片手でのキーボードの利用の問題点について

平成 24 年 2 月 16 日

情報電子工学科
川上 茂浩

目 次

概要

過去の卒業研究で一時的な片手キーボードの利用に関する研究があった。これは、UNIX 上の X Window System の xmodmap という仕組みを使い、キーの配置を変え、どのようにしてキーボードを片手で扱えるように変更するのか、そしてそれを実際にキーボードに割り当て、実験し考察した研究である。しかし、この研究にはいくつか問題点が残されていた。それは、記号を全く考慮していなかったこと、 BackSpace キーを押し続けることによる削除が不可能で、一文字毎に削除しなければならなかったことである。今回は 2 つの問題の解決を目指すとともに、過去の研究を改良したキー配列、さらに全く新しいキー配列を提案し、過去の研究で考察された配列と比較するための実験を行なった。

1　はじめに

現在、コンピュータで使われているキーボードは、両手前提のものである。しかし、以下のような理由で片手でしか入力できない場合がある。

- 怪我や腱鞘炎などで片手が使えない時
- 事故などで片手がなくなった時
- 片手で何かをしている時
- 小型のパソコンを片手で持っている時

X Window System という UNIX 系 OS で利用されるグラフィカルユーザインターフェース (GUI) 環境を使用すればキーボードのカスタマイズは容易に行なえるので、それを用いて本研究ではキーボードを片手で、しかも片手の範囲で全ての文字キーを打てるようキーボードをカスタマイズすることを考える。

過去の研究 [1)] では左手で使うことを考慮した「文字列を折り畳んだ形で左側に合わせたもの」、「文字列をスライドさせた形で左側に合わせたもの」と右手で使うことを考慮した「文字列をスライドさせた形で右側に合わせたもの」、「テンキーでアルファベットが打てるようにしたもの」を考察している。

前述したような理由によって、片手が使えなくなったり、塞がることによって、片手でキーボードを使用することになる。しかし、以下のような問題が発生する。

- キー範囲が広く、疲れやすく、時間がかかる
- Shift と同時押しする時、中心のキーは押しにくい
- 左手側のキーでタイプミスをすると BackSpace が遠い

これらは、キーボードを利用している人にとっては、片手が使えないというのは、不便であることを意味している。だからといって、一時的な片手利用者にとっては、市販されている片手キーボードを購入する必要もないであろう。

そこで、今回の研究ではこれらの問題を解消しつつ、過去の研究で考察されたテンキーは除外し、以下の状態を対象とする。

- 怪我などで利き手の右手がつかえない
- ローマ字で日本語文章を入力しなければいけない

実際にローマ字で日本語文章を入力する場合は、文章中にアルファベットが出て来たら「変換」を押し、大文字なら「CapsLock」を押すこととする。

2 X Window System

2.1 概要

本研究では、過去の研究でも使用された X Window System というものを使う。これは、UNIX 系 OS で利用されるグラフィカルユーザインターフェース (GUI) 環境である。

元来 UNIX は文字ベースの操作環境しか利用できなかったが、Athena Widget Project が中心となって、マサチューセッツ工科大学 (MIT) で開発された。現在では業界団体の The Open Group が開発を行なっている。ほとんどの UNIX 系 OS に標準で搭載されており、他の OS にも移植されている。クライアントがサーバの機能を呼び出して使う分散構造になっており、アプリケーションソフトや OS の処理はクライアントが、画面表示や入出力はサーバが行なう。

また、見栄えや操作方法を定義するウィンドウマネージャと呼ばれるソフトウェアには、Windows 風のものや、Mac OS 風のものなど様々な種類があり、ユーザが好きなものを選ぶことが出来る。

ちなみに、X Window System は、以下のような理由で広く普及した。

- ソースコードが無料で公開されている
- システム中でハードウェアに依存する部分が明確に切り分けられていので、UNIX ワークステーションであれば、容易に移植することが可能

2.2 特徴

以下に特徴を列挙する。

- サーバクライアントモデルに基づくシステム構成
- ネットワーク透過性
- 数多くのプラットフォーム (ワークステーション環境) で動作可能
- 豊富なカスタマイズ機能
- 種々の GUI スタイルをサポート
- 日本語入力を含む国際化機能

本研究では上から、4 番目の豊富なカスタマイズ機能のうち、マッピング (割り付け) を変更する機能を使用する。マッピングで変更できるものは、以下の通り。

- 文字キー
- 修飾キー
- マウスボタン

この内、文字キーと修飾キーをカスタマイズする。

3 キーボードのカスタマイズ

3.1 xmodmap

新しいワークステーションを使う場合には、これまで使っていたキーボードのキー配列と微妙に違っていたりすると、それに慣れるまでのストレスに耐えなくてはならない。しかし、このような悩みは特徴で挙げた豊富なカスタマイズ機能によって解消できる。キーボードのキー配列をカスタマイズするためのクライアントが「xmodmap」である。

「xmodmap」とは、X 上でキーマップや、ポインタボタンの割り当てを変更するユーティリティ。キーやマウスボタンの配列を専用の構文を用いて変更でき、その気になればすべてのキー配列を変更することができる。クライアントのアプリケーション上で、イベントのキーコードをキーシンボルへ変換する時に使われるキーボード modifier マップと keymap テーブルを、表示したり編集する時に使われる。

主なオプションを以下に記す。

- -help

  xmodmap オプションの簡単な説明を表示。

- -grammer

  変更式の簡単な説明を表示。

- -verbose

  xmodmap 実行時のログ情報を表示。

- -n

  キー・マッピングの変更を実際に実施せず、その変更情報を表示。

- -e ’ 変更式’

  コマンド行で変更式を指定。

- -pm

  現在の修飾キーのマッピングのリストを標準出力に表示。

- -pke

  現在のキー・マッピングのリストを標準出力に表示。

3.2 文字キーの変更式

X は、さまざまなワークステーションのキーボードでも動くことを想定して作られているため、以下の 2 階層のモデルを採用している。

- キーコード

  キーを物理的に認識するためのモデル。これは、キーボード上のすべてのキーに、キーの持つ意味とは関係なく機械的に番号を割り当てたもの。メーカーによって異なる。例：keycode 11 = a、keycode 37 = 1 など。

- キーシンボル

  キーボードに依存しない論理的にキーを認識するためのモデル。これは、キーに記された文字列やキーの持つ意味をもとに決めた番号であり、すべてのサーバで共通な番号である。

あるキーが押された場合に発生するキーコードには、そのキートップに印刷された文字に対応するキーシンボルが割り当てられている。

キーのマッピングを変えるためには、tabel1 のような 2 つの構文のうちいずれかを用いる。

変更式	意味
keycode KEYCODE = KEYSYM	キーシンボルを キーコードに割り当てる
keysym KEYSYM = KEYSYM	キーシンボルを 別のキーシンボルに置き換える

Table 1 文字キーの変更式

マッピングの変更の方法は、以下の 2 通り。

- -e オプションを使用してコマンド行で変更
- 変更内容をファイルに書いて、ファイルを読み込む (4.6 参照)

例として、「a」のキーを「b」に変えたい場合、キーコードで変更する場合は、「a」のキーコードが 11 なら、

```
% xmodmap -e 'keycode 11 = b'
```

とし、キーシンボルで変更する場合は、

```
% xmodmap -e 'keysym a = b'
```

とすると「a」を押した時に「b」が入力される。この時、「b」を押しても「b」のままで変更されない。元に戻すにはキーコードでは、

```
% xmodmap -e 'keycode 11 = a'
```

とすれば元に戻る。キーシンボルでは、一度

```
% xmodmap -e 'keysym a = b'
```

とするとイコールの左側の「a」が無くなってしまい、そのキーは「b」扱いになってしまう。その状態で、

```
% xmodmap -e 'keysym b = a'
```

とすると元々「a」「b」だったキーは両方とも「a」になってしまうので、keysym だけでは元に戻せなくなってしまう。よって変更するときも、戻すときも、keycode を使用した方がよい。また、イコールの右側は文字を最大 4 つまで指定でき、Shift キーが押された場合に二番目の文字が出力される。デフォルトの状態で

```
% xmodmap -e 'keycode 11 = b c'
```

または、

```
% xmodmap -e 'keysym a = b c'
```

とすれば、Shift キーを押しながら「a」を押すと「c」が出力されるようになる。

3.3 修飾キーの変更式

他のキーと一緒に押すキーを修飾キーという。X Window では、このような修飾キー名として shift、lock、control、mod1、mod2、mod3、mod4、mod5 の 8 つをサポートしている。MODIFIER には、修飾キー名を指定する。修飾キーに、キーを登録したり削除するには Tabel2 を参照。

変更式	意味
clear MODIFIER	修飾キーをクリア
add MODIFIER = KEYSYM	指定したキーを修飾キーに登録
remove MODIFIER = KEYSYM	指定したキーを修飾キーから削除

Table 2 修飾キーの変更式

例として、右側の Shift を Control にするには、

```
% xmodmap -e 'remove shift = Shift_R'
% xmodmap -e 'add control = Shift_R'
```

とする。この変更式は、 最初の操作で Shift_R を shift から削除し、次の操作で control に新たに登録し直している。削除操作を行なわないと shift と control の両方に Shift_R が登録され、いずれかの登録が無視される。

3.4 ファイルで指定

構文をいちいちコマンドラインから指定するのは面倒である。そこで、過去の研究でも使われたファイルで一度に指定する方法を記述する。

例 1 は文字キーの変更式、例 2 は、MODIFIER キーの変更式である。どちらも同じファイル内 (abc.xmodmap) の構文である。!!で始まる行はコメント行である。

例 1:文字キーの変更式

```
!!キーコードの元の状態は
!!keycode 40 = 4 dollar kana_U kana_u
!!keycode 45 = 9 parenright kana_YO kana_yo
keycode 40 = 4 9 kana_U kana_u
keycode 45 = dollar kana_YO kana_yo
```

例 2:MODIFIER キーの変更式

```
!!キーコードの元の状態は
!!232(Shift_L):Shift_L
!!49(BackSpace):BackSpace
remove shift = Shift_L
keycode 49 = Shift_L
keycode 232 = BackSpace
add shift = Shift_L
```

という中身のファイルを

```
% xmodmap abc.xmodmap
```

と実行すると、例 1 では標準の状態で Shift+4 で出力される『$』が数字の『9』が表示されることになり、『9』のキーを打つと『$』が表示されるように割り当てられ、例 2 では、Shift_L と BackSpace が入れ替わっている。

4　現在のキーマップの構成

過去の研究 [1] で使われていたキーボード (FIJITSUX8546F) と現在のキーボード (Suntype6) は、キーマップが違うようである。そこで現在のキーマップの一部を載せる。

文字キーのキーマップを見るには、

```
% xmodmap -pke
```

とすると、現在のキーマップのキーコード及びキーシンボルが以下のように表示される。

KeyCode Keysym(Keysym)
Value Value (Name)
(中略)
keycode 11 = a A kana_CHI
keycode 12 = b B kana_KO
keycode 13 = c C kana_SO
keycode 14 = d D kana_SHI
keycode 15 = e E kana_I kana_i
(中略)
keycode 37 = 1 exclam kana_NU
keycode 38 = 2 quotedbl kana_FU
keycode 39 = 3 numbersign kana_A kana_a
keycode 40 = 4 dollar kana_U kana_u
keycode 41 = 5 percent kana_E kana_e
keycode 42 = 6 ampersand kana_O kana_o
(中略)

イコールの右側の一つ目のカラムは、数字や記号、小文字のアルファベットで修飾キーを押さない時のキーシンボルで、二つ目のカラムは修飾キーを押した時のキーシンボル値である。三つ目のカラムは mod2 を、四つ目のカラムは Shift+mod2 を押した時のキーシンボル値である。

修飾キーは

```
% xmodmap -pm
```

とすると以下のように設定されている。

shift Shift_L, Shift_R
lock Caps_Lock
control Control_L
mod1 Alt_L
mod2 Mode_switch
mod3 Num_Lock
mod4 Meta_L, Meta_R
mod5

左側に修飾キー名が表示され、右側にその修飾キーに登録されているキーのキーシンボル名が表示される。

例えば、shift という修飾キーには Shift_L と Shift_R の 2 つのキーが登録されているのがわかる。それぞれは、キーボードに向かって左側の Shift キーと右側の Shift キーである。

X Window System を使用しているときに、これらのシフトキーが「アルファベットキーと一緒に用いると、入力されるアルファベットが大文字になる」という「シフトキーとしての機能」を持っているのは、shift という修飾キーに登録されているからである。

言い替えれば、shift という修飾キーに登録されていればどんな修飾キーでもシフトキーとして使用できるのである。なお mod1 は、メタキーを表している。

5　過去の研究の変更例

5.1　左手のみで使用する場合

以下は過去の研究で左手のみで使用する場合の変更例の内の一つである、右手側の文字列を左手側にスライドさせたものである。

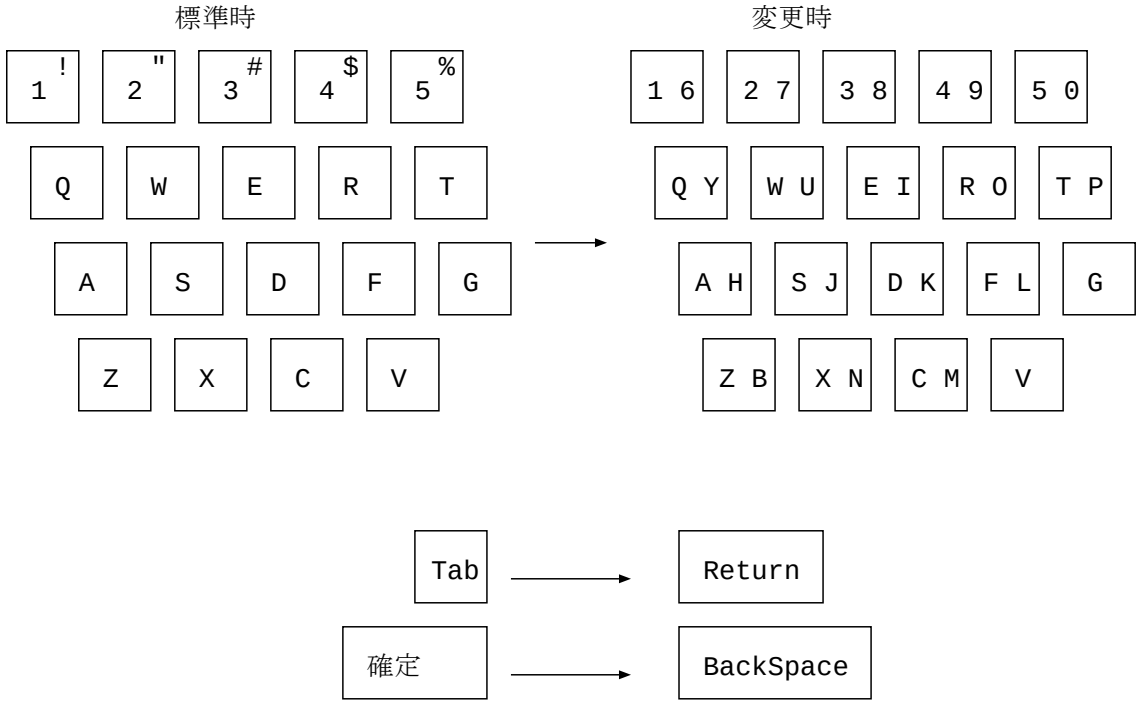

Fig. 1　左手用の変更例

特徴 左手で使うことを考慮し、文字列を左側部分に合わせスライドさせた形である。Tab キーを Return キーに割り当て、確定キーを BackSpace キーに割り当てた。それぞれは、右側にある文字は Shift キーと同時押しで、出力される。

長所 文字列をスライドさせた状態なので、文字、記号は比較的扱いやすい。

短所 記号に関しては全く考慮していない。BackSpace を押し続けによる連続的な削除が不可能で、一文字毎に削除することになる。

5.2 右手のみで使用する場合

以下は過去の研究で右手のみで使用する場合の変更例である、左手側の文字列を右手側にスライドさせたものである。

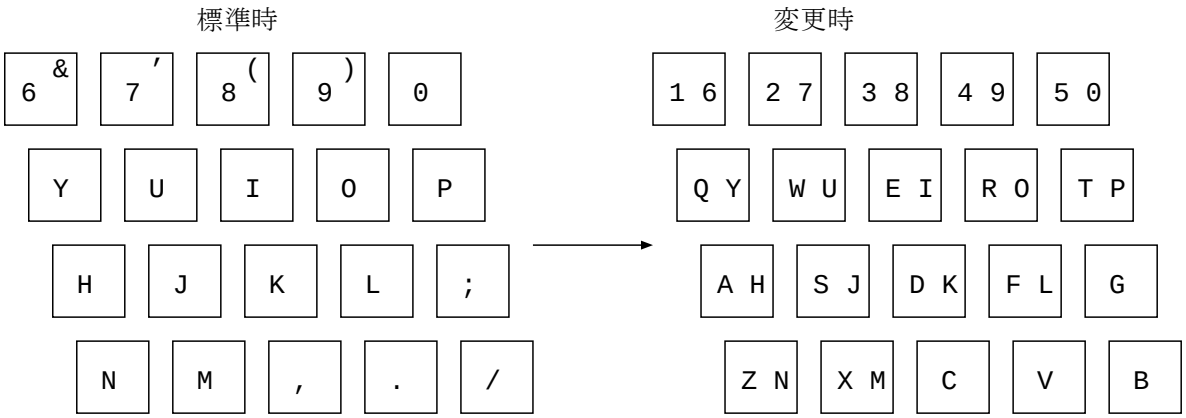

Fig. 2 右手用の変更例

特徴 右手で使うことを考慮し、右側部分の文字列に合わせスライドさせた形である。

長所 記号キーが横にあるので、左手で使うことを考慮したものよりは使いやすい。

短所 記号と G,C,V,B を入れ換えたので、多少、記号の扱いが不便になる。

5.3 過去の研究での反省点

左手用の変更例で挙げられた反省点は以下の通り。

- BackSpace キーは、確定キーではなくて、¥ キーに配置した方が良い。
- 確定キーを変換キーに配置した方が良い。
- “ー” や “？” は、日本語入力で使用頻度が高いので、左手で押せる範囲に配置をした方が良い。
- 日本語入力の場合、O、U、I の Shift キーを押しながらの出力は不便。

右手用の変更例で挙げられた反省点は以下の通り。

- 数字の並びはよいが、文字キーに関しては、通常入力と Shift+による入力は逆でも良い。
- 左手部分で構成した変更例よりは、記号キーや、使用頻度の高いキーがまとまっているので、左右どちらの手でも扱いやすい。
- G、C、V、B と変更した記号キーは、その変更したキーと Shift+で出力できた方が良い。

5.4 過去の研究で残された問題点

過去の研究で残された問題点は以下の通り。

1. 記号を考慮していない
2. BackSpace を押し続けによる連続的な削除が不可能だった

まず、一つ目の「記号を考慮していない」というのは、過去の研究ではスライドしてみただけなのでやむおえないが後で記載する新たな変更例についても一部の記号を考慮していない。

次に二つ目の「BackSpace を押し続けによる連続的な削除が不可能だった」というのは、押し続けて使うことのないキーを BackSpace にしていたからである。

「押し続けによる連続的な削除」が可能なキーと不可能なキーを調べたところ、以下のようであった。

可能なキーは以下の通り。

keycode 11 = a A kana_CHI
keycode 12 = b B kana_KO
keycode 13 = c C kana_SO
keycode 14 = d D kana_SHI
keycode 15 = e E kana_I kana_i
keycode 16 = f F kana_HA
keycode 17 = g G kana_KI
keycode 18 = h H kana_KU
keycode 19 = i I kana_NI

keycode 20 = j J kana_MA

keycode 21 = k K kana_NO

keycode 22 = l L kana_RI

keycode 23 = m M kana_MO

keycode 24 = n N kana_MI

keycode 25 = o O kana_RA

keycode 26 = p P kana_SE

keycode 27 = q Q kana_TA

keycode 28 = r R kana_SU

keycode 29 = s S kana_TO

keycode 30 = t T kana_KA

keycode 31 = u U kana_NA

keycode 32 = v V kana_HI

keycode 33 = w W kana_TE

keycode 34 = x X kana_SA

keycode 35 = y Y kana_N

keycode 36 = z Z kana_TSU kana_tsu

keycode 37 = 1 exclam kana_NU

keycode 38 = 2 quotedbl kana_FU

keycode 39 = 3 numbersign kana_A kana_a

keycode 40 = 4 dollar kana_U kana_u

keycode 41 = 5 percent kana_E kana_e

keycode 42 = 6 ampersand kana_O kana_o

keycode 43 = 7 apostrophe kana_YA kana_ya

keycode 44 = 8 parenleft kana_YU kana_yu

keycode 45 = 9 parenright kana_YO kana_yo

keycode 46 = 0 NoSymbol kana_WA kana_WO

keycode 47 = Return

keycode 49 = BackSpace

keycode 50 = Tab

keycode 51 = space

keycode 52 = minus equal kana_HO

keycode 53 = asciicircum asciitilde kana_HE

keycode 54 = at grave voicedsound

keycode 55 = bracketleft braceleft semivoicedsound kana_openingbracket

keycode 57 = bracketright braceright kana_MU kana_closingbracket

keycode 58 = semicolon plus kana_RE

keycode 59 = colon asterisk kana_KE

keycode 61 = comma less kana_NE kana_comma

keycode 62 = period greater kana_RU kana_fullstop

keycode 63 = slash question kana_ME kana_conjunctive

keycode 64 = Caps_Lock

keycode 65 = F1

keycode 66 = F2

keycode 67 = F3

keycode 68 = F4

keycode 69 = F5

keycode 70 = F6

keycode 71 = F7

keycode 72 = F8

keycode 73 = F9

keycode 74 = F10

keycode 75 = SunF36

keycode 76 = SunF37

keycode 77 = F22 F22 Print SunSys_Req

keycode 78 = F23 F23 Scroll_Lock

keycode 79 = F21 F21 Pause Break

keycode 81 = Home

keycode 82 = Prior

keycode 85 = Next

keycode 86 = Right

keycode 87 = Left

keycode 88 = Down

keycode 89 = Up

keycode 91 = F25 F25 KP_Divide

keycode 92 = F26 F26 KP_Multiply

keycode 93 = F24 F24 KP_Subtract

keycode 94 = KP_Add

keycode 95 = KP_Enter

keycode 96 = F33 F33 KP_1 End

keycode 97 = Down F34 KP_2

keycode 98 = F35 F35 KP_3 Next

keycode 99 = Left F30 KP_4

keycode 100 = F31 F31 KP_5

keycode 101 = Right F32 KP_6

keycode 102 = F27 F27 KP_7 Home

keycode 103 = Up F28 KP_8

keycode 104 = F29 F29 KP_9 Prior

keycode 105 = KP_Insert KP_Insert KP_0

keycode 106 = Delete Delete KP_Decimal

keycode 108 = Multi_key

keycode 109 = SunPowerSwitch SunPowerSwitchShift

keycode 125 = F13 F13 SunProps

keycode 126 = F15 F15 SunFront

keycode 127 = F11 F11 Cancel

keycode 128 = F12 F12 Redo

keycode 130 = F20 F20 SunCut

keycode 131 = F16 F16 SunCopy

keycode 132 = F18 F18 SunPaste

keycode 134 = SunAudioMute SunVideoDegauss

keycode 135 = SunAudioRaiseVolume SunVideoRaiseBrightness

keycode 136 = SunAudioLowerVolume SunVideoLowerBrightness

keycode 142 = backslash underscore kana_RO

keycode 144 = backslash bar prolongedsound

「押し続けによる連続的な削除」が不可能なキーは以下の通り。

keycode 48 = Escape
keycode 64 = Caps_Lock
keycode 79 = F21 F21 Pause Break
keycode 80 = Insert
keycode 84 = End
keycode 90 = Num_Lock
keycode 124 = Help
keycode 129 = F14 F14 Undo
keycode 133 = F19 F19 Find
keycode 143 = Henkan_Mode
keycode 145 = Kanji
keycode 146 = Execute
keycode 231 = Control_L
keycode 232 = Shift_L
keycode 233 = Alt_L
keycode 234 = Meta_L
keycode 236 = Shift_R
keycode 237 = Mode_switch
keycode 238 = Meta_R

6　新たな変更例

6.1　左手スライドの改良型

本研究では、右利きで右手が使えない状況を対象とするので、左手用スライドのみで過去の研究の反省点を生かしたキー配列と、自分で考えたキー配列を考察する。

以下は左側文字列に合わせスライドさせた配列の改良型である。

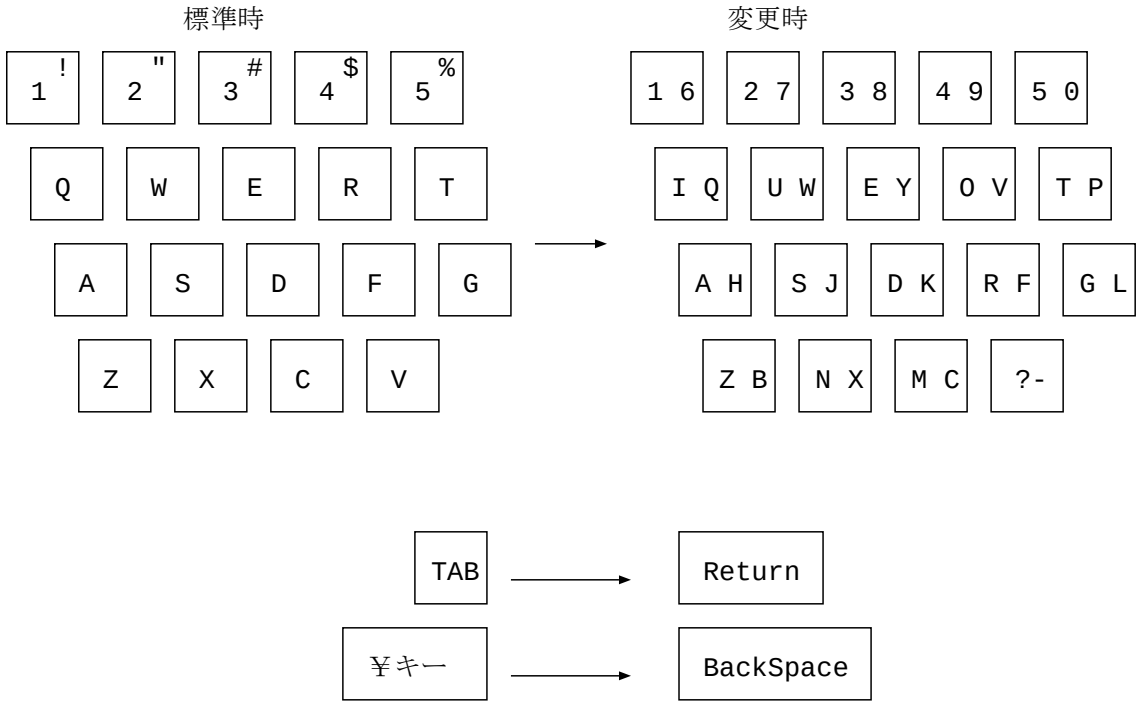

Fig. 3　左手スライドの改良型

特徴 変更点は¥キーを BackSpace キーに割り当て、R を F の左側に、F は右側に、V があった位置に『?』と『ー』を置いた。日本語入力の場合、過去の並びは、O、U、I が Shift キーと同時押しなければ出力できなかったので、同時押ししなくても出力できるようにした。

長所 文字列をスライドさせた状態をベースにしているので、文字、数字は原型ほどではないが、比較的扱いやすい。BackSpace を押し続けによる連続的な削除が可能である。母音は Shift キーを押さなくても出力できる。

短所 記号に関しては全く考慮していない。文字配列が過去の左手スライドと比べて変わってる所があり、慣れが必要。

6.2 自分で考えた並び

以下は自分で考えた左手用の並びである。

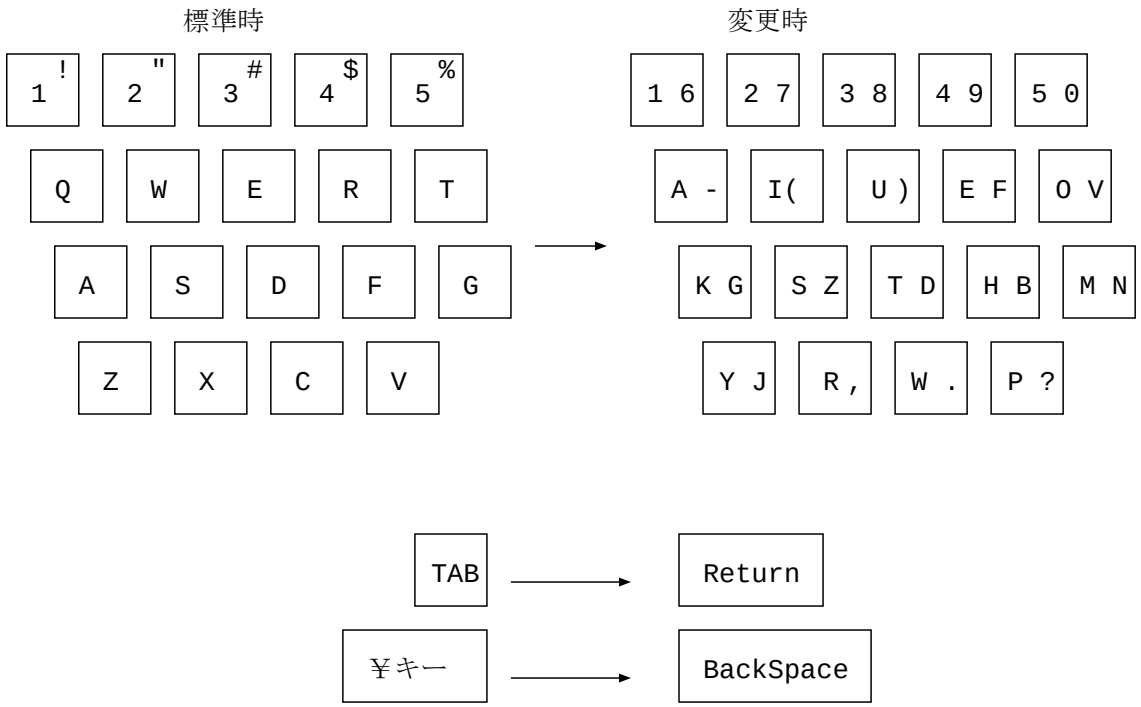

Fig. 4 自分で考えた並び

特徴 左側は、母音は数字の下に並べ、子音は五十音順に並べた。右側は、左側の濁音を置いた。日本語で使われない、X、L、Q は無い。

長所 BackSpace を押し続けによる連続的な削除が可能である。母音は Shift キーを押さなくても出力できる。

短所 一部の記号を考慮していない。並びが全く新しく、覚えるのが面倒。よく使う N が入力しづらい。

7 実験

7.1 実験にあたって

配置が使いやすくなっているかどうかは、見た目ではわからないので、被験者に試してもらうしかない。そこで、被験者 2 人に実際に使用させ、使いやすさや、慣れやすさ、か

かった時間で比較するために実験する。実験するにあたって過去の並びとその改良、その2つと新しい並びの改良点をまとめると以下の通りである。

- 過去の並び→過去の並びの改良

  よく使う母音を Shift キーを押さなくても出力できるようにした

- 過去の並び、過去の並びの改良→新しい並び

  右側になるアルファベットは左側のアルファベットの濁ったものを置き、慣れやすくした。

そこで改良されているかどうかを、調べるために以下の4通りの並びで被験者2人に試してもらい、時間計測とそれぞれ後述するアンケートに答えてもらった。

1. 過去の研究の並び
2. それを改良した並び
3. 全く新しい並び
4. 片手で文字キーの配列を全て入れ換えた並び (片手フルキー)

7.2 実験道具・方法

- 実験道具
  - ワークステーション
  - キーボード
  - キーボードカバー
  - シール
- カバーの詳細

  商品名：ピタッとシート

  製造元：エレコム株式会社

  材質：シリコーン変性熱可塑性ポリウレタンエラストマー

  厚さ：0.15mm
- 実験方法

  キーボードに以下の透明なカバーを貼り、さらにその上にシールのようなものを貼る。Fig.5、6 参照。

Fig. 5 拡大図

Fig. 6 全体図

7.3 調べる項目

過去の並びや、片手のみでの欠点の解消度を比較するために、以下の項目を比べる。

- 疲れ度
- 要した時間 (Enter キーを押してから、また Enter キーを押すまで)
- キー配列の使いやすさ

そのためには、被験者に以下の 2 つの例文をそれぞれ 2 回ずつ打ってもらいアンケートに答えてもらう。

1. everything in japan seems about half the size of things in our society
   (意味：日本の全ては、私達の社会の物事の約半分の大きさのように見える) [4)]
2. fngkf82tfn5igfzadnmk
   (ランダムに生成した 20 文字の数字とアルファベットの内、改良後の並びに無い x,l,q が無く、母音が少ないものを選んだ) [5)]

アンケート項目は以下の通りで、5 段階評価のアンケート (5 そう思う、4 少しそう思う、3 どちらでもない、2 少しそう思わない、1 そう思わない) とする。

1. 疲れにくいと感じた
2. 慣れれば速いと思った
3. キー配列は使いやすいと思った

時間計測は以下の通り。

1. それぞれに要した時間
2. 短縮された時間 (1 回目 −2 回目)

被験者が試す順番は、それぞれ被験者 A が片手フル→過去左→過去左の改良→新しい並びの順、被験者 B が過去左→新しい並び→片手フル→過去左の改良の順である。ただし、並びによっては無い文字があるので、その時のみ空いている所に配置した。

時間計測は、ミスタイプをした時間も含まれる。

アンケートは個人の感想である。

8　実験結果と被験者の感想

8.1　過去の左手スライド

過去の左手スライドでの実験結果を以下に記す。アンケート結果は Table3、かかった時間は、Table4 の通りとなった。

アンケート項目	被験者 A	被験者 B	平均
疲れにくいと感じた	2	3	2.5
慣れれば速いと思った	2	5	3.5
キー配列は使いやすいと思った	1	4	2.5

Table 3　過去の左手用のアンケート結果

(単位：秒)		1 回目	2 回目	差
被験者 A	例文 1	188.06	143.15	−44.51
	例文 2	67.57	54.96	−12.61
被験者 B	例文 1	190.99	132.26	−48.73
	例文 2	49.21	39.42	−9.79

Table 4　過去の左手用での結果

この結果から、過去の左手スライドは人によっては使いにくい並びであることがわかる。

8.2　過去の左手スライドの改良

過去の左手スライドでの改良の実験結果を以下に記す。アンケート結果は Table5、かかった時間は、Table6 の通りとなった。

アンケート項目	被験者 A	被験者 B	平均
疲れにくいと感じた	3	4	3.5
慣れれば速いと思った	4	5	4.5
キー配列は使いやすいと思った	3	5	4

Table 5　過去の左手用の改良型のアンケート結果

(単位：秒)		1 回目	2 回目	差
被験者 A	例文 1	126.27	108.10	−18.17
	例文 2	57.00	49.55	−7.05
被験者 B	例文 1	112.55	91.18	−21.37
	例文 2	43.74	34.69	−9.05

Table 6 過去の左手用の改良型の結果

この結果から被験者 A、B ともに過去の左手用より速い結果となった。アンケート結果も全て過去の左手スライドより良い結果となった。

8.3 新しい並び

新しい並びの実験結果を以下に記す。アンケート結果は Table7、かかった時間は、Table8 の通りとなった。

アンケート項目	被験者 A	被験者 B	平均
疲れにくいと感じた	3	4	3.5
慣れれば速いと思った	3	4	3.5
キー配列は使いやすいと思った	3	4	3.5

Table 7 新しい並びのアンケート結果

(単位：秒)		1 回目	2 回目	差
被験者 A	例文 1	141.27	129.35	−11.92
	例文 2	67.85	62.01	−5.84
被験者 B	例文 1	159.05	121.25	−38.80
	例文 2	51.99	41.47	−10.52

Table 8 新しい並びの結果

この結果から、過去の左手スライドのタイムと比べると例文 1 では被験者 A、B とも速くなっているが、例文 2 では逆に被験者 A、B とも遅くなった。改良型と比べるとタイムもアンケート結果も悪い結果となった。

8.4 並び変えフルキー

並び変えフルキーの結果を以下に記す。アンケート結果は Table9、かかった時間は、Tab le10 の通りとなった。

アンケート項目	被験者 A	被験者 B	平均
疲れにくいと感じた	3	2	2.5
慣れれば速いと思った	3	2	2.5
キー配列は使いやすいと思った	2	1	1.5

Table 9 並び変えフルキーのアンケート結果

(単位：秒)		1 回目	2 回目	差
被験者 A	例文 1	149.63	141.95	−7.68
	例文 2	68.23	42.27	−25.96
被験者 B	例文 1	156.06	116.90	−40.16
	例文 2	56.57	42.68	−13.89

Table 10 並び変えフルキーの結果

この結果から、被験者 A では、例文 1 では時間があまり短縮されなかったが、例文 2 では大幅に短縮された結果となった。被験者 B では、例文 1 の 2 回目で大幅に短縮されているが、例文 2 では一番遅く、アンケートの「慣れれば速いと思った」では「2」、「使いやすさ」では「1」という結果になった。

8.5 考察

本研究での結果をまとめると以下の通りになった。

- 例文 1 の時間:改良型 > 新しい並び > 並び変えフルキー > 過去の研究
- 例文 2 の時間:改良型 > 並び変えフルキー > 過去の研究 > 新しい並び
- アンケート結果:改良型 > 新しい並び > 過去の研究 > 並び変えフルキー

片手フルキーは Shift キーを押さくていいとはいえ、キーの配置がバラバラなので一番慣れにくいと思われた。しかし、例文 2 では他の並びよりタイムを短縮している。これは Enter キー以外のキーを押さず、文字数が少ないので、慣れで速くなったと考えられる。

逆に文字数が多いとあまり速くならない、もしくは 2 回目の方が遅くなることも考えられる。

過去の左手スライドは、片手のみで、片手の範囲で、さらに Shift キーを押しながらの入力は、慣れているはずが無く、ギャップで一番悪い結果になったと考えられる。順番によっては例文 1 では新しい並びと同程度のタイムだったと思われる。

一方、タイム、評価共に最も良かったのは過去の左手スライドの改良であるが、タイムの方は、片手入力に少し慣れた後ということで速くなったと考えられる。

新しい並びは、評価もタイムも普通だが、タイムがあまり短縮できていないので、一番慣れにくいと考えられる。やはり並びが全く新しいためである。

9　まとめ

本研究では、片手でのキーボード利用の問題点の解決を目指し、そして過去の研究で考察されたキー配列、それを改良したキー配列、さらに全く新しいキー配列を提案し、比較するための実験を行なった。

まず、過去の研究で残された問題点は大きく分けると二つある。一つ目は、記号を考慮していなかったということであるがスペースの関係で、本研究でも一部考慮していない。二つ目は、BackSpace を押し続ける事による連続的な削除ができなかったということであるが、これは、普段押し続けて使うことのないキーを BackSpace に変えたからである。

片手のみでフルキーを扱うこと自体に関する問題点は、以下の 3 つであった。

1. キー範囲が広く、疲れやすく、時間がかかる
2. Shift と同時押しする時、中心のキーは押しにくい
3. 左手側のキーでタイプミスをすると BackSpace が遠い

次に、片手のみでフルキーを扱うこと自体に関する問題点を解決するために、3 つの並びを実験して比較した結果をまとめる。まず 1. の問題について被験者 A、B ともにが片手フルキーより疲れにくいと感じたのは「過去の研究の改良型」と「新しい並び」、慣れれば時間がかからないと感じたのが「過去の研究の改良型」と「新しい並び」である。被験者 B にかぎり「過去の片手スライド」も慣れれば時間がかからないと感じている。慣れれば長い文なら改良型以降が速くなり、短い文なら Shift キーを押さなくていい片手フルが速くなる。2. の問題は片手のみの範囲になっていると実はどの位置のキーと同時押してもあまりかわらないようである。3. の問題は BackSpace を 1 の左の位置に置いたためかなり解決できたと思われる。

本研究では、右手がなんらかの理由で使えず、左手のみでキーボードを扱うことで生まれる問題点を解決しようと研究してきたが、問題点の一つの疲れやすさやかかる時間は個人差がある。そこで、より多くの人に試してもらい、より多くのデータを取ることが今後の課題である。

参考文献

[1] 柴信一郎：「一時的な片手でのキーボードの利用に関する考察」
(新潟工科大学工学部情報電子工学科卒業論文、2003)

[2] 松田晃一、暦本純一：入門 X Window(アスキー出版局、1993)

[3] 木下凌一、小嶋和子、日高明美：
入門 X Window OSF/Motif Window Manager(日刊工業社、1990)

[4] 石井隆之、喜多尊史、Joe Ciunci、Lance Burroes、馬渡秀孝:
Step Up to Better English『日英おもしろ文化比較』(朝日出版社, 2011)

[5] 複数のランダムな文字列を作成します:

`http://www.perfectsky.net/cgi/randomstringgenerator.cgi`