空中手書き文字入力におけるインプットメソッドの研究

西田好宏*、吉田大志*

Input method Editor for Aerial Handwritten Gesture Recognition

Yoshihiro Nishida*, Taishi Yoshida*

This paper describes a method to recognize a character handwritten in the air. This recognition method uses the motion direction instead of positions of the device. It also doesn't use the information of pen up and down. It selects and orders character candidates with DP (dynamic programming) matching. We prototyped an application software that recognize a character by using mouse movements. In this time, we made virtual key input motion for input method editor. We could edit a sentence by using handwritten gesture recognition.

Keywords: 文字認識, 空中手書き入力, インプットメソッド, Gesture, IME

１．はじめに

クラウド・コンピューティングの普及により、いつでも、どこでも、ふと思いついたアイデアなどをメモとしてクラウドに保存したり、出来事をつぶやきあったりする機会が増えている。当然、そのためにはテキストデータの入力が必要で、片手しか使えなくても簡単に文字を入力できたら、より便利になると考えられる。また、デジタルサイネージが普及し、単に情報を流すだけでなくインタラクティブな操作や検索が可能になっているが、不特定多数の人が操作するため非接触で操作や文字を入力したいという要求もある。

そのような背景の中、絶対位置情報を利用せずに相対的な移動方向情報のみを利用して、１文字単位で一筆続け書き文字として認識する空中手書き文字入力を開発している。

この空中手書き文字入力を実現のためには、下記の３つの処理が必要である。

①　人が文字を書く動きから筆記の移動方向の動きを検出するセンサ処理

②　相対的な移動方向から文字を一文字単位で認識する文字認識エンジン

③　認識した文字を基に変換や確定等で文章化して汎用的な文字入力にする処理

この３つのうち、①のセンサ処理は使用するデバイスや目的とするシステムにより異なるため最後に回し、②の文字認識エンジンから着手し概ね動作するものができた。

今回は③のインプットメソッドについて検討を行い、文字では表せないが文章の入力には必要となるエンターやバックスペースといった特殊なキーの入力方法の提案を含め、認識した文字を基に文章化してアクティブアプリケーション（メモ帳）に出力できるようにした。

＊ 電気電子情報工学科

２．インプットメソッド

２－１．概要

パソコンでは、一般に欧文の入力はキーボードでタイプライター同様直接タイプすればよいが、日本語（仮名、漢字）や中国語（漢字）、朝鮮語（ハングル）など、使用文字数が数千を超える言語の文章を入力する際には全ての文字に一つのキーを当てはめることは非現実的である。

そのため、複数のキー操作で１文字を入力するなどの仕組みが必要となり、このように広い意味で文字を入力するための仕組みと環境を提供するソフトウェアをインプットメソッドと言う。

日本語のインプットメソッド（日本語入力システム）は、日本語版 MS-DOS では多くの場合、フロントエンドプロセッサ（FEP、日本語入力フロントエンドプロセッサ）として、キー入力に割り込むかたちで実装されていた。Windows においては、インプットメソッドをインプット メソッド エディタ（IME）と呼び、これが定着している。

その結果、キーボードでパソコンに文字を入力する場合には、アプリケーションに関係なく同じ操作で文字の変換や確定を行い、文章を編集することができる。

その後、入力方法がキーボードからタッチパネルや携帯電話などに変化すると、英文入力にもキー数が不足するため何らかの方法が必要となった。例えば、タッチパネルを採用した iOS のインプットメソッド例を図１に示す。パソコンのキーボード入力と異なり、タッチパネル上に最小限のキーを表示して文字の選択や確定が行えるように工夫されている。さらに、１文字入力すると入力する文字を推測して候補を表示するなど予測変換等の技術が進化して、少ない文字入力で文章を作成・編集できるようになっている。

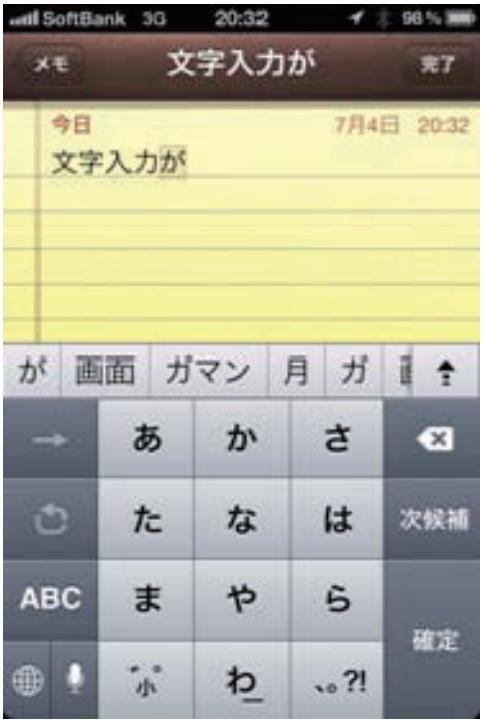

図１　iOS のインプットメソッド画面例

空中手書き文字入力用のインプットメソッドを開発するに当たっては、従来のキーボードやタッチパネルで培われて来た技術を出来るだけ利用するように努め、図２に示す通り今まで開発してきた文字認識エンジンで認識した文字を IME に渡す事に加えて、従来のキー入力の代わりとなる空中操作（ジェスチャ）を認識して IME に渡し、その変換結果をアクティブアプリケーションに渡す構成とした。

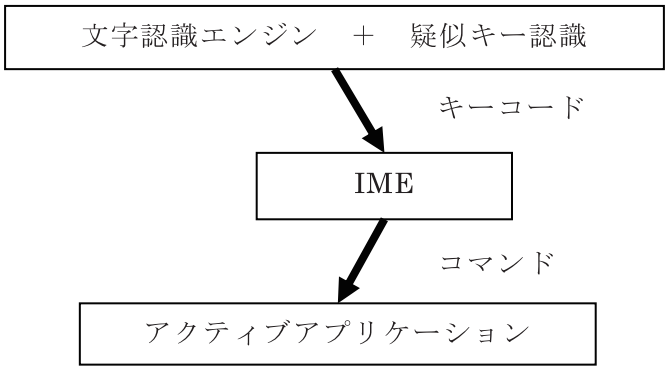

図２　インプットメソッドの構成

２－２．実装

今回は予測変換を使用するので、IME を GoogleIME、出力するアプリケーションをメモ帳としてプログラムの作成を行った。プログラムの起動と共に ShellExecute 関数によってメモ帳を起動し、そのメモ帳に文字を出力していく形をとっている。

図 3 の ShellExecute 関数の第二引数 lpVerb には実行するべき操作を指定する。"open"の 場合、第三引数 lpFile に指定する文書または実行可能なファイル、あるいは１つのフォルダを開く命令を送る。第三引数に指定したファイルの種類によっては第二引数の命令が実行されない場合がある。第六引数の nShowCmd には実行するファイルの表示方法を指定する。第三引数が文書ファイルの場合はそれを開くアプリケーションに指示が渡される。SW_SHOWNORMAL が指示された場合はウィンドウをアクティブにして表示する。

```
HINSTANCE ShellExecute(
    HWND hwnd,              // 親ウィンドウのハンドル
    LPCTSTR lpVerb,         // 操作
    LPCTSTR lpFile,         // 操作対象のファイル
    LPCTSTR lpParameters,   // 操作のパラメータ
    LPCTSTR lpDirectory,    // 既定のディレクトリ
    INT nShowCmd            // 表示状態
);
```

```
ShellExecute(NULL,"open","notepad.exe",NULL,NULL,SW_SHOWNORMAL);
```

図 3. ShellExecute 関数と実際のコード

文字の出力には図４に示したWindowsAPIのSendInput関数を使用する。第一引数である入力イベントの数は、第二引数の挿入する入力イベントの配列の数を指定する。第二引数には挿入する入力イベントの配列を指定する。

```
UINT SendInput（
  UINT nInputs,     // 入力イベントの数
  LPINPUT pInputs,  // 挿入する入力イベントの配列
  int cbSize        // 構造体のサイズ
);
```

図 4. SendInput 関数

実際のプログラムは図５に示したKeyAction関数を利用して、第一引数にWORD型の一文字あるいは仮想キーコードを渡した。また、第二引数のBOOL型変数であるbKeepPressingはキーを押し続けるか否かを指定して、例えばアルファベットの大文字を入力する際に必要なSHIFTキーに使用している。

```
void KeyAction(WORD VirtualKey, BOOL bKeepPressing)    //SendInput API
{
                                INPUT input[1];

    input[0].type = INPUT_KEYBOARD;
    input[0].ki.wVk = VirtualKey;
    input[0].ki.wScan = MapVirtualKey(input[0].ki.wVk, 0);
    input[0].ki.dwFlags = KEYEVENTF_EXTENDEDKEY;
    input[0].ki.time = 0;
    input[0].ki.dwExtraInfo = ::GetMessageExtraInfo();
    ::SendInput(1, input, sizeof(INPUT));
    if (!bKeepPressing)
    {
        input[0].ki.dwFlags = KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP;
        ::SendInput(1, input, sizeof(INPUT));
    }
}
```

図５．KeyAction 関数

このSendInput関数による命令でメモ帳へと文字を送るためには、メモ帳が最前面のアクティブウィンドウでなければならない。そのため、SendInput関数を実行する前にメモ帳をアクティブウィンドウにするための命令を実行する必要がある。プログラムではその命令に図６および図７に示したFindWindow関数およびSetForegroundWindow関数を使用している。

```
HWND FindWindow(
  LPCTSTR lpClassName,  // クラス名
  LPCTSTR lpWindowName  // ウィンドウ名
);
```

```
::FindWindow("Notepad", NULL);
```

図６．FindWindow 関数と実際のコード

```
BOOL SetForegroundWindow(
  HWND hWnd   // ウィンドウのハンドル
);
```

```
::SetForegroundWindow(hWndID);
```

図７．SetForegroundWindow関数と実際のコード

３．疑似キー認識方法

文章を作成するためには、認識した文字を基に変換、確定や削除を行うためキーボードのエンターやバックスペースといった文字以外の入力が不可欠である。そこで、文章の入力に必要な最低限のキーをエンター、バックスペース、タブ、スペース、半角/全角、カタカナ変換（F7）の６つとし、これらのキーに対して簡単な動作（疑似キー操作）を割り当て、認識する方法を採用した。表１にキーとその動作を示す。各動作は２ステップ程度の直線の組み合わせで構成されている。これらの動作について、エンターは改行記号と同じ動作、バックスペースは１文字戻ることをイメージした動作、スペースはボタンを押す動作といった具合に、できるだけわかりやすい、覚えやすい動作を意識して決定した。全角入力、半角入力の双方で使用する句点・ピリオドおよび読点・コンマについても同様にできるだけわかりやすい、覚えやすい動作を意識して決定している。句点・ピリオドについては０（ゼロ）や０（オー）とは逆方向へ１回転させる動作を、読点・コンマについては右下へと直線を引く動作を割り当てている。

また、全角入力における小文字（「っ」や「ゅ」など）を入力する場合、文字を入力する前に横に３回動かす動作を入れることで、小文字で入力できるようになっている。この動作は、横に３回動かすという動きが左右どちらか片方から始まるものと決定してしまうと、使用できる動作がひとつ増えるものの、左右どちらから始めるのか覚えづらいため、どちらから始めてもよいように右から書き始めても左から書き始めても認識するようにしている。

４．予測変換導入の効果

GoogleIME を用いてその予測変換の効果例を確認した。GoogleIME ではパソコンのキーボード入力の場合に文字を入力した後に TAB キーを入力することで予測変換が可能である。そこで、文字を入力する度に自動的に TAB キーコードも送り毎回予測変換を使用した場合と、しない場合について「９時までに会場に集合する」という例文でキーの入力回数を比較した。

この時の入力ログは次のとおりである。

予測変換を使用した場合

HZ９じ TBEN までに EN かいじ TBEN に EN し SM ゆう TB２EN

予測変換を使用しなかった場合

HZ９じ SPEN までに EN かいじ SM ように SPSPEN し SM ゆうごう SPSPEN する EN

ログ中の記号はそれぞれ、HZ は半角/全角、TB はタブ、EN はエンター、SP はスペース、SM は小文字のキー入力を表している。

予測変換を使用したのは３カ所で、そのうち入力を短縮できたのは２カ所だけとなった。

１カ所目は「９じ TBEN」の部分で、予測変換で「９時」に変換しているが、予測変換を使用していない通常の変換でも同じように変換できるため、短縮はできていない。

２カ所目は「かいじ TBEN に EN」の部分で、「かいじ TBEN」を予測変換で「会場」に変換してい

る。予測変換を使用していない場合と比べ、３回入力を短縮できている。

3 カ所目は「し SM ゆう TB２EN」の部分で、予測変換の第二候補で「集合する」に変換している。第二候補ということで、候補を選択するために一回分入力が増えているが、予測変換を使用していない場合と比べ、5 回入力を短縮できている。

結果として、予測変換を使用した場合は 23 回の入力、文字のみの入力は 12 回、予測変換を使用しなかった場合は 31 回の入力、文字のみの入力は 18 回となり、全体の入力では 8 回、文字の入力では 6 回、入力を短縮することができた。

今回の結果は、あまり学習されていない状態での予測変換を使用してのものなので、学習をすすめることでより入力回数が短縮できるものと思われる。

表１．特殊キーとその疑似操作

キー	動作	プログラム内での記号
エンター		EN
バックスペース		BS
タブ		TB
スペース		SP
半角／全角		HZ
カタカナ変換		RN
小文字		SM
句点、ピリオド		PR
読点、コンマ		CM

５．今後の課題

現状の辞書データは図８に示した通り、ひとつのテキストデータにひらがな・アルファベットの区別なく、まとめて記述されているものを取り込んで使用している。そのため、プログラム側で現在の変換状態に合わせて出力する文字と、そうでない文字とを区別する必要性があるうえ、テキストデータの内容が膨大で煩雑となり変更を加えづらいという問題がある。

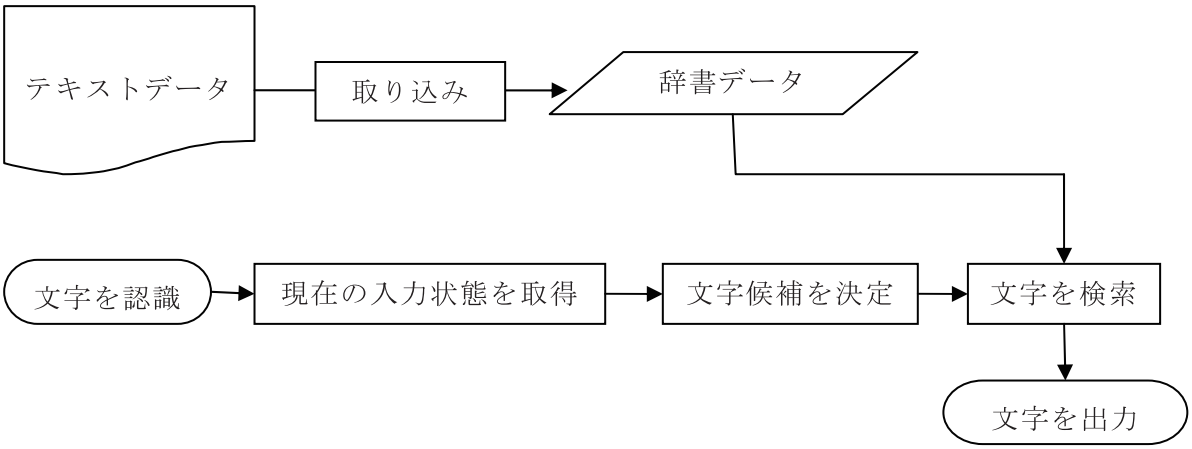

図８．現状の辞書データとアルゴリズ

この問題の対策として、元となるテキストデータを分割することが考えられる。こうすることで図９のように辞書データも全角入力・半角入力といった具合に分割し、その時の入力状態にあわせて辞書データを切り替えることによって、プログラム側で変換状態に合わせて出力する文字を区別する必要がなく、また、テキストデータを分割するため、その内容がひらがな・アルファベットなどで統一されるので現状よりも見やすくすることができると考えられる。

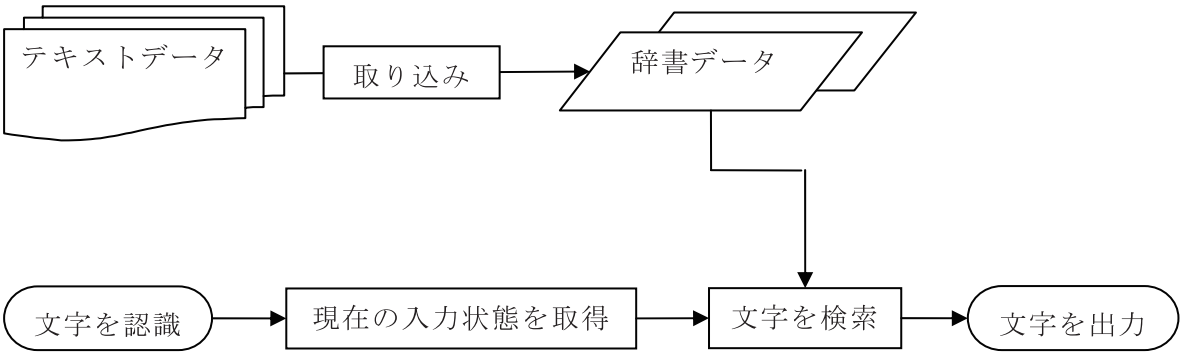

図９．問題の対策例

６．まとめ

今回、空中手書き文字入力のインプットメソッドとして、パソコンの汎用的な文字入力メソッドのインタフェースを利用したキーコード入力、アクティブアプリケーションのフォーカス処理について検討して、認識した文字を基に文章化してアクティブアプリケーションとして、メモ帳に出力できるようにした。

また、最小限必要な疑似キーを選びその空中操作を定義し、マウスで手書きした文字に加えて疑似キー入力を認識して IME 経由で文章を作成できることを確認した。文字ではない特殊なキーの動作については、そのキーをイメージしたわかりやすい動作を割り当てるようにしている。現状ではエンターは改行記号と同じ動作、バックスペースは 1 文字戻ることをイメージした動作、スペースはボタンを押す動作といった具合に 2 行程ないし、3 行程の動作を割り当てている。

さらに、自動的に予測変換機能を利用するキーコードを追加することで、より少ない入力数で文章を作成できることが確認できた。

今後もインプットメソッドの検討を続け、さらに覚えやすく効率の良い空中手書き用のインプットメソッドに改良して行くとともに、今までのマウスの動きを基にした文字認識エンジンおよびインプットメソッドの研究に加え、これからは残りの課題である筆跡の動きを検出するセンサ技術の研究に注力し、センサを含めたシステムとして完成させたい。

参考文献

[1] 西田好宏，小倉一孝、三浦浩一、松田憲幸、瀧寛和、安部憲広：移動方向情報のみを利用した空中手書き文字認識；ヒューマンインタフェース学会誌、Vol.12、No.3、pp.289-296、2010 年

[2] 西田好宏: 移動方向情報とストローク比率を利用した空中手書き文字認識;ヒューマンインタフェースシンポジウム，HIS2010,pp.311-314 (2010).

[3] 青池勇樹，西田好宏：移動方向に基づく空中手書き文字認識のアルファベット対応の検討：ヒューマンインタフェースシンポジウム，HIS2011,pp.701-702 (2011).

[4] 西田好宏，入江臣知，吉田大志：空中手書き文字入力におけるインプットメソッドの検討：ヒューマンインタフェースシンポジウム，HIS2012,pp.317-318 (2012).

(平成 25 年 3 月 31 日受理)