電子音の認識のための JavaScript ライブラリの開発

栗原一貴[†1]　板谷あかり[†1]　植村あい子[†2]　北原鉄朗[†2]

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上で有意義であるが，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つ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．そこで再生毎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 等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．

Development on a JavaScript Library for Recognizing Machine-generated Sounds

KAZUTAKA KURIHARA[†1]　AKARI ITAYA[†1]
AIKO UEMURA[†2]　TETSURO KITAHARA[†2]

Our daily life is now surrounded by various machine-generated sounds, so it is valuable to establish a way to detect and recognize those synthesized sounds as a trigger to invoke information systems. However, it is still not easy to enable end-user programmers to create custom-built recognizers for each usage scenario through the supervised learning. Therefore, focusing on a feature of synthesized sounds whose auditory deviation is small for each replay, we implement a JavaScript library that detects and recognizes sounds by the traditional pattern-matching algorithm. We quantitatively evaluate its performance, and show its effectiveness by proposing various usage cases including an autoplay system, an augmentation of digital games, and a 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等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等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 の世界に接続する事が可能になり，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 EC 研究コミュニティ，および近年盛んになっている 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遂げている．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挙げている[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・研究機関等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，これらを検出する上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

†1 津田塾大学
Tsuda College
†2 日本大学
Nihon University

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ラム等の特徴量を抽出して，Dynamic Time Warping 等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みなす．「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[3]や Sony MESH[4]などのインターネットサービス・IoT デバイス等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 等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．

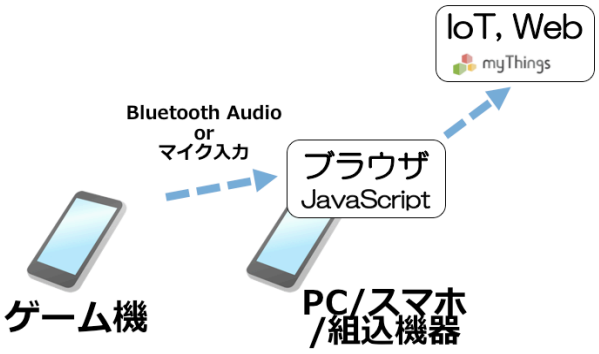

図 1　Picognizer の概要

Figure 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よい検出対象として，任天堂ファミリーコンピュータのようないわゆる 8bit 音源を対象にした認識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．Rui Cui 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 [5]．Aggelos 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 [6]．実際にはガンショット音の他，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．Aki 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[7]．

これらは，spectrum，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．その他にも，音のシーンを階層的にモデル化したり[8]，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するもの[9]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[10,11]．

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 [12]，Google Cloud Speech API [13]，IBM Watson Speech to Text [14]，docomo 音声認識 API [15]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 [16] 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 [17]，Listnr [18] などが挙げれれる．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 機器の連携等を行うものである．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により，様々な応用が可能なオープンシステムとなっている．本研究の提案システムも，追加の認識エンジンとして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 [19]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[20]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．

IFTTT[21]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[22]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．また，Node.js 版の実装も現在進めており，Raspberry Pi 等の小型コンピュータでも動作する予定である．

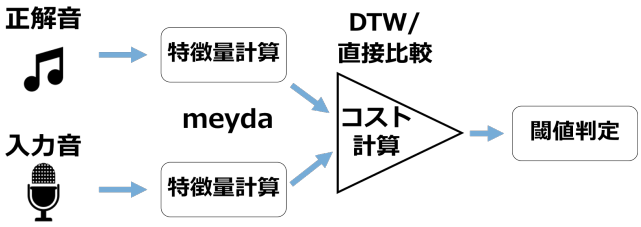

図 2 Picognizer の処理フロー

Figure 2　Workflow of Picognizer.

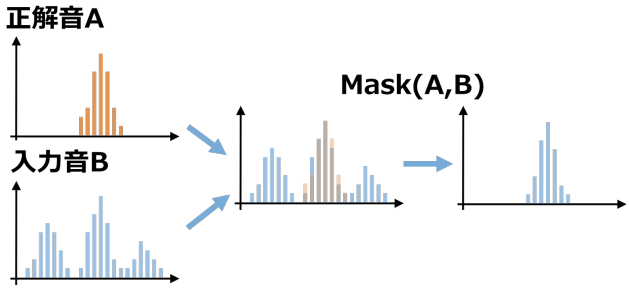

図 3 *Mask(A,B)*の図解．正解音 ***A*** と入力音 ***B*** を直接比較すると，BGM などの正解音以外の音が混入している際に性能が低下する．そこで *Mask(A,B)*を計算し，正解音 ***A*** と *Mask(A,B)*を比較する．

Figure 3　Graphical explanation of *Mask(A,B)*. Naïvely comparing target ***A*** with input ***B*** results in low performance when ***B*** contains noise derived from sources such as background music. To exclude the noise, *Asym_Dist* calculates the Euclidean distance between ***A*** and *Mask(A,B)*.

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[23]によりパワースペクトラム，MFCC 等，17 種類の特徴量の組み合わ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[24]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，検出されたとみ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(target, input) = Dist(target, Mask(target,input))

ここで，*target* は正解音の特徴ベクトル，*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，$Mask(\boldsymbol{A},\boldsymbol{B})$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．

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

これを図解したものが図 3 である．

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．

http://picog.azurewebsites.net/script.html?cri=10&surl=http://picog.azurewebsites.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時に実行する 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 にアクセスした時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．

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	認識対象音源ファイルのURL (.wav or .mp3)
st,et	認識対象音源中で認識に使う箇所の開始時刻と終了時刻
surl	認識時に実行されるJavaScriptファイルのURL
cri	認識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 (default: powerSpectrum)
frame	特徴量抽出の周期 (default: 0.02)
dur	コスト計算の周期 (default: 1.0)

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全ての情報を 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．

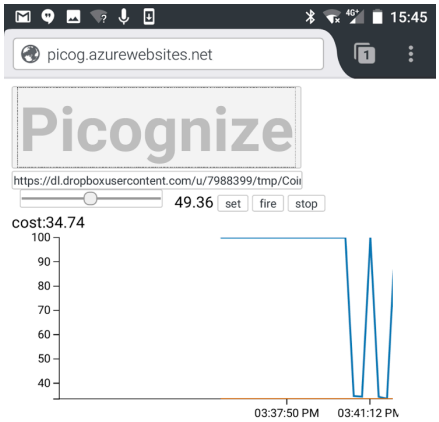

図 4 ブラウザ版の Picognizer の画面

Figure 4 Screenshot of browser-based Picognizer

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もつ，以下のような URL を使用する．

http://picog.azurewebsites.net/trigger.html?cri=33&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同士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．

http://picog.azurewebsites.net/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）．

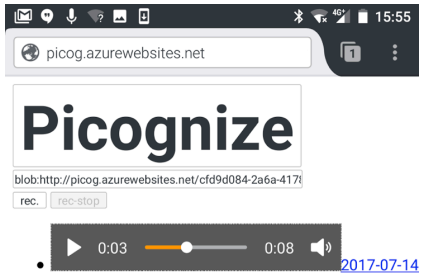

図 5 録音用 UI の画面

Figure 5 Screenshot of browser-based Picognizer with the recording UI.

3.2.2 直接的なコーディング

前節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('./Pico');
var P = new Pico;
var option = {
    windowFunc:"hamming",
    mode:mod,"direct",
    feature:[“mfcc”],
    framesec:0.1,
    duration:1.0
};
P.init(option);
P.recognized(['audio1.mp3','audio2.mp3’],
function(cost){
    //do something with an array of cost
});
```

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，認識したい電子音の再生環境に依存する．図 6 から 9 に様々なパターンにおける音声入力方法を列挙する．認識したい電子音が 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（ステレオミキサーなどと呼ばれる）を準備すれば，雑音の影響なく同一 PC の Web ブラウザ上の Picognizer で認識可能となる（図 6）．認識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等で Picognizer 動作機器に入力することで雑音の影響を軽減させる．図 7 はモニター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ーを用いて無線接続した例である．これ以外の再生環境の場合は Picognizer 動作機器に接続されている物理的なマイクにより，実空間に発せられている音声を集音する（図 9）．この際は環境音等のノイズの影響が大きくなるので，音声の前処理や閾値調整を詳細に行うことになる．

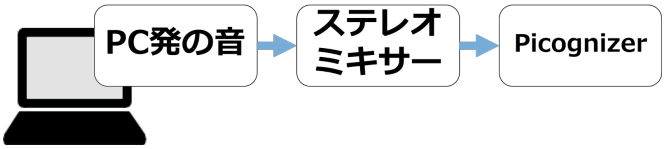

図 6 仮想的マイクデバイスによる接続

Figure 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

Figure 7 Connection with an audio cable.

図 8 Bluetooth Audio による無線接続

Figure 8 Connection with a wireless Bluetooth audio.

図 9 内蔵スピーカと内蔵マイクによる接続

Figure 9 Connection with a physical speaker and a microphone through the aerial sound transmission.

4. 基礎性能評価

本論文では研究の初期段階として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よい認識対象として，いわゆる 8bit 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．

(1) **テストデータセット**

本実験では 4 つの実験を行った. 実験番号が進むにつれて, 入力音源の複雑性が増す傾向をもつように設定している. 各実験の詳細は以下の通りである.

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), 1up きのこ取得音（1up）の 3 種の効果音を選択した. 35 秒のサンプリング周波数 22050Hz 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 0dB, 5dB 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全てについて, オンセット（再生開始）時刻のラベル付けを全て手動で行った.

(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よ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, 複数のパラメータの組み合わ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 40ms 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[25]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±50ms 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.

(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 2～5 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, 5dB と 0dB は実験 3 の結果に対応する. それぞれ, 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）の影響を比較した.

結果から, *Asym_Dist* と直接比較の組み合わせが最もよい検出性能だったことがわかる.

表 2 実験 1・実験 2 における「ユークリッド距離および 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 における「ユークリッド距離および直接比較」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 における「*Asym_Dist* および 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 における「*Asym_Dist* および直接比較」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

次に，表 6〜9 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，5dB と 0dB 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．

表 6 実験 3・実験 4 における「ユークリッド距離および 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

表 7 実験 3・実験 4 における「ユークリッド距離および直接比較」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 における「*Asym_Dist* および 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 における「*Asym_Dist* および直接比較」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

結果から，再び *Asym_Dist* と直接比較の組み合わせが最もよい検出性能だったことがわかる．

(4) **考察**

実験を通して *Asym_Dist* と直接比較の組み合わせは，F 値が多くの条件下で 1.0 となり，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．

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言える．*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，更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，再生毎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，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．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．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．

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．

5.1 ゲームの拡張

5.1.1 Toolification of Games

図 10 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[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上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，何らかの社会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．

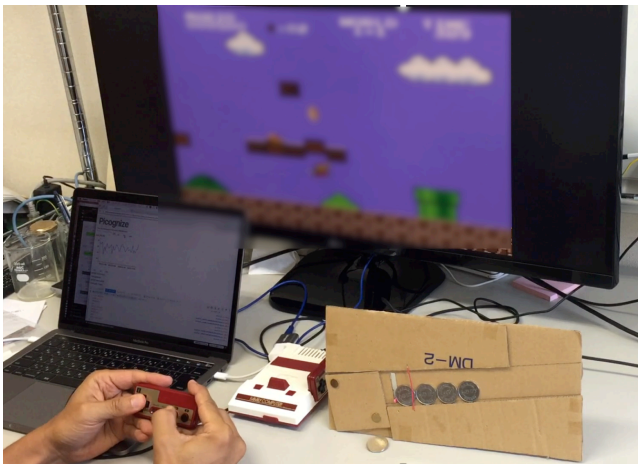

図 10 スーパーマリオブラザーズを用いた Toolification of Games 事例，「Coins For Two」．コイン取得時の効果音を検出し，コインサーバーから硬貨が排出される．

Figure 10 An example of Toolification of Games with Super Mario Brothers. Picognizer detects the sound effects of gaining a coin to eject a real coin.

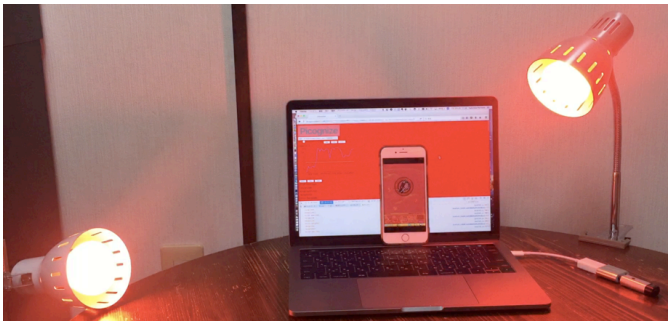

図 11 モンストストライクと Hue との連携の例．ボス敵登場時の警告音を検出し，Hue の色彩が変化する．

Figure 11 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.

5.1.2 Hue との連携

図 11 は，スマートフォンのゲーム「モンスターストライク[26]」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[27]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．

5.2 ゲームの自動操作

図 12 は，Nintenodo Switch のゲーム「1-2-Switch[28]」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．

図 12 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．
Figure 12 An example of an autoplay system for QuickDraw, a game for Nintendo Switch. The game controller is manipulated physically by an IoT motor Webmo.

6. 課題と今後の展望

(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（距離）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，およびニーズにもとづき，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．

(2) より複雑な音響特性をもつ電子音への拡張

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 8bit 音源を対象にし，環境音ノイズのない状況下での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，その全てに対し高性能の検出器を構成するのは容易ではない．研究の第一歩として，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よい対象として，これを選定した．前節で例示したように，より複雑・高度な音源への適用も現状のプロトタイプである程度可能であるが，それらへのチューニングや性能検証は今後の課題である．

(3) ディジタルゲーム音以外の電子音認識への応用

本研究では Picognizer のパフォーマンス評価を，ディジタルゲーム音認識に特化させて行った．しかし，ディジタルゲーム音以外にも，エンドユーザプログラマが認識したい電子音は多岐に渡っている．現状でも詳細な性能評価は行っていないが，そのような応用は可能である．たとえば図 13 は，日本のマクドナルドでよく用いられるフライドポテトの調理終了時刻を通知する電子音を検出し，スマートフォンに「揚げたてである」という push 通知を行う例である．電子音検出には 2.1 節で挙げたような関連研究があるが，それらとの性能比較や使い勝手等を検証するのは今後の課題である．

図 13　フライドポテトの調理終了時刻通知音検出の例．スマートフォンのマイクにより調理終了時刻通知音が検出されると，slack に push 通知される．
Figure 13 An example of the detection of synthesized sounds in a nongame context. An alert sound from McDonald's French fry potato fryer is detected and a push notification is delivered to the user.

(4) **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）[29] 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．

(5) **教師付き学習に向けて**

本研究では，ユーザの身近にある電子音について，教師付き学習を用いずテンプレートマッチする手法について扱ったが，構築したブラウザ版 Picognizer が広く用いられるようになれば，ユーザにとって関心のある様々な電子音の検出に関するデータをサーバ側に蓄積可能である．これに深層学習等の教師付き学習技術を用いることにより，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等の電子音検出・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し，また環境音ノイズ下でマイクから集音した音に対する精度の向上などを検証する．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 の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) G. Hinton and L. Deng and D. Yu and G. E. Dahl and A. r. Mohamed and N. Jaitly and A. Senior and V. Vanhoucke and P. Nguyen and T. N. Sainath and B. Kingsbury: 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) myThings, https://mythings.yahoo.co.jp/
4) Sony MESH, http://meshprj.com/
5) Rui Cai, Lie Lu, Hong-Jiang Zhang, and Lian-Hong Cai: Highlight Sound Effects Detection in Audio Stream, Proceedings of IEEE International Conference on Multimedia and Expo (ICME 2003), vol. III, pp.37-40 (2003).
6) Aggelos Pikrakis, Tehodors Giannakopoulos, and Sergious Theodoridis: Gunshot Detection in Audio Streams from Movies by Means of Dynamic Programming and Bayesian Networks, Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2008), pp.21-24 (2008).
7) Aki Harma, Martin F. McKinney, and Janto Skowronek: Automatic Surveillance of the Acoustic Activity in Our Living Environment, Proceedings of IEEE International Conference on Multimedia and Expo (ICME 2005), (2005).
8) Keisuke Imoto and Suehiro Shimauchi: Acoustic Event Analysis based on Hierarchical Generative Model of Acoustic Event Sequence, IEICE Transactions on Information and Systems, Vol. E990D, No.10, pp.2539-2439 (2016).
9) Xuang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori: Sparse Representation based on a Bag of Spectral Exemplars for Acoustic Event Detection, Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2014), pp.6255-6259 (2014).
10) Yun Wang, Leonardo Neves, and Florian Metze: Audio-based Multimedia Event Detection Using Deep Recurrent Neural Networks, Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2016), (2016).
11) Emre Cakir, Giambattista Parascandolo, Toni Heittola, Heikki Huttunen, and Tuomas Virtanen: Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection, IEEE/ACM Transactions on Audio, Speech, and Language Processing, Vol.25, No.6, pp.1291-1303 (2017).
12) Bing Speech API, https://azure.microsoft.com/ja-jp/services/cognitive-services/speech/
13) Google Cloud Speech API, https://cloud.google.com/speech/
14) IBM Watson Speech to Text, https://www.ibm.com/watson/developercloud/speech-to-text.html
15) Docomo 音声認識 API, https://dev.smt.docomo.ne.jp/?p=docs.api.page&api_name=speech_recognition&p_name=api_usage_scenario
16) Shazam, https://www.shazam.com/
17) Amazon Echo, https://www.amazon.com/dp/B00X4WHP5E
18) Listnr, https://listnr.cerevo.com/ja/
19) MagicKnock, http://magicknock.com/
20) Yeh, Tom and Chang, Tsung-Hsiang and Miller, Robert C., Sikuli: Using GUI Screenshots for Search and Automation, In Proc. of UIST'09, pp.183-192 (2009).
21) IFTTT, https://ifttt.com/
22) 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).
23) Hugh Rawlinson, Nevo Segal, and Jakub Fiala. Meyda: an Audio Feature Extraction Library for the Web Audio API. In The 1st Web Audio Conference (WAC). Paris, France (2015).
24) dtw, https://www.npmjs.com/package/dtw
25) C. Raffel, B. McFee, E. J. Humphrey, J. Salamon, O. Nieto, D. Liang, and D. P. W. Ellis: mir_eval: A Transparent Implementation of Common MIR Metrics, Proceedings of the 15th International Conference on Music Information Retrieval (2014).
26) Monster Strike, http://us.monster-strike.com/
27) Philips Hue, https://www2.meethue.com
28) 1-2-switch, https://www.nintendo.co.jp/switch/aacca/
29) Js-Interpreter, https://github.com/NeilFraser/JS-Interpreter