ながらデバッグマスク：口の表情を入力として ソースコードの読み上げとバグの検出・記録が可能な仕組み

塙克樹[†1]　橋田朋子[†1]

概要：一般的にプログラミングは，モニタの前でソースコードを視認しながら，キーボードを打って行う．しかし従来のプログラミング方法は手を動かしての入力とモニタからの視覚情報の提示を必要とする．本稿では，この問題を解決するために，発声の伴わない口の表情を入力とし聴覚情報の提示でデバッグを行うシステムを提案する．外部に音声に漏らすことなく使用可能な口の表情を 5 種類マスク型デバイスで識別して，ソースコードの読み上げとバグの検出および記録を操作するシステムを実装した．マスク型デバイスの精度実験とシステムのユーザスタディを行った．

Debugging Mask: A system that can read source code and detect and record bugs by using oral shapes as input

KATSUKI HANAWA[†1]　TOMOKO HASHIDA[†1]

Abstract: We generally program by using a keyboard while viewing source code on a monitor. However, conventional programming methods require manual input and the presentation of visual information on the monitor. In this paper, we propose a system in which we can debug by the presentation of audio information with the input of oral shapes without utterance. We implement a system that manipulates the reading of source code line-by-line and that detects and records bugs by using a mask device to acquire five kinds of non-voiced oral shapes. We evaluate the mask device and conduct a user study of this system.

1. はじめに

2020 年から小学校でプログラミング教育が義務化される．このことからも見られるように，プログラミングは一部のエンジニアだけでなく，多くの一般的な人が行うものになりつつある．一方でプログラミングを行う環境は，依然としてモニタを注視しながら，キーボードとマウスを用いる従来の方法に限定されている．そのため，モニタを注視できない環境や手を動かすことのできない環境ではプログラミングを行うことができない．筆者らはプログラミングの必要性が増す中で，プログラミングを行うことが可能な環境を拡げていくことが望ましいと考える．

この課題に対する先駆的な試みとして音声でプログラミングを行う方法が挙げられるが[1]，公共空間でこの方法を用いることができない．公共空間において一人で話すことは，他人に不快感を覚えさせ，気まずい行為とされているためである．そこで，本研究では”公共空間での歩行時”のように従来プログラミングを行うことが難しかったシチュエーションにおいて，周りに迷惑をかけず簡易なプログラミングを行うことを目指す．この目的を達成するためにプログラミングの中でも，確認作業に近いデバッグに注目する．ソースコードをユーザにヘッドフォンで聴覚情報として提示し，ユーザの発声を伴わない口の表情を識別し入力として用いることで，デバッグを行えるシステムを提案する．今回は市販のマスクの下にフォトリフレクタモジュールを複数配置することで，発声を伴わない 5 種類の口の表情の識別を実現する．

本稿では提案システムの詳細と，口の表情の識別精度および，聴覚情報の提示によるデバッグに関するユーザスタディの結果を報告する．

図 1　システムを用いて口の表情を識別しソースコード読み上げを操作することで i が j になっているバグを聴覚情報の提示で発見している様子

Figure 1　The situation of finding a bug of i for j, manipulating the reading source code with this system as he input of oral shapes

†1 早稲田大学
Waseda University.

2. 関連研究

口の表情を識別する研究の代表例として, Ando らはイヤフォン型のウェアラブルデバイスで口の表情を含む顔全体の動きを検出している[2]. 顔を動かすときに耳の中で生じる左右の気圧の差をイヤフォンに組み込んだ気圧センサで計測し, 機械学習によって 11 種類の顔の動きを 87.6%の精度で識別している. また Lyons は MouthType で手と口の表情を用いて, テキスト入力できるシステムを提案している[3]. これらの研究は口の表情を識別し入力として用いているが, プログラミングに応用されるものではない.

一方, プログラミングのスタイルを拡張させる試みもある. 全盲のエンジニアが画面を見ずに, スクリーンリーダの音声フィードバックとブラインドタッチのみでプログラミングを行う例がある[4]. Wnger らは Myna で取得した音声コマンドによって, Scratch と呼ばれる初心者向けビジュアル言語のプログラミングを行うことを可能にしている[5]. ただし, これらの事例や研究では音声認識を用いているため, 公共空間での使用が難しい. 本研究は発声を伴わない口の表情を識別し入力として用いることで, 周囲の他者に気づかれずにプログラミング作業を行うことが可能である点が異なる.

3. システム

提案システムの構成を図 2 に示す. 提案システムはユーザの口の表情を検出する表情検出部と機械学習を用いた口の表情を識別する表情識別部, 口の表情に対応してソースコードの読み上げとバグが存在する行の記録を行う出力部から構成される.

3.1 表情検出部

口の表情を識別するための 14 個のフォトリフレクタモジュールとマイコン（Arduino Uno）, AD コンバータから構成されるマスク型デバイスを製作した. このデバイスはマスクの下に設置し, 口を覆う形になっている. 各フォトリフレクタモジュールと顔との距離をリアルタイムに計測することで, 口の周りの動きを検出している. 4 個ずつの計測している. フォトリフレクタには TPR-105F を用いている. また Arduino Uno のセンサピンの不足を補うために AD コンバータ（MCP3208-CI/P）を用いた. 表情を識別するために, 各フォトリフレクタモジュールから取得した 14 個のセンサデータをシリアル通信で PC に送信している.

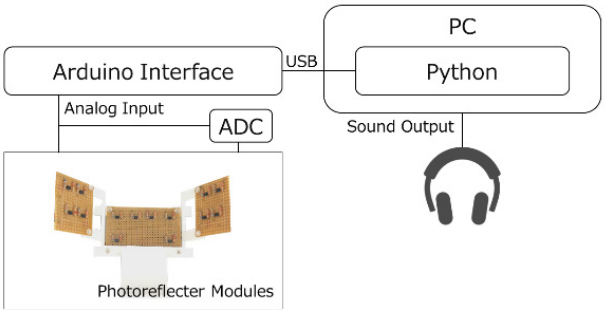

図 2 システム構成
Figure 2 System configuration

3.2 表情識別部

本システムでは無表情（Normal）, 笑顔（Smile）, 驚き顔（Surprised）, 下顎を右に動かした口の表情（Right）, 下顎を左に動かした口の表情（Left）の 5 種類の口の表情を識別した. 識別を行った口の表情を図 3 に示す. これらは個人間での理解と共有が簡易な口の表情である. 5 種類の口の表情を識別するための機械学習には Python を用いた. Python のバージョンは 3.6.0 で, 機械学習に用いたモジュールには scikit-learn である. 学習器には線形カーネルの Support Vector Machine（SVM）を用いた. Python のプログラムを実行するために用いた PC は Microsoft 社の Surface Pro3 である. ユーザが本システムを用いるための学習は 10 試行繰り返すので, ユーザ毎に合計で 50 のデータが得られる. 収集したデータを用いて, 5 種類の口の表情を識別するための識別モデルを個別に生成した. 学習によって識別された口の表情を, ソースコードの読み上げとバグが存在する行の記録を行う出力部に OSC 通信で送信している.

3.3 出力部

OSC 通信によって送信された口の表情に応じて, ソースコードの読み上げもしくはバグが存在すると判断した行の記録を行った. 読み上げに用いたプログラミング言語は Python でモジュールは win32com である. Python で読み込んだソースコードを, 識別部と同様の PC で読み上げる. 下顎を右に動かした口の表情でソースコードを一行下に移動して読み上げ, 下顎を左に動かした口の表情でソースコードを一行上に移動して読み上げる. 笑顔で同じ行を再び読み上げる. 驚き顔でバグがあるとユーザが判断したソースコードの行を記録する.

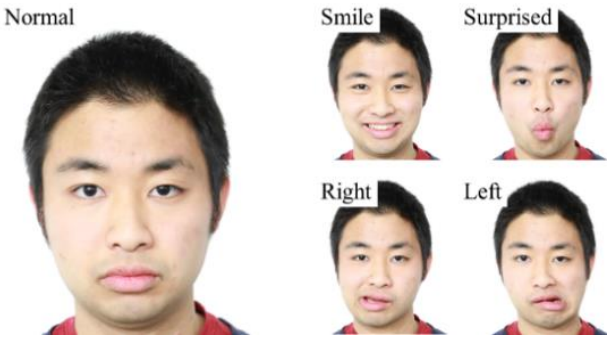

図 3 認識セット一覧
Figure 3 Recognition sets

4. 精度実験

製作したマスク型デバイスを用いた口の表情の識別精度を評価するために, 実験を行った.

4.1 実験の手順

実験には 4 人の被験者（20 代の男性 3 人，女性 1 人）が参加した．被験者はマスク型デバイス，固定用のゴム，紙マスクを重ねて装着し，室内環境で椅子に座った姿勢で机の上に置かれたモニタと正対する．認識セットに含まれる 5 種類の口の表情について練習をしたのち，実験データの収集を開始した．

認識セット中からランダムに選ばれた 1 つの口の表情を表す画像を提示し，被験者は提示された口の表情を行う．1 つの認識セットのデータ収集が終わったのちに，次の認識セットでデータの収集を行う．すべての口の表情で 10 セットずつデータの計測が完了した時点で終了とした．

4.2 実験の結果

分析には 10 分割交差検証を用いており，5 種類の口の表情での被験者毎の認識率を算出した．具体的には，ある 1 人の被験者から収集したセンサデータのうち 9 回分のデータを教師データとして学習し，その口の表情の識別モデルを生成する．その被験者の残り 1 回分のデータをテストデータとして，生成したモデルの精度を算出する．これを 1 セットずつずらして 10 回全ての組み合わせに対して同様の操作を行い，算出された結果の平均値を求めた．この結果をその被験者の認識率とし，4 人の被験者全員の認識率の混合行列を求めた．実験の結果，平均認識率は 96.0%（SD=4.5）であった．被験者毎の認識率の混合行列を表 1 に示す．この結果から被験者内であれば，マスク型デバイスを用いて，5 種類の口の表情の識別を行うことができる精度があると判断した．

表 1　混合行列:口の表情の認識率
Table 1　Confusion matrix of Oral shapes

		Oral Shape				
	Actual/Predict	N	SM	SU	R	L
Oral Shape	Normal(N)	100	0	0	0	0
	Smile(SM)	2.5	97.5	0	0	0
	Surprised(SU)	0	0	92.5	7.5	0
	Right(R)	5	5	0	90	0
	Left(L)	0	0	0	0	100

5. ユーザスタディ

提案システムではソースコードを聴覚情報の提示で行う．本実験では，聴覚情報でソースコードを提示して実際にバグを発見することができるのかを明らかにする．具体的には，その時間的・内容的な精度は一般的な視覚情報でソースコードを提示するときに比べてどの程度であるかを調べる．一般的には視覚情報で提示を行うときに比べて聴覚情報で提示を行うときは低い精度が予想されるが，例えば視覚的に類似しているバグの場合には聴覚情報での提示がより効果的な可能性もある．この点についても検討する．

5.1 方法

被験者はプログラミング言語 Processing を初めて学んでから半年以上の 20 代の男性 4 人である．被験者の課題は，歩行をしながら Processing のソースコードを確認・記録するデバッグである．ここで，デバッグの手法は聴覚と視覚の 2 つの水準を設けた．バグは単純なスペルミスであるが，このバグの種類にも，視覚的に区別しづらい文字とランダムに生成した文字の 2 つの水準を設けた．被験者は 2 要因 2 水準で合計 4 セットに関して，歩行しながらデバッグを行った．4 セットの順序は被験者ごとにランダムにしている．ユーザスタディでの判断尺度はすべてのバグを発見したと被験者が判断した時間[s]とバグの発見率[%]である．最後に自由記述によるアンケートを行った．

5.2 デバッグの手法

デバッグの手法のうち聴覚条件では，ノイズキャンセリングヘッドフォン（BOSE QuietComfort25）でソースコードを聴覚情報として提示し，被験者はゲーミングコントローラ（ACGAM R1）をコマンド入力することでソースコードの一行読み上げとバグの記録を行った．本条件では，聴覚情報でのソースコード提示の聴認性に重点を置くため，ソースコードの読み上げとバグの記録を操作するものとしてマスク型デバイスを用いず，手元を見ずに簡易に操作できるゲーミングコントローラを用いた．視覚条件では被験者はタブレット状態にした Surface Pro3 を用いた．Windows に標準搭載されたテキストエディタでソースコードを視覚情報として提示し，被験者はこの PC 上でソースコードの確認を行った．

5.3 ソースコードとバグの種類

ソースコードは Processing のサンプルコードから 20 行程度のものを用いる．ソースコードに組み込むバグの種類は上述のように 2 種類ある．1 種類は視覚的に区別しづらい類似文字を 3 文字入れ替えたものである（以下，類似文字条件）．類似文字は，プログラミングで用いられる文字，記号のうち Python 上の OpenCV で類似度が高いと判定されたものに関して，簡易なユーザテストを行って決定した．フォントは視認性が高く，エンジニアに人気と言われる Ricty Diminished である．類似文字を入れ替えた例を図 4 に示す．バグとして”l”が”l”に変更されている．ただし，類似していると判断されても c と C のような聴覚的に区別が不可能な文字は排除した．もう 1 種類はランダムにソースコードの 3 文字を別の文字もしくは記号に入れ替えたものである（以下，ランダム条件）．ランダムに文字を入れ替えた例を図 5 に示す．バグとして”)”が”X”に変更されている．水準ごとに 2 つのソースコードを用意したため，用いたソースコードは合計 2×2 の 4 種類である．

```
translate(width/2+30, height/2, 0);
trans1ate(width/2+30, height/2, 0);
```

図 4　類似文字を入れ替えたスペルミスのバグ
（上：元の命令文，下：バグを含めた命令文）

Figure 4　Bug of a spelling mistake on visually similar words
(top : original sentence, bottom : sentence with a bug)

```
noStroke();
noStroke(X;
```

図 5　ランダムに文字を入れ替えたスペルミスのバグ
（上：元の命令文，下：バグを含めた命令文）

Figure 5　Bug of a spelling mistake on random words
(top : original sentence, bottom : sentence with a bug)

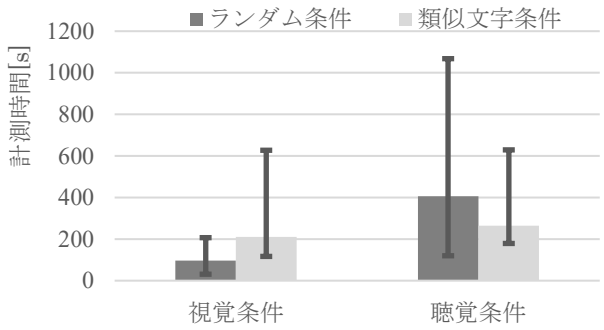

図 6　バグをすべて発見したと判断した時間

Figure 6　Time to determine completion

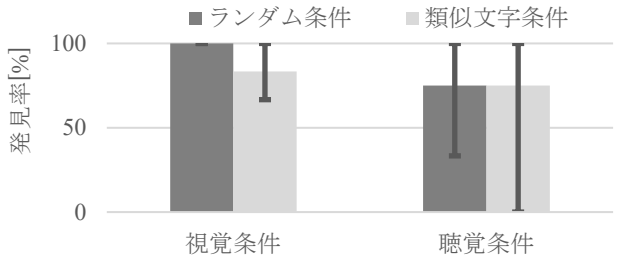

図 7　バグの発見率

Figure 7　Percentage of bugs found

5.4 ユーザスタディの結果と考察

すべてのバグを発見したと判断した時間[s]とバグの発見率[%]の結果をそれぞれ図 6，図 7 に示す．図 6 から視覚条件でのバグ発見に要する時間は 154s で聴覚条件でのバグ発見に要する時間は 334s がわかる．図 7 から視覚条件のうちランダム条件でのバグ発見率が 100%であるのに対して類似文字条件では 83.3%である．聴覚条件ではランダム条件，類似文字条件ともにバグ発見率は 75.0%である．これらの結果から，バグの発見に時間を要するものの，視覚的な類似性によらず聴覚でのバグ発見が行えることが示唆される．次にユーザアンケートの結果を下記に示す．

- 視覚では画面を俯瞰できるが，聴覚では全体のうち一行の読み上げを聞くので，得意不得意がある
- 聴覚によるバグ発見では歩行をスムーズに進めながら行うことができた．
- 慣れによる影響が強く出ると考えている．
- プログラミングは孤独な作業であるが，本システムによって温かみのあるプログラミングが期待できる．

ユーザアンケートから，聴覚のメリットを活かすことで新しいバグ発見の形が期待できることが示唆された．

6. まとめと今後の展望

本論文では，公共空間で歩行時のように本来プログラミングを行えない環境で口の表情を識別し入力として用いることで，ソースコードの聴覚情報での提示とバグの発見が可能なシステムを提案した．また，口の表情を識別するためのマスク型デバイスを作成し精度を評価した．口の表情の高い識別精度を得た．最後に歩行時でも聴覚によってバグ発見が可能であるかを調べるためのユーザスタディを行った．視覚的なバグ発見のしやすさに関わらず，聴覚によってバグを発見できることが示された．

聴覚情報の提示でバグの発見を行う際には，違う変数だが音が同じである場合に聴覚的に弁別できない問題や，人によって読み方の異なる語句の読み上げ方の問題がある．

今後は口を使う上での入力の少なさと聴覚による得意な領域を考慮した新しいプログラミングの形を検討していく．一例として，読み上げに効果音を加えるなどの工夫で聴覚情報の提示によるデバッグの効率性を高めることが期待される．

参考文献

[1] “Using Python to Code by Voice”. https://www.youtube.com/watch?v=8SkdfdXWYaI, (参照 2017-12-12).

[2] Toshiyuki Ando, YukiKubo, Buntarou Shizuki, and Shin Takahashi. 2017. CanalSense: Face-Related Movement Recognition System based on Sensing Air Pressure in Ear Canals. .In Proceedings of the 30th Annual Symposium on User Interface Software and Technology (UIST '17). ACM, Quebec City, QC, Canada, 22-25.

[3] Michael J. Lyons, Chi-Ho Chan, and Nobuji Tetsutani. 2004. MouthType: Text Entry by Hand and Mouth. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04). ACM, Vienna, Austria, 24-29.

[4] “目が見えなくてもプログラミングできるよ”. https://qiita.com/moutend/items/2696139d0b96805ea415, (参照 2017-12-12).

[5] Amber Wagner, Ramaraju Rudraraju, Srinivasa Datla, Avishek Banerjee, Mandar Sudame, and Jeff Gray. 2012. Programming by Voice: A Hands-Free Approach for Motorically Challenged Children. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12). ACM, Austin, TX, USA. 2087-2092.