

# マルチメディア信号処理 第1回レポート課題

河野達彦 小寺健太

2017年6月14日

## 1 課題 1.1

### 1.1 アルゴリズム概要

私たちの班は C++ で以下のようなプログラムを作成した。

---

**Algorithm 1** 拡大・縮小・回転に対応するパターンマッチング

---

**Require:** テンプレート画像 10~15 枚, 探索対象画像 (image.pgm)

**Ensure:** 各テンプレート画像の中心位置座標, 拡大倍率, 回転角  
画像の読み込み

**for** image.pgm を左上からスキャン **do**

**if** 画素が黒でない **then**

        target  $\leftarrow$  8 近傍で接続しているオブジェクトを切り抜く

**for** 各テンプレート画像に対して **do**

            オブジェクトの画素値の合計から各テンプレートの拡大倍率を算出

**for** 回転角 = -90 to 90 **do**

                template  $\leftarrow$  拡大倍率と回転角を反映したテンプレート画像を用意し切り抜く

                diff  $\leftarrow$  (target と template の画素値の差)<sup>2</sup>の和/比較画素数

**end for**

**end for**

        target に対して最小の diff を与える template 番号, 中心位置座標, 拡大倍率, 回転角を出力

**end if**

**end for**

---

### 1.2 実装の詳細

#### 1.2.1 画像の読み込み

画像に関するデータをメンバにもつクラスを用意し, そのメソッドとして Readdata() を作成した. Readdata() はファイルパスを引数とし, bool 値を返す. メソッド内で画素値データを 2 次元 vector の data に格納し, またその画素値が走査済みかどうかを保持する 2 次元 vector の visited を作成する (makevisited()). 与えられた pgm ファイルは画像の幅や高さ, 各画素値はスペース区切りで羅列されている. よって ifstream で画像を開いたのち, 演算子を用いることでうまく読み込みができた. template 画像の番号は連続であるがフォルダによって画像の数が異なるため, ファイルの取得に失敗した場合は false を返す関数として実装し全ての template 画像を読み込むようにした.

```
// 画像の読み込みファイルが存在しない場合 false を返す
bool readdata(string filename){
    string str;
    ifstream fin(filename);
    if(fin){
        fin >> str; fin.ignore();
        if(str != "P2") cout << "file format error" << endl;
        getline(fin, str); // コメント読み捨て
        fin >> W >> H;
        data.resize(H);
        for(int i=0; i<H; i++){
            data[i].resize(W);
        }
        fin >> str; //255
        int h=0, w=0;
        while(fin >> str){
            data[h][w] = stoi(str);
            w = (w+1) % W;
            if(w == 0) h++;
        }
        makevisited();
        return true;
    }
    else return false;
}
```

### 1.2.2 画像の切り抜き

Algorithm1 において切り抜きと表現した操作は `trimming()` メソッドによって行われる。 `trimming()` は黒でない画素座標を引数とし、8近傍で連結するオブジェクトに内接する長方形を考えその左上点の座標と長方形の幅および高さを計算する。それに加え拡大倍率算出に使われるオブジェクトの画素値の合計も同時に計算する。連結部の走査は Queue を用いた幅優先探索アルゴリズムで実現した。これに伴い、画素値が走査済みかどうかを保持する2次元 vector である `visited` が必要になる。 `visited` の要素は bool であり、 `Raddata()` 内で黒でない画素が格納されている座標に `false` を格納した。1度訪れた画素座標には `true` が格納される。

```
// 未踏はfalse, 訪問済みtrue
// が入っている場所は0true, ついでに総面積S画素値の合計も算出(=)
void makevisited(){
    visited.resize(H);
    for(int i=0; i<H; i++){
        visited[i].assign(W,false);
    }
    S = 0;
    for(int h=0; h<H; h++){
        for(int w=0; w<W; w++){
            if(data[h][w] == 0){
                visited[h][w] = true;
            }
            else{
                S += data[h][w];
            }
        }
    }
}

// 内接する四角の左上端と幅, オブジェクトの画素値の合計を計算
void trimming(int h, int w){
```

```

queue<Point> que;
int dp[3] = {-1,0,1};
que.push({h,w});
visited[h][w] = true;
S_trim = data[h][w];
int max_h=0, min_h=MAX, max_w=0, min_w=MAX;
while(que.size()){
    Point p = que.front(); que.pop();
    max_h = max(max_h, p.h);
    min_h = min(min_h, p.h);
    max_w = max(max_w, p.w);
    min_w = min(min_w, p.w);
    for(int i=0; i<3; i++){
        for(int j=0; j<3; j++){
            Point np = {p.h + dp[i], p.w + dp[j]};
            if(np.h>=0 && np.h<H && np.w>=0 && np.w<W){
                if(! visited[np.h][np.w]){
                    que.push(np);
                    S_trim += data[np.h][np.w];
                    visited[np.h][np.w] = true;
                }
            }
        }
    }
}
upleft.h = min_h;
upleft.w = min_w;
H_trim = max_h - min_h;
W_trim = max_w - min_w;
}

```

### 1.2.3 ノイズ処理

ノイズが非常に多い画像に対するマッチング処理を行うため、ノイズ除去を行う関数を実装した。ノイズ部分は白か黒、すなわち画素値は 0 or 255 であるため、この二種類の値を取っている画素について処理を行うことでノイズ除去による情報量の減少を抑えようと考えた。周辺 8 近傍の平均値がしきい値を上回った・下回った場合にはそれぞれ画素値を 0, 255 とする実装を行った。また、我々のパターンマッチングのアイディアは黒でないピクセルから、隣接する白でない画素をなぞることで画像を切り抜くが、この切り抜いた部分の大きさが小さかった場合はノイズと判断する様に実装したため周辺が黒い部分での白のノイズの除去は一定の成果が出た時点で完成とした。

```

// 画素情報と周辺平均を投げると条件によってノイズの画素を綺麗にする
void remove_noise_pixel(vector<vector<int>>& nonoise_data,int h,int w,int peripheral_ave){
    if(data[h][w] == 0 && peripheral_ave > REMOVE_NOISE_WHITE_THRESHOLD){
        nonoise_data[h][w] = 255;
    }
    else if(data[h][w] == 255 && peripheral_ave < REMOVE_NOISE_BLACK_THRESHOLD){
        nonoise_data[h][w] = 0;
    }else{
        nonoise_data[h][w] = data[h][w];
    }
}

// ノイズ除去
void remove_noise(){
    int peripheral_ave = 0;

    vector<vector<int>> nonoise_data;
    nonoise_data.resize(H);

```

```

for(int i=0; i<H; i++){
    nonoise_data[i].resize(W);
}

for(int h=0; h<H; h++){
    for(int w=0; w<W; w++){
        // 四边上の端は周辺画素ずつは取れないはみ出すので対策1()
        if(h == 0){
            // 上辺
            if(w==0){
                peripheral_ave = (data[h][w+1] + data[h+1][w] + data[h+1][w+1]) / 3;
            }
            if(w==W){
                peripheral_ave = (data[h][w-1] + data[h+1][w-1] + data[h+1][w]) / 3;
            }
            else{
                peripheral_ave = (data[h][w-1] + data[h][w+1] + data[h+1][w-1]
                    + data[h+1][w] + data[h+1][w+1]) / 5;
            }
            remove_noise_pixel(nonoise_data,h,w,peripheral_ave);
        }else if(h == H-1){
            // 下辺
            if(w==0){
                peripheral_ave = (data[h-1][w] + data[h-1][w+1] + data[h][w+1]) / 3;
            }
            if(w==W){
                peripheral_ave = (data[h-1][w-1] + data[h-1][w] + data[h][w-1]) / 3;
            }
            else{
                peripheral_ave = (data[h-1][w-1] + data[h-1][w] + data[h-1][w+1]
                    + data[h][w-1] + data[h][w+1]) / 5;
            }
            remove_noise_pixel(nonoise_data,h,w,peripheral_ave);
        }else if(w == 0){
            // 左辺
            peripheral_ave = (data[h-1][w] + data[h-1][w+1] + data[h][w+1]
                + data[h+1][w] + data[h+1][w+1]) / 5;
            remove_noise_pixel(nonoise_data,h,w,peripheral_ave);
        }else if(w == W-1){
            // 右辺
            peripheral_ave = (data[h-1][w-1] + data[h-1][w] + data[h][w-1]
                + data[h+1][w-1] + data[h+1][w]) / 5;
            remove_noise_pixel(nonoise_data,h,w,peripheral_ave);
        }
        // 隣接周辺画素がすべて取れるエリア
        else if(data[h][w] == 0){
            // 黒画素
            peripheral_ave = (data[h-1][w-1] + data[h-1][w] + data[h-1][w+1]
                + data[h][w-1] + data[h][w+1] + data[h+1][w-1] + data[h+1][w]
                + data[h+1][w+1]) / 8;
            if(count_black(h,w) < 4){
                nonoise_data[h][w] = 255;
            }else if(peripheral_ave > REMOVE_NOISE_WHITE_THRESHOLD){
                nonoise_data[h][w] = 255;
            }else{
                nonoise_data[h][w] = data[h][w];
            }
        }else if(data[h][w] == 255){
            // 白画素
            peripheral_ave = (data[h-1][w-1] + data[h-1][w] + data[h-1][w+1]
                + data[h][w-1] + data[h][w+1] + data[h+1][w-1] + data[h+1][w]
                + data[h+1][w+1]) / 8;
            if(count_white(h,w) < 4){
                nonoise_data[h][w] = 0;
            }else if(peripheral_ave < REMOVE_NOISE_BLACK_THRESHOLD){
                nonoise_data[h][w] = 0;
            }
        }
    }
}

```

```

        }else{
            nonoise_data[h][w] = data[h][w];
        }
    }else{
        nonoise_data[h][w] = data[h][w];
    }
}
}
data = nonoise_data;
}

```

#### 1.2.4 拡大倍率と回転への対応

実装当初は画素数の比を元に template 画像の拡大倍率を算出していたが良い結果が得られなかった。某氏の助言により画素値の総和の比を元に算出したところ非常に良い精度で得られた。image.pgm 中の注目しているオブジェクトに対して切り出しを行なっているため画素値の総和は既知である。さらに各 template 画像についても読み込み時に既に算出している。したがって総当たりをすることなく、各 template 画像が注目オブジェクトに一致するとしたときの拡大倍率を算出することが可能である。一方、回転については-90 度～90 度の 180 通りを総当たりした。

#### 1.2.5 target と template の比較方法

今回の課題において要となるのは image.pgm 中の注目しているオブジェクト (target) と拡大倍率、回転角を反映した各種 template 画像 (template) を何を基準にして比較するかという点だろう。本プログラムでは target と template の両方に対して内接する長方形で切り出しを行うことで解決した。もし target に対して適切な template 画像、拡大倍率、回転角の template であれば、切り出した後の画像は等しくなるはずである。したがって単純に左上のピクセルから比較すればよいことが分かる。この手法を実現するためには template の画像データを用意し切り出しを行う必要がある。balance() 関数はある template 元画像クラス、拡大倍率、回転角を引数とし、それらを反映した新たな template 画像クラスを返す関数である。なお生成する画像は特に補完を行わず、写像後小数点以下を切り捨てた画素値を与えた。

```

// 入力画像を倍し度回転させた画像を返す scalerot
Image balance(Image templates, double scale, int rot){
    double theta = rot*M_PI/180;
    Image temp;
    temp.H = templates.H * scale;
    temp.W = templates.W * scale;
    // のオリジナル28*28画像から画素を持ってくるので拡大倍率によらず中心は常にtemplate(14,14)
    Point c = {templates.H/2, templates.W/2};

    temp.data.resize(temp.H);
    for(int i=0; i<temp.H; i++)
        temp.data[i].resize(temp.W);画像中心を中心として回転

    //
    for(int h=0; h<temp.H; h++){
        for(int w=0; w<temp.W; w++){
            double h_s = h/scale;
            double w_s = w/scale;
            int h_r = cos(-theta)*(h_s-c.h) - sin(-theta)*(w_s-c.w) + c.h;
            int w_r = sin(-theta)*(h_s-c.h) + cos(-theta)*(w_s-c.w) + c.w;
            if(h_r >= templates.H) h_r = templates.H-1;
            if(w_r >= templates.W) w_r = templates.W-1;
            if(h_r < 0) h_r = 0;
            if(w_r < 0) w_r = 0;

```

```

        temp.data[h][w] = templates.data[h_r][w_r];
    }
}
temp.makevisited();
return temp;
}

```

また比較の範囲は target と template が重なっている部分のみとした。これは切り出し後の幅と高さについて 2 画像の最小値をとることで実現できる。target と最も一致する template は、1 画素あたりの画素値の差の 2 乗が最小のものとした。某氏の指摘により比較領域の画素数で割ることに気づき精度が向上したことを記しておく。

### 1.2.6 特徴点抽出

手書きの数値画像の特徴点として、以下の項目が使えると考えた。

- 画素値の合計
- 画素値の上下の偏り
- 画素値の左右の偏り

画素値の合計とは、画像の画素の値を合計しただけのものである。単純な値ではあるが、文字の面積を図ることが出来るので特徴点として利用可能だと考えた。これはマッチング処理にて既に実装していた為、そのまま利用した。

また、画素値が左右、上下でどれくらい偏ってるかによって上下対照・左右対称等の特徴を反映できるのではないかと考え、実装し実験を行った。

## 1.3 実行環境と結果

以下は Macbook Pro(Intel Corei5 2GHz プロセッサ, 8GB メモリ) で実行した結果である。また実行時間は chorno ライブラリを用いて計測した。なおコンパイルは g++ -O3 -std=c++11 で行った。

### 1.3.1 image1

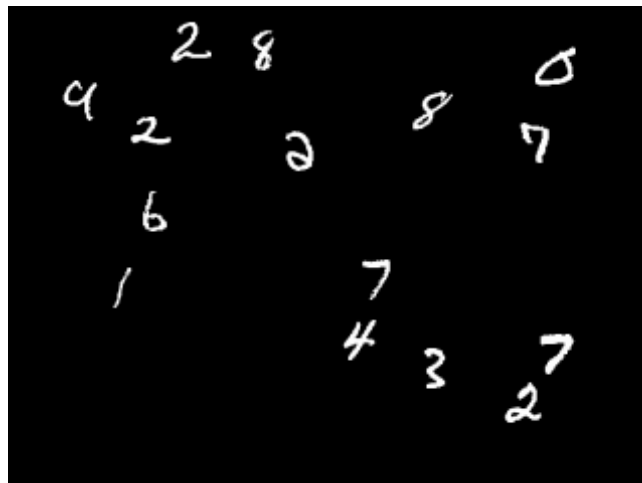


図 1 image1 解答と実行結果

実行時間は 1.351 秒であった。座標，回転角，拡大倍率の全てを正しく得ることができた。

表 1 image1 実行結果と解答

解答					実行結果				
画像	w	h	回転角	拡大倍率	画像	w	h	回転角	拡大倍率
template1	212	179	0	1	template1	212	179	0	1
template2	145	72	0	1	template2	145	72	0	1
template3	126	22	0	1	template3	126	22	0	1
template4	70	63	0	1	template4	70	63	0	1
template5	36	45	0	1	template5	36	45	0	1
template6	56	139	0	1	template6	56	139	0	1
template7	184	132	0	1	template7	184	132	0	1
template8	262	66	0	1	template8	262	66	0	1
template9	71	104	0	1	template9	71	104	0	1
template10	272	170	0	1	template10	272	170	0	1
template11	209	51	0	1	template11	209	51	0	1
template12	272	31	0	1	template12	272	31	0	1
template13	256	198	0	1	template13	256	198	0	1
template14	174	165	0	1	template14	174	165	0	1
template15	89	19	0	1	template15	89	19	0	1

表 2 image1 実行結果と解答

解答					実行結果				
画像	w	h	回転角	拡大倍率	画像	w	h	回転角	拡大倍率
template1	212	179	0	1	template13	174	165	0	1
template2	145	72	0	1	template2	85	14	0	1
template3	126	22	0	1	template3	127	20	0	1
template4	70	63	0	1	template4	265	28	0	1
template5	36	45	0	1	template5	34	43	0	1
template6	56	139	0	1	template6	208	48	0	1
template7	184	132	0	1	template7	63	62	0	1
template8	262	66	0	1	template8	260	62	0	1
template9	71	104	0	1	template9	144	68	0	1
template10	272	170	0	1	template10	69	102	0	1
template11	209	51	0	1	template11	182	131	0	1
template12	272	31	0	1	template12	58	137	0	1
template13	256	198	0	1	template13	253	194	0	1
template14	174	165	0	1	template14	272	172	0	1
template15	89	19	0	1	template15	212	177	0	1

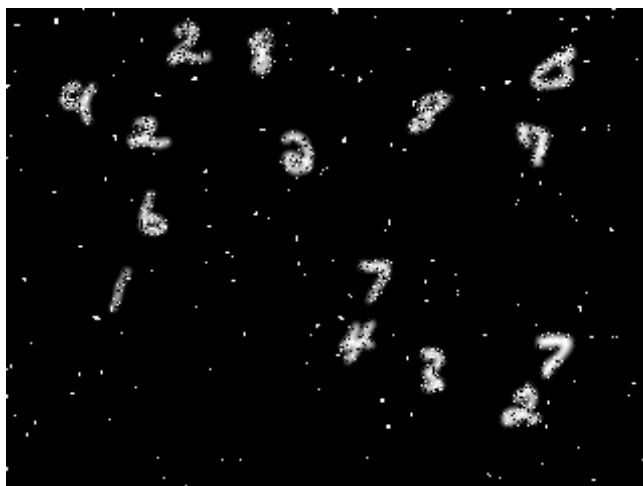


図 2 image2 ノイズ除去後の画像

### 1.3.2 image2

実行時間は 40 ミリ秒であった.

なお, この課題のみ回転角と拡大倍率は固定とした. template3,5,8,13 以外は全て誤認識となった.

### 1.3.3 image3

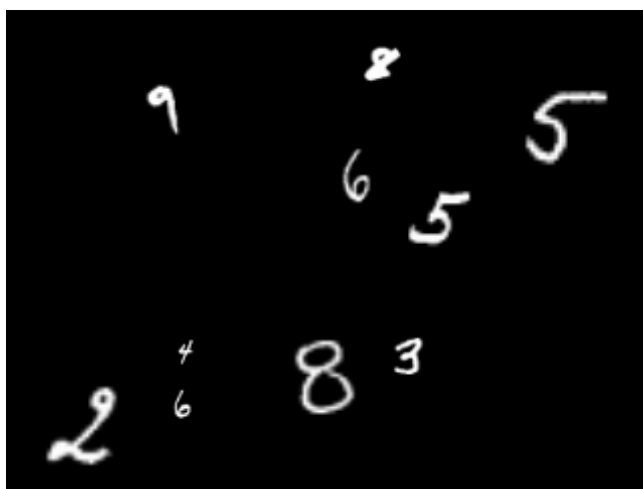


図 3 image3 回答と実行結果



表 3 image3 解答と実行結果

解答					実行結果				
画像	w	h	回転角	拡大倍率	画像	w	h	回転角	拡大倍率
template1	40	209	0	1.9	template1	38	207	-1	1.89766
template2	87	197	0	0.7	template8	86	196	-18	0.820375
template3	157	181	0	1.8	template3	156	180	3	1.79956
template4	89	169	0	0.6	template4	88	169	0	0.599469
template5	78	45	0	1.2	template5	76	43	3	1.20264
template6	186	26	0	0.9	template6	185	25	0	0.896784
template7	214	104	0	1.5	template7	212	102	-2	1.4978
template8	173	85	0	1.3	template8	172	84	-2	1.30326
template9	275	55	0	2	template9	274	54	-2	2.00127
template10	200	171	0	0.9	template10	199	170	-3	0.898823

実行時間は 0.879 秒であった。template2 が template8 と誤識別されてしまった。他のオブジェクトに関しては座標は 2 以内、拡大倍率は 0.01 以内、回転角は 3 度以内の誤差で検出できた。

#### 1.3.4 image4

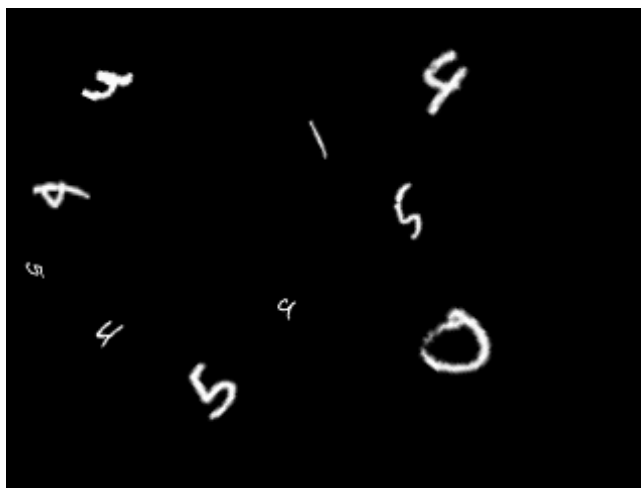


図 4 image4

表 4 image4 解答と実行結果

解答					実行結果				
template 番号	w	h	回転角	拡大倍率	template 番号	w	h	回転角	拡大倍率
template1	217	36	-35	1.6	template1	216	34	-33	1.59986
template2	50	37	65	1	template2	48	36	65	0.999354
template3	225	165	-38	1.6	template3	224	163	-37	1.6004
template4	50	161	-8	0.7	template4	49	160	-9	0.698452
template5	139	148	-11	0.5	template5	137	147	-22	0.498287
template6	25	91	87	1.4	template6	23	91	87	1.39994
template7	14	129	-69	0.5	template7	13	128	-78	0.49995
template8	155	64	29	1	template8	154	63	29	1
template9	199	101	55	1.1	template9	197	100	53	1.09943
template10	102	190	44	1.3	template10	100	189	41	1.29979

実行時間は 0.769 秒であった。全てのオブジェクトに関して十分な精度で検出できた。座標は 2 以内、拡大倍率は 0.01 以内、回転角は 11 度以内の誤差で検出できた。

### 1.3.5 image5

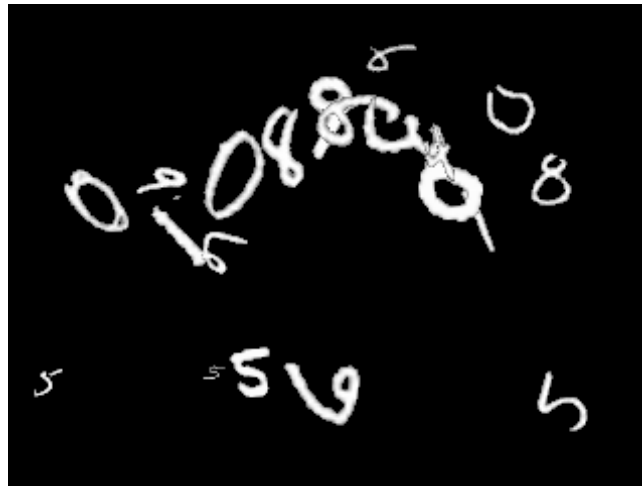


図 5 image5

20 個のうち正しく識別できたのは template7,8,9,12,17,20 の 6 個であった。オブジェクト同士が重なっている画像を識別することは今後への課題とする。

### 1.3.6 特徴点抽出

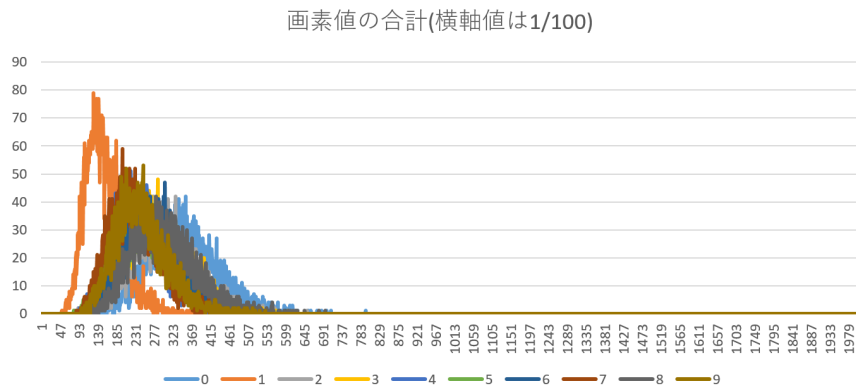


図6 画素値の合計

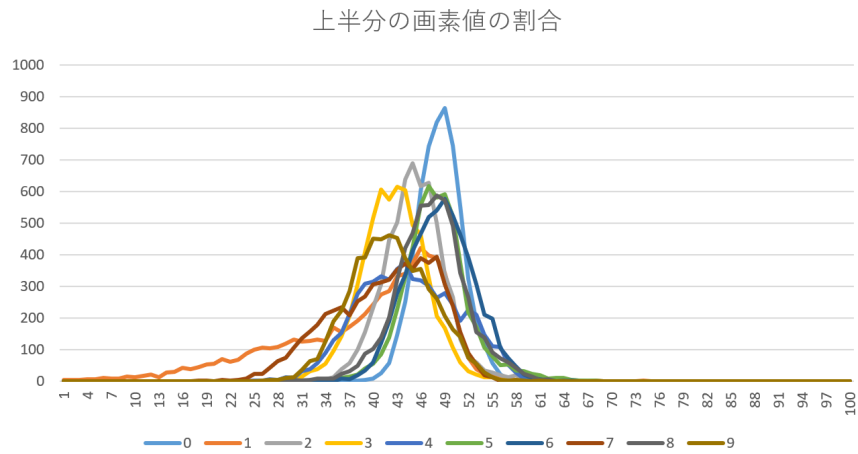


図7 上半分の画素値の合計

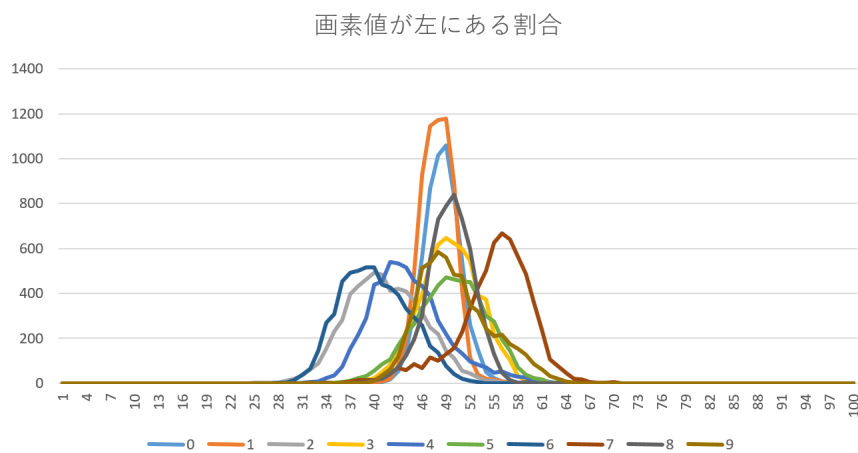


図8 左半分の画素値の合計

## 1.4 考察

### 1.4.1 計算時間

全ての拡大倍率, 回転角について総当たりをした場合の計算量は, image.pgm の高さ と 幅  $H, W$ , template 画像の高さ と 幅  $H_t, W_t$ , 拡大倍率  $S$ , 回転角  $R$ , template 画像数  $T$  を用いて

$$O(H \cdot W \cdot H_t \cdot W_t \cdot S \cdot R \cdot T)$$

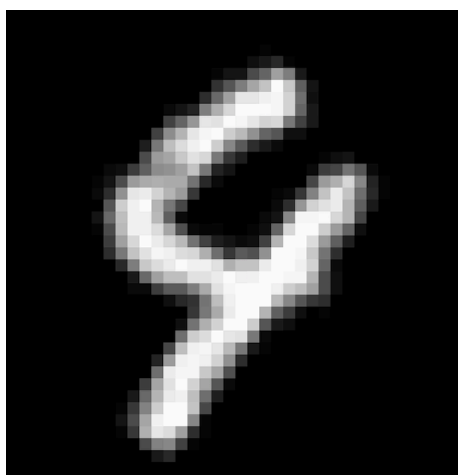
で表される. これはおよそ  $10^{13}$  ステップあり全テンプレートの走査に 10000 秒程度かかることが予想される. 一方今回私たちのプログラムは以下の計算量で走査を終える.

$$O(H \cdot W + H_t \cdot W_t + H_t \cdot W_t \cdot R \cdot T \cdot T)$$

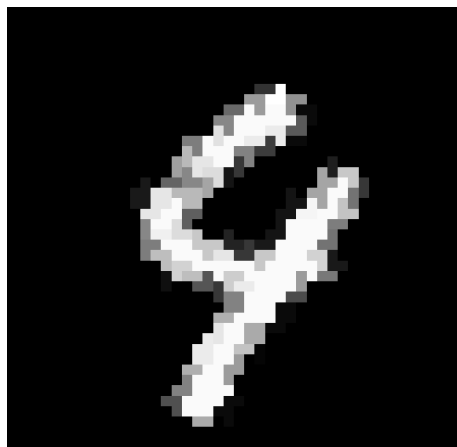
これはおよそ  $10^8$  ステップあり, 1/10 秒ほどで全テンプレートの走査を終えることができると考えられる. 上記の実験では 1 秒前後で終わることができ, 明らかな改善があったと言える.

### 1.4.2 精度

image3 における誤識別や, 出力結果の誤差について考察する. 図 5 は image.pgm 中のオブジェクトと本アルゴリズムで作られた template 画像である.



(a)image.pgm 中のオブジェクト



(b)template 画像に拡大・回転処理をした画像

図 9

このように明らかな補完処理の差があるため, 特に違いの少ない手書き文字の識別は非常に難しくなる.

## 2 感想

### 2.1 河野

僕自身が C++ でのコードの実装，また論理的にアルゴリズムを考えて実装するといった事には慣れていなかったため，チームメイトである小寺君へのアドバイスをやる余裕があまりなく，予定より全体的に手間取ってしまった．カプセル化や関数切り分けのポイント等を指示するなど，互いの経験値になることを考えていたが課題の処理に負われてしまったのは反省点である．だが，この課題のコードを作りっぱなしにせず提出後も一部修正を加える予定であるのでそれを通じて僕もプログラミングのレベルアップを図りたいと考えている．

### 2.2 小寺

河野さんがメインで実装を行うと簡単にできてしまいそうだったので，僕の実装をサポートをしていただくお願いを聞き入れてもらった．複雑ではないプログラムとはいえ，1 から実装するのは思ったよりも苦労した．今回の課題で最大の山場は target と template を比較する方法である．重心を使わずにできないかと考え，思いついた時は非常に嬉しかった．また画像の回転を実装する際，あくまで template の元画像から画素値を参照しているために画像の中心位置は拡大倍率によらない点に気付くまで苦労した．加えて，あらかじめ引数をそのまま返す関数を書いておき後ほど拡張したり，可読性を上げるために名前が短く分かりやすい変数を新たに宣言したりといった，ただ結果を出すだけでなく今後に繋がるような学びが多くあった．様々な角度での確かな助言をくれた河野さん並びに西田くんには非常に感謝している．まだまだ直すところがあるため今後も修正を続けたい．

## 付録

```
// g++ -O3 -std=c++11 find.cpp && ./a.out
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <queue>
#include <cmath>
#include <chrono>
using namespace std;

typedef struct{
    int h;
    int w;
}Point;

typedef struct{
    int template_num;
    int diff;
    Point c_dist;
    double scale;
    int rot;
}Output;

class Image{
public:
    const int MAX = 10000;
    int W, H, W_trim, H_trim;
    double S, S_trim;
    Point upleft;
```

```

vector<vector<int>>> data;
vector<vector<bool>>> visited;

// 画像の読み込みファイルが存在しない場合を返す false
bool readdata(string filename){
    string str;
    ifstream fin(filename);
    if(fin){
        fin >> str; fin.ignore();
        if(str != "P2") cout << "file format error" << endl;
        getline(fin, str); // コメント読み捨て
        fin >> W >> H;
        data.resize(H);
        for(int i=0; i<H; i++){
            data[i].resize(W);
        }
        fin >> str; //255
        int h=0, w=0;
        while(fin >> str){
            data[h][w] = stoi(str);
            w = (w+1) % W;
            if(w == 0) h++;
        }
        makevisited();
        return true;
    }
    else return false;
}

// 未踏はfalse, 訪問済みtrue
// が入っている場所は0true, ついでに総面積S画素値の合計も算出(=)
void makevisited(){
    visited.resize(H);
    for(int i=0; i<H; i++){
        visited[i].assign(W,false);
    }
    S = 0;
    for(int h=0; h<H; h++){
        for(int w=0; w<W; w++){
            if(data[h][w] == 0){
                visited[h][w] = true;
            }
            else{
                S += data[h][w];
            }
        }
    }
}

// 内接する四角の左上端と幅, オブジェクトの画素値の合計を計算
void trimming(int h, int w){
    queue<Point> que;
    int dp[3] = {-1,0,1};
    que.push({h,w});
    visited[h][w] = true;
    S_trim = data[h][w];
    int max_h=0, min_h=MAX, max_w=0, min_w=MAX;
    while(que.size()){
        Point p = que.front(); que.pop();
        max_h = max(max_h, p.h);
        min_h = min(min_h, p.h);
        max_w = max(max_w, p.w);
        min_w = min(min_w, p.w);
        for(int i=0; i<3; i++){

```

```

        for(int j=0; j<3; j++){
            Point np = {p.h + dp[i], p.w + dp[j]};
            if(np.h>=0 && np.h<H && np.w>=0 && np.w<W){
                if(! visited[np.h][np.w]){
                    que.push(np);
                    S_trim += data[np.h][np.w];
                    visited[np.h][np.w] = true;
                }
            }
        }
    }
    upleft.h = min_h;
    upleft.w = min_w;
    H_trim = max_h - min_h;
    W_trim = max_w - min_w;
}
};

// 入力画像を倍し度回転させた画像を返すscalerot
Image balance(Image templates, double scale, int rot){
    double theta = rot*M_PI/180;
    Image temp;
    temp.H = templates.H * scale;
    temp.W = templates.W * scale;
    // のオリジナル28*28画像から画素を持ってくるので拡大倍率によらず中心は常にtemplate(14,14)
    Point c = {templates.H/2, templates.W/2};

    temp.data.resize(temp.H);
    for(int i=0; i<temp.H; i++)
        temp.data[i].resize(temp.W);画像中心を中心として回転

    //
    for(int h=0; h<temp.H; h++){
        for(int w=0; w<temp.W; w++){
            double h_s = h/scale;
            double w_s = w/scale;
            int h_r = cos(-theta)*(h_s-c.h) - sin(-theta)*(w_s-c.w) + c.h;
            int w_r = sin(-theta)*(h_s-c.h) + cos(-theta)*(w_s-c.w) + c.w;
            if(h_r >= templates.H) h_r = templates.H-1;
            if(w_r >= templates.W) w_r = templates.W-1;
            if(h_r < 0) h_r = 0;
            if(w_r < 0) w_r = 0;
            temp.data[h][w] = templates.data[h_r][w_r];
        }
    }
    temp.makevisited();
    return temp;
}

int main(){
    // 時間計測開始

    Image target;
    target.readdata("images/images4/image.pgm");

    // 画像がある限り読み込みtemplate
    vector<Image> templates;
    for(int i=0; 1;i++){
        templates.resize(i+1);
        if(! templates[i].readdata("images/images4/template"+to_string(i+1)+".pgm")){
            templates.resize(i);
            break;
        }
    }
}

```

```

    }
}
auto start = std::chrono::system_clock::now();
// の走査target オブジェクトを見つけるとに最適な探すtemplate
for(int h=0; h<target.H; h++){
    for(int w=0; w<target.W; w++){
        if(! target.visited[h][w]){
            target.trimming(h,w);
            Output ans;
            ans.diff = 2147483647;
            Output now;
            for(int i=0; i<templates.size(); i++){
                now.template_num = i;
                // の倍率を画素値の和の比から求め、角度はtemplateまで総当たり-90~90
                now.scale = sqrt((double)target.S_trim / (double)templates[i].S);
                for(int r=0; r<180; r++){
                    now.rot = r-90;
                    Image temp = balance(templates[i], now.scale, now.rot);
                    // は度できればよいtemplate trimming 縮小により離れた点を読むことを防ぐ
                    bool trimmed = false;
                    for(int h_=0; h_<temp.H; h_++){
                        for(int w_=0; w_<temp.W; w_++){
                            if(! temp.visited[h_][w_] and ! trimmed){
                                temp.trimming(h_, w_);
                                now.c_dist = {temp.H/2-temp.upleft.h, temp.W/2-temp.upleft.w};
                                trimmed = true;
                            }
                        }
                    }
                    // 左上の点を合わせて比較範囲は小さい方に合わせる
                    int diff = 0;
                    Point range = {min(target.H_trim,temp.H_trim), min(target.W_trim, temp.W_trim)};
                    for(int dh=0; dh<range.h; dh++){
                        for(int dw=0; dw<range.w; dw++){
                            int temp_pixel = temp.data[temp.upleft.h+dh][temp.upleft.w+dw];
                            int target_pixel = target.data[target.upleft.h+dh][target.upleft.w+dw];
                            // 画素あたりの画素値の差の乗を求める12
                            diff += pow((temp_pixel - target_pixel), 2) / (range.h * range.w) ;
                        }
                    }
                    // が小さい方を答えとするdiff
                    if(diff < ans.diff){
                        now.diff = diff;
                        ans = now;
                    }
                }
            }
            int center_h = target.upleft.h + (ans.c_dist.h);
            int center_w = target.upleft.w + (ans.c_dist.w);
            cout << "template" << ans.template_num+1 << " " << center_w << " " << center_h
                << " " << ans.rot << " " << ans.scale << endl;
            ofstream ofs("result.csv",ios::app);
            ofs << "template" << ans.template_num+1 << "," << center_w << "," << center_h
                << "," << ans.rot << "," << ans.scale << endl;
        }
    }
}

auto end = std::chrono::system_clock::now();
auto dur = end - start;
auto msec = std::chrono::duration_cast<std::chrono::milliseconds>(dur).count();
std::cout << msec << " milli sec \n";
return 0;
}

```