麻雀点数計算プログラム 注釈書

14EC004 飯田頌平

2016 年 2 月 22 日

1　はじめに

本稿は吉田さんの麻雀点数計算プログラム [1] の注釈書にあたる。

このプログラムはカメラで麻雀の手牌を撮影すると点数を自動で計算するというものであり、大まかにサーバ、画像認識、点数計算の三つのモジュールに分かれている。

このうち本稿では画像認識の部分について触れる。画像認識の実装ファイル TemplateMatching.scala およびユーザ定義関数の実装ファイル package.scala を付録に掲載するので、参考にしながら読むこと。

2　画像認識のフロー

画像認識はパターンマッチングによって行われている。よって、最初に訓練データからテンプレートを生成し、その後テストデータとテンプレートを照合させて結果を求める。まずはテンプレートの生成手順を示す。

1. 雀牌と背景を判別し、雀牌だけを切り取る
2. 雀牌を黒と白で二値化する
3. 雀牌から牌ひとつあたりの縦幅と横幅を求める
4. 白と判別された誤差（牌の隅）を黒く塗りつぶす
5. 雀牌から牌のシーケンスを抽出する
6. 雀牌の絵柄部分の輪郭を得る
7. 雀牌のグレースケールからテンプレートを生成する

詳細な説明については後述するとして、次に照合の手順を示す。

1. 手牌と背景を区別し、手牌だけを切り取る
2. テンプレートから各牌を回転させたイメージを得る
3. 手牌とテンプレート（及び回転イメージ）を照合させ、座標の形で一致箇所を得る
4. 座標を基に手牌を判定する
5. 門前か否かを判定する

こうして手牌を認識することができた。次章以降ではこれらの流れの詳細について述べる。

3　テンプレートの生成

テンプレートの生成にあたって、まずは元画像を用意する。以下にその要件を示す。

- 雀牌を 4*9 の矩形上に隙間なく並べる
- 牌の位置は固定である
  一萬からはじまる SEQ が予め定められている
- 背景は一色のものを選ぶ
- 雀牌と背景以外のものを除外する
- 背景もできるだけ写り込まないようにする
- 撮影時に、各牌が等しい高さと幅を持つよう注意する

背景には麻雀マットなどを用いれば良い。また斜めから撮影された画像からは正しいテンプレートが得られないことに気を付ける。

元画像はプログラムへ Mat 型で渡される。Mat 型は opencv[*1]のライブラリが提供する行列の型であり、opencv ではこの型で画像データを扱う。createTemplate の引数 mat が元画像のデータを示す変数である。

mat に対して最初に行われる処理は、定数 MaxResolution のサイズに近づくよう resize で画像サイズを修正する。大きい画像であると処理時間が増えてしまうため、これによって効率化する。

次に雀牌と背景を識別し、雀牌だけを切り取る crop を実行する。crop は元画像を引数に取り、Buffer 型のシーケンスで識別結果の画像を返す。（なお Buffer はタプルであり、画像だけでなく輪郭の二次元配列も返しているが、この値は createTemplate では参照しない。）

crop の実装は、まず findContours によって元画像を輪郭で分離し、minAreaRect で矩形を得て、warpAffine で矩形の傾きを底面と垂直になるよう修正し、getRectSubPix で矩形領域のピクセル値を得ることで、入力画像を輪郭毎に分離した上で補正をかけた画像を得るようになっている。

こうして得られた分離後の画像のシーケンス（返り値は Buffer 型であることに注意）から headOption を使うことで先頭要素を取り出せる。findContours では大きさ順に画像をソートする作用があるので、先頭要素には一番大きな画像が入る。一番大きな画像が雀牌であるという前提であるなら（元画像の背景部分が大きすぎるとこの前提は崩れる）、この先頭要素は雀牌であるため、雀牌のテンプレートは引数 template に束縛される。

threshold によってテンプレートを黒と白で二値化し、width と height に牌ひとつあたりの幅と高さを代入する。このとき雀牌の並びが 4*9 でなかったとき、想定外の値を取ってしまうので注意すること。

こうして得られた二値化後の画像について見てみると、牌の模様が白、牌の背景が黒で表されていることが分かる。しかしそれと同時に、牌と牌の間なども白く判定されてしまっている。牌の模様以外が白と認識されてはパターンマッチに失敗するため、次はこの誤判定部分を黒く塗りつぶす。それには floodFill を使う。floodFill は座標を指定し、連結部分を指定した色で塗りつぶす関数である。これを牌と牌の間にすべてに対して実行することで、誤判定が塗りつぶされる。

そして牌の模様だけを抽出する。findContours によって再び輪郭ごとの分離をかければ、模様だけを抽出することができる。四筒のように模様が複数に分離されてしまうケースの場合は、convexHull によって結合する。最後に、得られた模様の中で、もっとも面積の大きなものを size に代入する。マッチングを行う際にはこのサイズを基準にするためである。

ここまでで二値化の済んだ雀牌のテンプレート画像が得られた。今度はこれを牌ひとつひとつに分離して、牌ごとのテンプレートを得る。

それには grid を用いる。grid は submat を実装に含み、元画像から矩形範囲を抽出できる関数である。grid で牌を 4*9 に区切り、zip で牌ごとに模様の輪郭の情報を付与し、それらに flatMap を挟んで getRectSubPix をかけることによって、輪郭だけが抽出されたすべての牌のテンプレートを得ることができる。なお、このテンプレートは白黒ではなくグレースケールである点に注意。

以上でテンプレートの生成が完了した。

4　テストデータとの照合

照合は関数 recognize によって実装されている。この引数 mat に手牌の画像データを、templates にテンプレートの画像データを渡す。

手牌のデータは crop によって背景から切り離される。しかしテンプレートを crop で切り離したときと違い、この場合は牌のデータが複数に分かれる。手牌には門前と鳴きがあり、鳴いた牌は手牌と離して置くためだ。

手牌のデータは collect によって hand に束縛される。テンプレートに向きを回転させた牌を加えて

[*1] https://github.com/bytedeco/javacv

tiles に代入すると、いよいよ go でパターンマッチを行う。

go の内部では、まず各テンプレートに対して OpenCV のテンプレートマッチング関数たる matchTemplate を呼ぶ。その結果が result へと代入されるので、minMaxLoc を通して最もスコアの高い位置を取得し、テンプレートと位置のタプルを locs に加えてゆく。この手順をすべてのテンプレートに対して実行する。

参考文献

[1] Sanshiro Yoshida. mahjongs. https://github.com/halcat0x15a/mahjongs