行う方法が考えられる．Alsina-Pages らは，このようなアーキテクチャを centralized intelligence strategy と名付けた [42]．彼らは，複数の生活空間における音響イベントの発生状況を遠隔でモニタリングするシステムの設計において，centralized intelligence strategy と生活空間ごとに音響イベント検出を行い，検出結果のみ集約する distributed intelligence strategy を比較し，後者を採用した．主な理由は，モニタリングすべき生活空間の個数が増えたときに前者の方法ではサーバがボトルネックになること，および，生活音そのものは個人情報を含み，そのままサーバに送るには追加のセキュリティ対策が必要になることである．

Picognizer は，上述のとおり，ユーザの端末内のブラウザで動作させるのが基本であるため，一義的には，distributed intelligence strategy に対応する．これによりサーバレスの動作が実現でき，手軽さの実現に一役買っている．一方，centralized intelligence strategy を採用することも不可能ではない．Picognizer のプログラムをサーバ上で動かすことも原理的には可能である．将来的に計算コストの高い特徴量抽出などの導入が必要になった場合には有効なアプローチである．ただし，この場合，ユーザの端末からサーバ上に音響信号を送るコストを考慮する必要がある．

サーバ上で検出処理を行う別の利点として，検出処理のたびに入力された音響信号を蓄積できることがあげられる．多数の音響信号が蓄積されれば深層学習などの教師付き学習の学習データとして利用することができるようになり，より汎用の電子音検出・認識サービスを構築できる可能性がある．

7. おわりに

本論文では，エンドユーザプログラマがディジタルゲームの効果音などの電子音検出・認識を起点にした情報システム開発を容易に行える JavaScript ライブラリ，Picognizer を開発し，基礎的な性能評価と多様な応用事例の提示を行い，その有用性と今後の展望を議論した．今後はより複雑・多様な電子音へと対象を拡大する．Node.js による実装も完成し，組み込み機器などでの駆動を可能にし，活用の局面の拡大を目指す予定である．Picognizer の一般公開後はワークショップ開催などにより，普及と活用事例収集を行いたい．

謝辞 本研究は JSPS 科研費 JP15H02735, JP16H02867, 科研費 17H00749，および中山隼雄科学技術文化財団の研究助成を受けたものです．

参考文献

[1] 栗原一貴：Toolification of Games：既存ゲームの余剰自由度の中で非ゲーム的目的を達成するゲーミフィケーション周辺概念の提案と検討，情報処理学会論文誌，Vol.58, No.14, pp.1–13 (2017).
[2] Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N. and Kingsbury, B.: Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups, *IEEE Signal Processing Magazine*, Vol.29, No.6, pp.82–97 (2012).
[3] Sakoe, H. and Chiba, S.: Dynamic Programming Algorithm Optimization for Spoken Word Recognition, *IEEE Trans. Acoust. Speech Signal Process*, Vol.26, No.1, pp.43–49 (1978).
[4] myThings（オンライン），入手先 ⟨https://mythings.yahoo.co.jp/⟩（参照 2017-07-28）．
[5] Sony MESH（オンライン），入手先 ⟨http://meshprj.com/⟩（参照 2017-07-28）．

[6] Cai, R., Lu, L., Zhang, H.J. and Cai, L.H.: Highlight Sound Effects Detection in Audio Stream, *Proc. IEEE International Conference on Multimedia and Expo* (*ICME 2003*), Vol.III, pp.37–40 (2003).
[7] Pikrakis, A., Giannakopoulos, T. and Theodoridis, S.: Gunshot Detection in Audio Streams from Movies by Means of Dynamic Programming and Bayesian Networks, *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing* (*ICASSP 2008*), pp.21–24 (2008).
[8] Harma, A., McKinney, M.F. and Skowronek, J.: Automatic Surveillance of the Acoustic Activity in Our Living Environment, *Proc. IEEE International Conference on Multimedia and Expo* (*ICME 2005*) (2005).
[9] Lozano, H., Hernaze, I., Navas, E., Gonzalez, F.J. and Idigoras, I.: Household Sound Identification System for People with Hearing Disability, *Proc. Conference & Workshop on Assistive Technologies for People with Vision & Hearing Impairments: Assistive Technologies for All Ages* (*CVHI 2007*), Paper ID 27 (2007).
[10] Imoto, K. and Shimauchi, S.: Acoustic Event Analysis based on Hierarchical Generative Model of Acoustic Event Sequence, *IEICE Trans. Information and Systems*, Vol.E990D, No.10, pp.2539–2439 (2016).
[11] Lu, X., Tsao, Y., Matsuda, S. and Hori, C.: Sparse Representation based on a Bag of Spectral Exemplars for Acoustic Event Detection, *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing* (*ICASSP 2014*), pp.6255–6259 (2014).
[12] Wang, Y., Neves, L. and Metze, F.: Audio-based Multimedia Event Detection Using Deep Recurrent Neural Networks, *Proc. International Conference on Acoustics, Speech, and Signal Processing* (*ICASSP 2016*), pp.2742–2746 (2016).
[13] Cakir, E., Parascandolo, G., Heittola, T., Huttunen, H. and Virtanen, T.: Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection, *IEEE/ACM Trans. Audio Speech Lang. Process*, Vol.25, No.6, pp.1291–1303 (2017).
[14] Microsoft：Bing Speech API，Microsoft Azure（オンライン），入手先 ⟨https://azure.microsoft.com/ja-jp/services/cognitive-services/speech⟩（参照 2017-07-28）.
[15] Google Cloud: Cloud Speech API (online), available from ⟨https://cloud.google.com/speech/⟩ (accessed 2017-07-28).
[16] IBM Watson: Speech to Text (online), available from ⟨https://www.ibm.com/watson/developercloud/speech-to-text.html⟩ (accessed 2017-07-28).
[17] Docomo：音声認識 API，docomo Developer support（オンライン），入手先 ⟨https://dev.smt.docomo.ne.jp/?p=docs.api.page&api_name=speech_recognition&p_name=api_usage_scenario?⟩（参照 2017-07-28）.
[18] Shazam (online), available from ⟨https://www.shazam.com/⟩ (accessed 2017-07-28).
[19] Amazon: Amazon Echo (online), available from ⟨https://www.amazon.com/dp/B00X4WHP5E⟩ (accessed 2017-07-28).
[20] Google Home (online), available from ⟨https://store.google.com/jp/product/google_home⟩ (accessed 2018-04-13).
[21] Listnr（オンライン），入手先 ⟨https://listnr.cerevo.com/ja/⟩（参照 2017-07-28）.
[22] MagicKnock（オンライン），入手先 ⟨http://magicknock.com/⟩（参照 2017-07-28）.
[23] Yeh, T. and Chang, T.H. and Miller, R.C.: Sikuli: Using GUI Screenshots for Search and Automation, *Proc. UIST'09*, pp.183–192 (2009).
[24] Kato, J., Sakamoto, D. and Igarashi, T.: Picode: Inline Photos Representing Posture Data in Source Code, *Proc. CHI'13*, pp.3097–3100 (2013).
[25] IFTTT (online), available from ⟨https://ifttt.com/⟩ (accessed 2018-04-13).
[26] 栗原一貴，橋本美香：srt.js：映像コンテンツへの IoT 指向拡張プログラム埋め込みフレームワーク，WISS 第 20 回インタラクティブシステムとソフトウェアに関するワークショップ論文集，pp.24–29 (2016).
[27] Rawlinson, H., Segal, N. and Fiala. J.: Meyda: An Audio Feature Extraction Library for the Web Audio API, *Proc. 1st Web Audio Conference* (*WAC*), Paris, France (2015).
[28] dtw (online), available from ⟨https://www.npmjs.com/package/dtw⟩ (accessed 2018-04-13).
[29] Raffel, C., McFee, B., Humphrey, E.J., Salamon, J., Nieto, O., Liang, D. and Ellis, D.P.W.: mir_eval: A Transparent Implementation of Common MIR Metrics, *Proc. 15th International Conference on Music Information Retrieval* (2014).
[30] Claypool, M. and Claypool, K.: Latency and player actions in online games, *Comm. ACM*, Vol.49, No.11, pp.40–45 (2006).
[31] Monster Strike（オンライン），入手先 ⟨http://us.monster-strike.com/⟩（参照 2018-04-13）.
[32] Philips Hue (online), available from ⟨https://www2.meethue.com⟩ (accessed 2018-04-13).
[33] 1-2-switch（オンライン），入手先 ⟨https://www.nintendo.co.jp/switch/aacca/⟩（参照 2018-04-13）.
[34] Nicholas, J., Bullock, J. and Stables, R.: JS-Xtract: A Realtime audio feature extraction library for the web, *International Society for Music Information Retrieval Conference* (2016).
[35] FFTs in JavaScript (online), available from ⟨https://thebreakfastpost.com/2015/10/18/ffts-in-javascript/⟩ (accessed 2018-07-31).
[36] Nishizaki, H.: Data Augmentation and Feature Extraction using Variational Autoencoder for Acoustic Modeling, *Proc. APSIPA Annual Summit and Conference 2017*, P4-3 (2017).
[37] Hamel, P. and Eck, D.: Learning Features from Music Audio with Deep Belief Networks, *Proc. 11th International Society for Music Information Retrieval Conference* (*ISMIR 2010*), pp.339–344 (2010).
[38] Meyer, M., Beutel, J. and Thiele, L.: Unsupervised Feature Learning for Audio Analysis, *Proc. International Conference on Learning Representations* (*ICLR*), workshop track, (2017).
[39] Längkvist, M., Karlsson, L. and Loutfi, A.: A Review of Unsupervised Feature Learning and Deep Learning for Time-series Modeling, *Pattern Recognition Letters*, Vol.42, No.1, pp.11–24 (2014).
[40] Karen, S. and Zisserman, A.: Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556 (2014).
[41] Js-Interpreter (online), available from ⟨https://github.com/NeilFraser/JS-Interpreter⟩ (accessed 2018-04-13).
[42] Alsina-Pagès, R.M., Navarro, J., Alías, F. and Hervás, M.: homeSound: Real-Time Audio Event Detection Based on High Performance Computing for Behaviour and Surveillance Remote Monitoring, *Sensors*, Vol.17, No.4, p.854 (2017).

栗原 一貴 （正会員）

津田塾大学学芸学部情報科学科准教授，博士（情報理工学）．HCI および EC 分野において物議を醸すシステム開発研究を得意とする．著書に『消極性デザイン宣言』がある．2012 年イグノーベル賞等受賞．

植村 あい子 （正会員）

2016 年早稲田大学大学院基幹理工学研究科博士後期課程修了．博士（工学）．2017 年より日本大学文理学部情報科学科助手．音楽情報処理，音響信号処理の研究に従事．

板谷 あかり

津田塾大学大学院理学研究科情報科学専攻修士課程在学．ゲーミフィケーションおよびウェアラブルデバイスを用いた HCI 研究に興味を持つ．

北原 鉄朗 （正会員）

日本大学文理学部情報科学科准教授．博士（情報学）．音響信号処理から作曲・演奏支援まで音メディア情報処理の研究に幅広く従事．第 2 回京都大学総長賞等受賞．

長尾 確 （正会員）

名古屋大学大学院情報学研究科知能システム学専攻教授，博士（工学）．スマートパーソナルモビリティ，ビルディングスケール VR，ディスカッションマイニング等の研究に従事．著書に『ディスカッションを科学する』，『Digital Content Annotation and Transcoding』,『エージェントテクノロジー最前線』等がある．