文字認識ソフトの作成

３年Ｃ組　上田　樹
指導教員　米田　隆恒

１．要約

私は、看板などで読めない漢字を見つけたとき、読みがわからなければ意味を調べるのも困難であると感じた。そこで、日本語に限らず、それらの文字をすべて写真に撮るだけで認識できれば、外国などへ行って読めない文字に出会っても便利であると感じた。

市販の文字認識ソフトもあるが、文字の状態によっては読めないことがある。そこで、Visual Basic を用いて文字認識ソフトを自分で作成することにした。

キーワード　　文字認識、２値化、細線化、輪郭追跡、学習機能

２．研究の背景と目的

看板などで読めない漢字や知らない言語の文字などを見つけたとき、そのたびに意味を調べるのは大変である。そこで、それらをすべて写真に撮るだけで文字を認識できればよいと考えた。携帯電話などに付属している文字認識ソフトは、文字の並びや大きさによっては読めない、文字のバランスが悪いと読めない、斜めだと読めない、言語によっては読めないなど、多くの問題がある。そこで、これらの問題を解消した文字認識ソフトを作成することにした。

３．研究内容

フォントデータから取得した文字の形を数値化し、データベースにその文字の読みと共に保存する。写真から取得した文字の形を数値化し、データベースの数値と比較することで、機械的にどの文字かを判定することができると考えた。文字の形を数値化する方法として、「輪郭追跡」の方法をとった。それには写真から文字部分だけを取り出す必要がある。それには写真を白黒２色にする「２値化」と、線の中心１画素分だけを残し線を細くする「細線化」が使えるのではないかと考えた。

輪郭追跡の説明は後にし、２値化について説明する。「２値化」とは、写真などの濃淡のある画像を白と黒の２階調に変換する処理を２値化という。文字の色は普通、背景色とは違う色で書かれるので、文字部分を取り出すのに使えるのではないかと考えた。

文字認識をするためには、文字だけを黒、その他が白になるような処理を行う必要がある。まず、白と黒の境界を決めるための「輝度のしきい値」を、画像全体の輝度値の平均にしてみたがうまくいかなかった。そこで、まず全画素の輝度値を計算し、各輝度値の画素数をカウントし、図１のようにグラフにした。

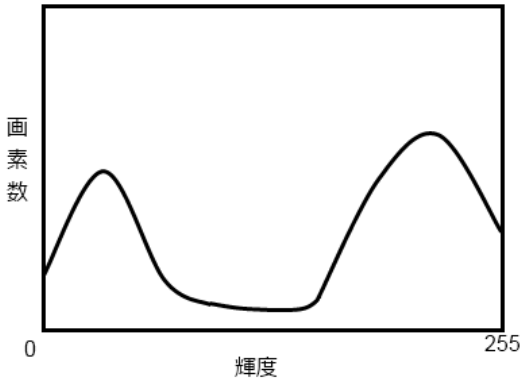

図１　輝度のグラフ

このグラフで、両隣より数値が高い所をピーク、低いところをボトムとする。グラフのピークとボトムを記録し、最も画素数が多いピーク、２番目に画素数が多いピークを、それぞれ背景の部分と文字の部分であると考えて、その２つのピークの中央をしきい値とした。

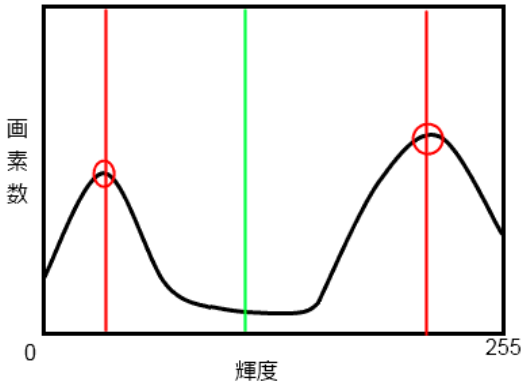

図２　ピークの中央をしきい値とする

図３　結果

しかし、これでは図３のように、まだノイズが残ってしまった。そこで、図４のように２つのピークの中央値を含む凹部分のボトム、中央値がピークの場合は最も近いボトムをしきい値としたところ、図５のようにノイズとなっていた部分が正しく２値化され、きれいに２値化ができるようになった。

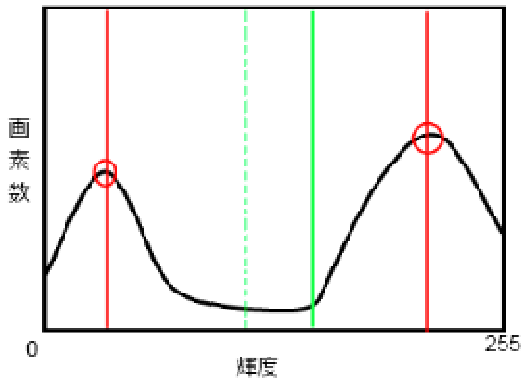

図４　しきい値を求める

図５　改善後の二値化結果

だが、その後さまざまな写真で試したところ、背景にグラデーションなどが使われている場合は読み取れないことがわかった。

グラデーションを背景とする画像の輝度値をグラフにしたところ、図６のようになっていた。

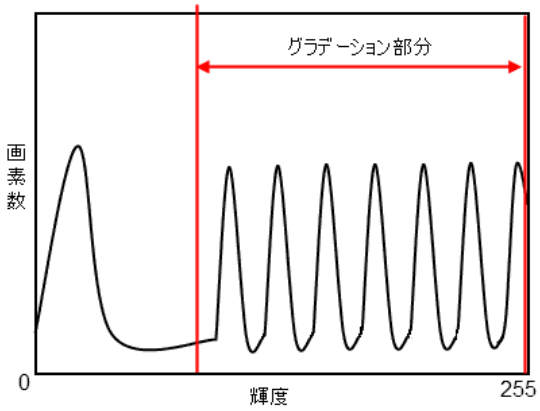

図６　グラデージョン時の輝度値のグラフ

本来、グラデーション部分が平らになるはずだが、画素数が小さいため、間が省略され波になっていると考えられる。この問題は画素数を上げることでも解決できるが、処理に時間がかかってしまうため、グラデーション部分がまとめられるようにした。まず、輝度値の平均より数値が高いピークをすべて記録し、隣り合ったものとの距離を求める。グラデーションの波と波の間隔は、文字部分との間隔より狭いはずなので、間隔が最も広い２箇所のピークから先程のようにしきい値を求めれば、背景がグラデーション、文字がグラデーションなどの場合でも正確に２値化が行えた。

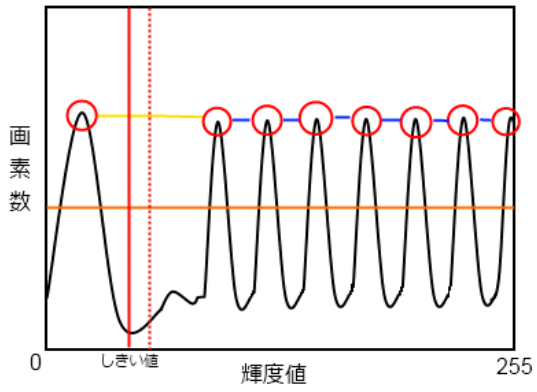

図７　グラデーション入り画像の２値化

今回の文字認識では、文字の周りを「輪郭追跡」により１周し、進んだ向きを記録し形を数値化した。まずは、図８のように左上から横向きに画像をスキャンしていき、画像の中で最も上にある黒のピクセルを見つける。

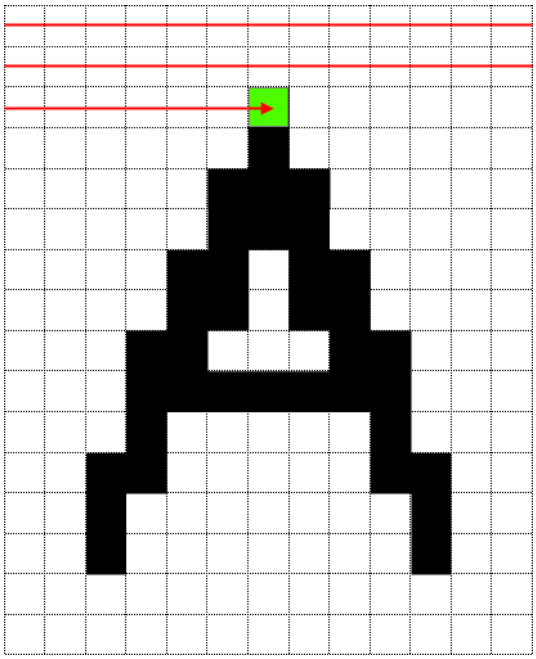

図８　文字の上の画素の検出

進む方向を図９のように番号を振り、図10 のように文字の外周を数字で方向を記録しながら反時計回りに辿っていく。

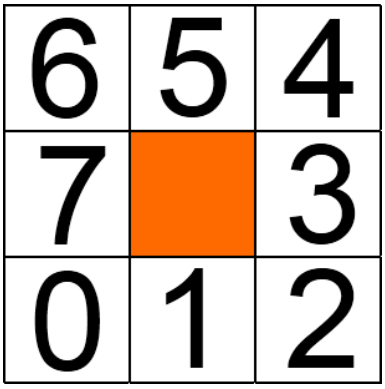

図９　進行方向に割り当てた数字

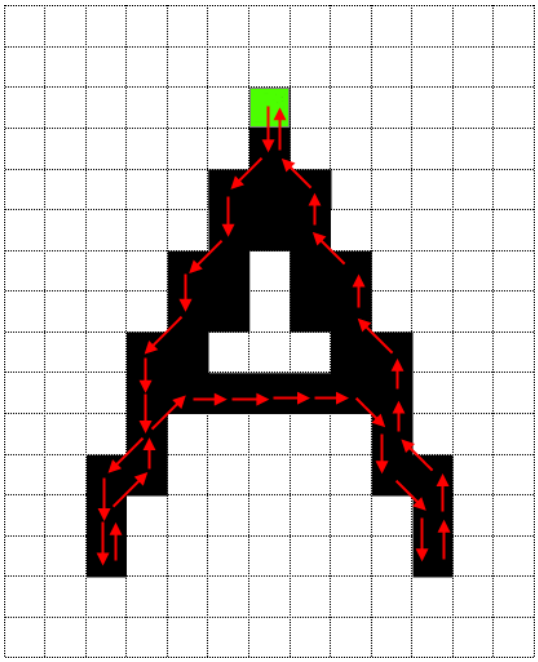

図 10　輪郭追跡

このとき、例えば下（1）に進んだ次の方向に、左上（6）や左（7）が選ばれることは無い。なぜなら、下（1）が選ばれたということは、すでにそのときの左（7）と左下（0）はチェックされ除外されており、下に進んだ後にも選ばれることはない。

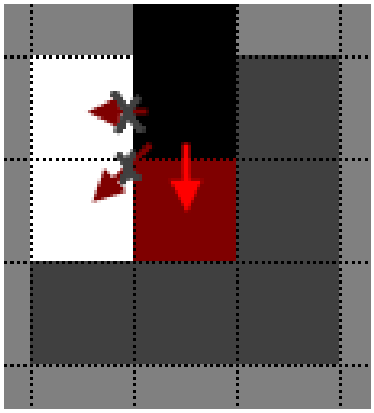

図 11　下に進んだときに
白とわかった場所

これをすべての方向で試したところ、ひとつ前の向きの時計回りに 3 つ回した向きには進まないことがわかった。よって、輪郭の向きは、ひとつ前の向きの時計回りに 2 つ回した向きから反時計回りにチェックしていけば効率的であることがわかった。

こうして文字の周りを一周すると、図 10 のＡの場合、形のデータは
10101011011545433332121555655656565
と数値化できる。しかし、このままでは全く同じ形で登録されていないと同じ文字だと判定することができないので簡略化を行う。まず、数値をある程度揃えるため、1 つ前の数字との差が 2 未満の場合、1 つ前の数字に合わせていく。先ほどの例では 2 文字目の「0」は一つ前の「1」との差が 2 未満なので、一つ前の「1」に合わせて 1 となり、全体を同じように処理すると
11111111111555533333111555555555555
となる。さらに、同じ数字が連続している部分は、一つ前の数字と同じ数字を消していく。例えば、2 つ目の「1」はひとつ前の「1」と同じ数なので削除、というようにしていく。これで、図 10 のＡの形は「15315」、要するに線の進む向きが下、上、右、下、上とあらわせるようになる。この簡略化により、例えばすごく縦に長いＡでも、簡略化したときに数値が「15315」にさえなればＡと読むことができるようになる。

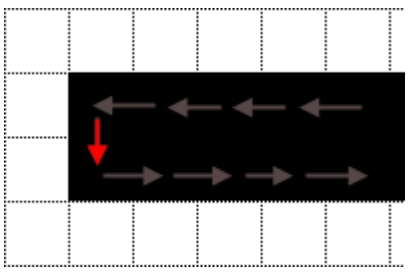

図 12　線に太さがあった時

輪郭追跡の際、図 12 のように線に太さがあり、濃い赤の矢印の部分のように文字の形に含まれていると、簡略化しても違う形とされてしまい、うまく読み取れなくなってしまう。そこで、線の中心 1 画素分だけを残す「細線化」を行う。今回は、細線化方法の一つである、参考文献[4]に記載されている「田村の方法」を元にして、細線化アルゴリズムを開発した。田村の方法では、注目画素が黒である場合に、注目画素を中心とする 3×3 画素の並びが図 13 のパターンの場合、中心画素を除去する。これを全画素で行い、除去した画素があるなら図 14 のパターンで同じことを行う。除去した画素が無ければそこで細線化処理を終了する。その結果が図 16 である。

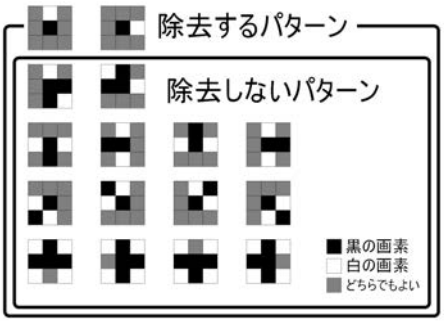

図 13　除去パターン 1

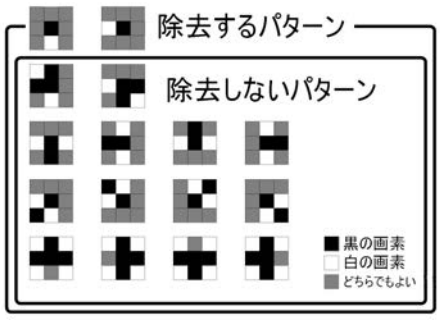

図 14　除去パターン 2

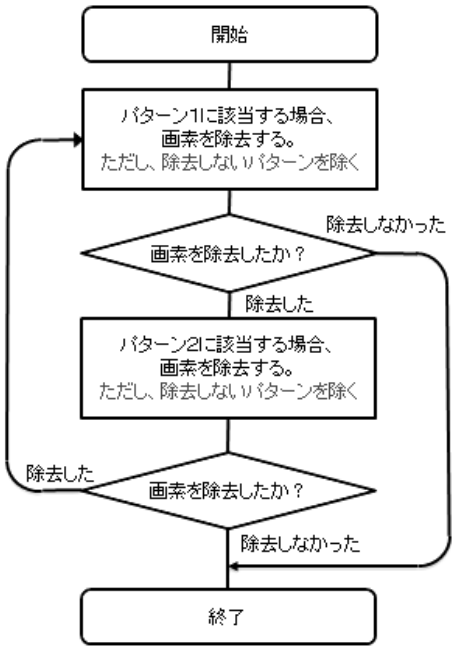

図 15　処理フローチャート

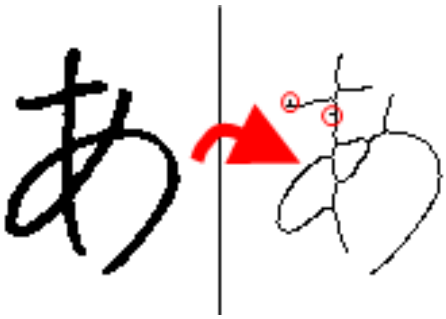

図 16　田村の方法による細線化

図 16 の結果をよく見ると、赤丸で囲われた所に線が割れてしまっているところがある。これらは元の線の淵が荒れており、図 17 のように、すこし出ている部分が「線の端」であると判断されてしまったため、除去されなくなってしまっている。

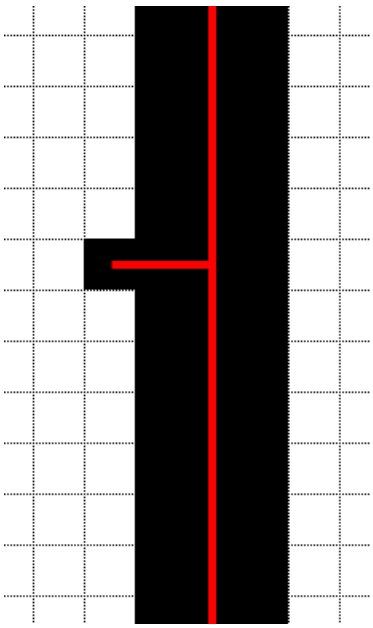

図 17　細線化の失敗例

そこで、注目画素を中心とする 3×3 画素の並びが 1 ピクセル分の分岐線（図 18）である時は無条件で除去されるようにした。その結果、図 19 のように、線の分岐は解消された。

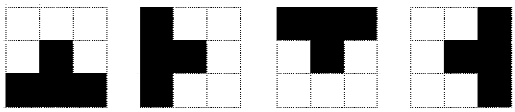

図 18　無条件で除去されるパターン

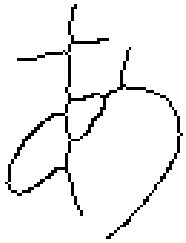

図 19　改善後の細線化結果

＜分かれている文字の連結＞

文字には「い」や「こ」などいくつかの形が集合しているものがある。そのような「分かれている文字」も認識できるようにするため、連結させる方法を考えた。文字の別々の状態での配置は記録されているので、それぞれの距離を求め、近いものを含めて一つの文字とし、データベースの文字と比較した。

例えば図 20 の「い」の場合、２つの形の両方が丁度入る四角形の中心を求め、それぞれの形の中心はどの方向に離れているかでどのように並んでいるかを求める。離れている方向は記録時に上下左右の 4 方向にし、例えば「離れている方向を調べたい部分」が文字全体の中心の左上にある場合は横方向と縦方向のどちらにより離れているかで「左」と「上」のどちらにあるのかを判断している。図 17 では、水色の部分が「左」、緑色の部分が「右」にあると判断できる。これを一つの文字とし、より上にあるものを左から順番に並べて登録し、同じようにして登録された判定用データと比較することで、分かれている文字を読み取ることができる。

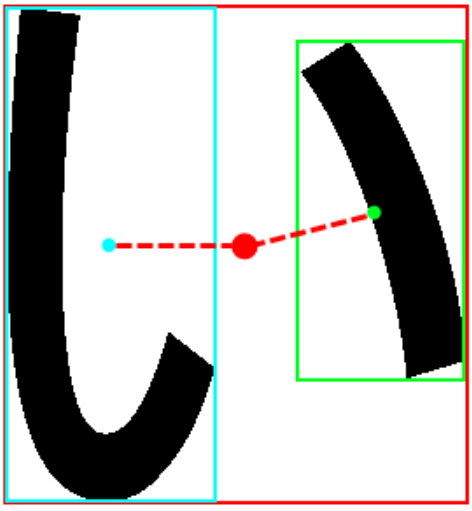

図 20　文字の連結

ただ、漢字などには、多くの形が集合している文字がある。１つの形のみで構成されている文字も多く存在するので、どこまでを一文字とするかの判定が必要であると感じた。

まず、画像内のすべての形の幅、高さのそれぞれの平均を求め、ある２つの形を繋げた時、その幅、高さに近づけば連結する、というようにした。この処理を繰り返せば、細かく分かれている部分は連結できるようになったが、画像内に「連結せず、小さい形」が含まれていると、ひらがなの「け」など、大きめの形同士の連結ができなくなってしまう。そこで、他の数字で何度か試したところ、幅、高さそれぞれの基準値を、「それぞれの平均と最大」の平均、にしたときうまく連結が行えた。

＜文字の形の一致度の計算＞

文字の形を簡略化しても、字の癖が大きすぎたりすると、全く同じ形のデータにはならないことがある。そこで、形のデータが「完全に一致しているかどうか」ではなく「どれだけ一致しているか」という判定方法をとることにした。

登録されている文字の形データは癖や歪みが無いとすると、同じ文字であれば、読み取った文字の形データの数値の数は登録されている文字の形データの数値の数より多いもしくは同じであるはずである。そこで、読み取った形データの数値の数が、ある判定用データの数値の数より多い場合、データの中のいくつかの数値を除けば判定用データと一致すれば、「そのときの判定用データの数値の数/読み取ったときの数値の数」を一致度とし、その一致度が最も高い判定用データを、読み取った文字の読みとするようにした。

＜斜めになった文字の認識＞

文字が斜めになっているとき、判定時に別の形とされてしまうことがある。そこで、判定するときに、スタートの値を合わせ、そのずらした分をその後の数も全てずらして判定する。

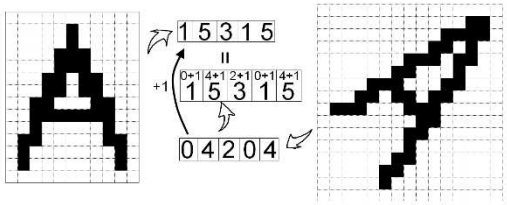

図 21　斜めの文字の認識

例えば図 21 の場合、左の「A」の形のデータ「15315」が登録されていて、右の斜めになった「A」を読もうとしているとすると、右のAの形のデータは「04204」であり登録された形と違っていて、「A」と判定されない。そこで、読み取ろうとしている文字の初めの向きを判定用データの初めの文字に合わせる。このとき、反時計回りに１つ分ずらしているので、残りの数も全て１つずつずらし、「15315」となる。すると、登録された「A」の形のデータと等しくなり、Aと読み取ることができる。

＜文字の形データのループ化＞

ひらがなの「め」など文字の上に複数の線が出ている場合、どの線が一番上に出ているかによりスタートとされる部分が変わってしまう。それ以外にも、文字のバランスや向きによっても、スタート値となる場所が変わってしまうこともある。そこで、

文字の形データをループ化し、判定時に読み取った形のデータのループを判定用データのループと回転させながら比較するようにした。

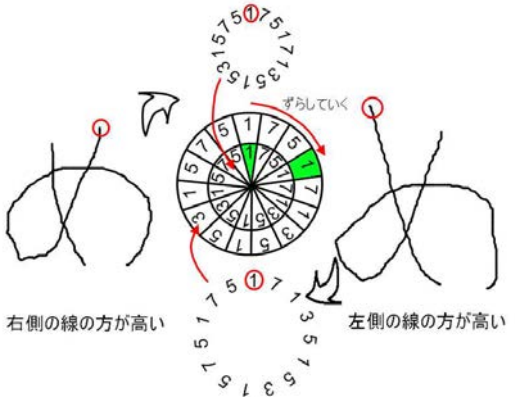

図 22　文字データのループ化

例えば図 22 のような「め」の例では、右側の線の方が高い方が登録されていたとすると、左側の線の方が高い方の形のデータは、数値の順番は同じなのに違う形であると判定されてしまう。そこで最初と最後を円形につなぎ、1 つずつずらして比較していき、判定を行うようにした。こうすればより少ない判定用データで様々な形に対応できるようになり、さらに斜めの文字の認識方法と組み合わせれば、たとえ文字が逆さになっていても文字を読み取ることができる。

＜文字の軽量な登録＞

文字は日本語だけでもかなりの数があり、初めからすべての文字を登録するのは困難である。また、人によっては線の始まりと終わりに字の癖がある場合があり、簡略化しきれず読みとれないことがある。そこで、読み取れない文字があったときに、人が登録できるようにした。また、保存するとき文字データは文字の読みと輪郭追跡で数値化した文字の形をコンマで区切ることで 1 つの文字列型変数に入れ、テキスト形式で保存した。テキスト形式で保存することで、ひらがな 50 音の文字の形を全て登録したデータが 10KB 程度とかなり小さく収めることができた。

＜学習機能＞

文字を手動で登録できるようになったが、読ませるたびに人が登録していては機械で文字認識をしているとはいえないので、自動で読めていなかった文字を登録する「学習機能」が必要であると考えた。しかし、読めていない文字を何の情報もなく覚えるのは不可能であり、いくらかのデータから考えさせる必要がある。そこで、前後の文字を参考にして読めていない文字の推測を行わせる方法を考えた。

＜文字の推測＞

人は、崩れていて読めない字を文章中で見つけたとき、前後の文字が読めていれば、知っている言葉から推測して読むことができる。これを機械で実現させれば、読み取れなかった文字や間違った判定がされた文字の数を減らすことができると考えた。

推測のために、まず単語のデータベースを作り言葉を大量に登録する。そして、推測で訂正したい文字列を登録したすべての単語と比較し、一致度を調べた。一致度は、（文字の一致度）×（文字数の一致度）とした。文字の一致度は、単語の初めの文字と最後の文字が一致で 4、それ以外の文字が場所も揃っていて 2、場所が違っていて 1 とし、「(その合計）÷（すべてあってい

たときの合計)」とした。文字数の一致度は、「1÷（文字数の差＋1)」とした。これで、両者ともに完全一致で 1 となる一致度が求められるようになった。その一致度が最も高かったものが正しい読みであるとし、それに置き換えるようにした。

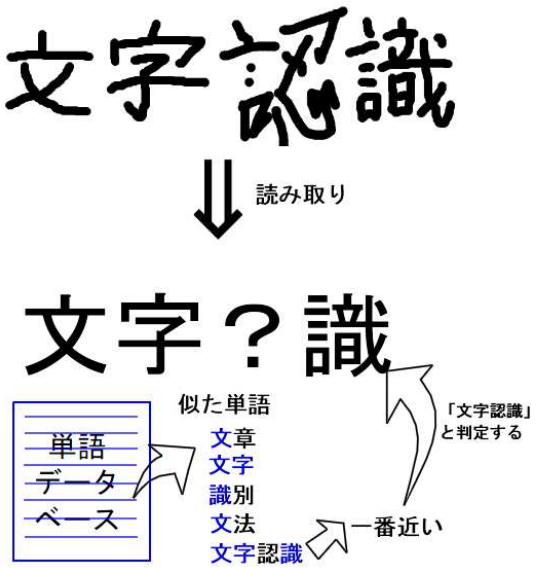

図 23　文字の推測補正

例えば、図 23 の「文章」と「文字？識」という 2 つの言葉の一致度は、文字の一致度が、一文字目のみが一致で 4、全てあっていた時は 2 文字なので 8 なので 1/2 となり、文字数の一致度が、文字数の差が 2 文字なので 1/3 となる。よって一致度は 1/6 となる。このように比較していくと、「文字認識」という言葉が最も一致度が高くなり、「文字？識」が「文字認識」に校正される。

＜学習機能（つづき）＞

この「文字の推測」を行い校正されたとき、一定条件を満たしていれば自動で文字の登録を行うようにした。条件として、まず文字数が同じであることが必要だと考えた。ノイズを誤認識して文字数が変わってしまっていることも考えられるが、その場合でもどれがノイズであるかの見分けが難しく、「ノイズの形」が文字の形として登録されるなどの誤登録を避けるためである。また、一致度が 7 割を超えていることを登録する条件とした。何度か自分で試したところ、文字数があっていて、なおかつ文字が 7 割以上合っていれば、誤登録されることはほとんど無かった。これにより、7 割が読みとれていれば、使っているだけで精度が上がっていくようになった。

＜日本語の文を単語で区切る＞

文字の推測校正を行うためには、文が単語ごとに区切られている必要がある。英語のように単語ごとに隙間がある言語では、その隙間を認識すれば区切ることが可能だが、日本語のように単語ごとに隙間のない言語ではそれができない。そこで、日本語の文章を登録された単語を元に区切る方法を考えた。

まず単語が間違っていないとして、文全体から一文字ずつ減らして登録された単語と比較していき、一致した中で最も文字数が多いものを、そこにある単語であると考えるようにした。これで大抵は間違わずに短時間で区切ることができた。

また、判定する文字数を 1 文字まで縮めてもどの言葉とも一致しなかった場合、もう一度全体から文字数を減らして比較していき、比較したときに一致度が 0.9 を超えていたら、その言葉をそこにある単語と考

える、というように徐々に基準の一致度を下げていくことで、間違った文字を含む単語が文中に含まれている場合でも、単語で区切り推測校正を行うことができるようになった。

４．まとめと今後の課題

今回、輪郭追跡により文字の形と位置を把握し、ある程度の精度で認識できるようになった。しかし、輪郭追跡では、外側の形を辿っているため、内側の線は無視されている。今後、文字の形の把握方法を改良し、文字の中の線を認識させることで、さらに精度を上げていきたいと思う。また、今回の研究で、文字の形を登録すればどんな言語の文字でも読み取ることができるようになった。そこで、今後翻訳機能を搭載し、当初の目的である「看板の文字を写真にとって翻訳する」ことができるようにしたいと思っている。

５．参考文献

[1] Visual　Basic　中学校
http://homepage1.nifty.com/rucio/main/main.htm

[2] DOBON.NET
http://dobon.net/index.html

[3] C#と VB.NET の入門サイト
http://jeanne.wankuma.com/

[4] 画像処理ソリューション
http://imagingsolution.blog107.fc2.com/blog-entry-151.html

６．謝辞

今回の研究にあたり指導してくださった顧問の米田先生、ありがとうございました。また、サイエンス研究会の先輩方にもご指導、ご協力いただきました。ありがとうございました。