2015年度 卒業論文

骨格タイプ判別システムの開発

大阪産業大学 デザイン工学部 情報システム工学科
情報教育システム研究室

12H035 北川由貴

目次

1　はじめに

ファッション誌や街の服屋にあるマネキンの格好、テレビに移る女優の服装などファッションコーディネートの参考にできるものが増えてきている。近年では、一般人がファッションを提案する iQON [1] や WEAR [2] のような、アプリケーションでも簡単に見れるようになってきている。しかし、ファッション誌に載っている服の組み合わせや街の服屋にあるマネキンの格好を真似してみても、垢抜けない、自分に似合わないと感じる人は少なくないだろう。

この問題を解決するためには、自分に似合うファッションスタイル把握する必要がある。

第 2 章では本研究の目的について述べる。第 3 章では骨格診断 [3] について述べる。第 4 章では本研究で開発したシステムの概要を述べる。第 5 章では骨格タイプ判別システムの開発の結果をその考察とともに述べる。第 6 章には研究の成果とともに今後の課題についてまとめる。

2　本研究の目的

自分の似合うファッションスタイルが分からないという人は、自分の体格をよく分かっていない、または客観的に把握できていないためであると考えられる。

本研究の目的は、骨格診断を用いたユーザーの似合うファッションスタイルを判別するシステムの開発である。骨格診断を用いることで、ユーザーの似合うスタイルを客観的に判別するシステムを開発することを目指す。

3　骨格診断

骨格診断とは、ユーザーが本来持っている体型のラインの特徴を基に骨格タイプ毎に分類し、ユーザーの体型にあった服の素材やデザイン、ファッションアイテムを導き出すことをいう。

骨格診断によって導き出されるタイプは 3 つあり、スタンダードな品格あるスタイルが似合うストレートタイプ、華やかで可愛らしいスタイルが似合うウェーブタイプ、ラフで大人カジュアルなスタイルが似合うナチュラルタイプの 3 つある。自分に適した骨格タイプのファッションをすると、垢抜けた印象をあたえたり着やせして見えるようになる。タイプ別の主な体格 [4] を図 1 , 2 , 3 にそれぞれ示す。

本研究では、この 3 つのタイプの中の 1 つであるストレートタイプにどれほど当てはまるかを測定する。

図 1　後頭部に丸みがあり、首は短め、胸元に厚みがあり、手足が小さいストレートタイプの体格 [4]

図 2　後頭部は平らな印象であり、首は長め、胸元に厚みがなく、全体的に身体が薄いウェーブタイプの体格 [4]

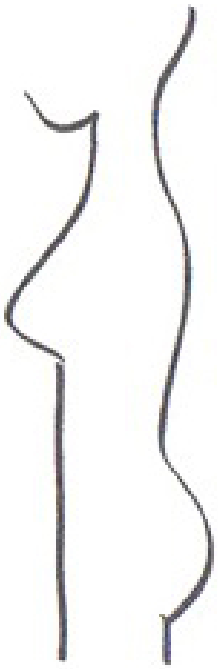

図 3　頭のハチは大きめであり、縦スジが目立つ首であり、手足が大きいナチュラルタイプの体格 [4]

3.1 ストレートタイプ

ストレートタイプは体にメリハリがあり、体に厚みがあり、上重心などの特徴がある。
ストレートタイプの身体の特徴を以下に示す。

- 筋肉がつきやすい
- バストの位置が高い
- 首が短い
- 身体に対して足が小さい
- 身体に対して手が小さい
- 後頭部が出ている

この特徴を基にシステムを開発する。
スタンダードな品格あるスタイルが似合うため以下の特徴のファッションアイテムやファッションスタイル [4] が似合う。ファッションスタイルを図 4 に示す。

- ヘアースタイルは、ショートボブ、フルアップ
- イメージは、シンプル、ベーシック、クール、シャープなど
- 素材はシルクやカシミヤ、表革などハリのある上質な素材
- 柄は大きめの柄や規則的な柄

図 4　左がショートヘアで大きめの柄でクールな印象のスタイル。中央がフルアップでベーシックな印象のスタイル。右がショートヘアでシンプルな印象のスタイル [4]。

4 システム

本研究では、骨格タイプの 1 つであるストレートタイプにどれほど当てはまるかを測定する。そのため、ユーザーを撮影し画像データから骨格タイプの特徴を抽出し、いくつ当てはまるかでストレートタイプの該当率を出力する。

システムは C++ で作成し、OpenCV を利用した。

4.1 OpenCV

OpenCV [5,6] の正式名称は Open Source Computer Vision Library である。

コンピューターで画像や動画を処理するのに必要な、フィルター処理や領域分割、物体認識などの機能が実装されている。マルチプラットフォーム対応しているため、幅広い場面で利用されている。サポート言語は C 言語や C++、Python、Java がある。

今回バージョンは OpenCV2.4.9 を使用した。

4.2 動作

以下にシステムの動作を示す。

1. 測定するユーザーの正面から撮影した画像データとユーザーの左側から撮影したデータを用意する。
2. 測定に必要な体のパーツの大きさや高さを測定する。
3. 身長から測定したパーツがどれほどストレートタイプのデータに当てはまるかをだす。
4. 結果を出力する。

4.3 撮影方法

撮影はカメラの高さ 120cm、カメラから被写体であるユーザーまでの距離は 300cm、背景はブルー一色の背景を用いて行った。被写体であるユーザーは身体のラインが分かるような黒の T シャツ、黒のボトムを着用する。髪は後ろで結び、左側から撮影するときは首にバンドを付けて、首の後ろが分かるようにする。撮影時の服装を図 5 に、撮影場所を図 6 に示す。

図 5　撮影時の服装は身体のラインが分かるような黒の T シャツ、黒のボトムスを着用する。

図 6　カメラの高さ 120cm、ユーザーまでの距離 300cm の撮影スタジオ

4.4 測定方法

以下にストレートタイプの測定に必要な体のパーツを示す。

- バストの高さ
- 首の長さ
- 身長に対しての足の比率
- 身長に対しての手の比率
- 後頭部の突出

測定に利用する正面からの撮影方法を図 7 に、左側面からの撮影画像を図 8 にそれぞれ示す。これらの画像データを BGR を用い、ユーザーの体にあたる部分の BGR 値をすべて 255 に設定する。それ以外の背景部分の BGR 値をすべて 0 に設定する。

2 値化処理した画像データを図 9,10 にそれぞれ示す。

画像の原点を左上にし、y 軸は下方向、x 軸は右方向とする。

サンプル画像の正面からの画像データは y 軸に 700pixel、x 軸に 212pixel であり、左側からの画像データは y 軸に 700pixel、x 軸に 170pixel である。

4.4.1 BGR

BGR [7] とは OpenCV で用いられるカラーモデルである。カラーモデルはチャンネル数、各チャンネルで用いられる色、各チャンネルで用いるピクセル値のデータ型で定義される。BGR は 3 チャンネルの赤、緑、青の RGB カラーであり、1pixel24 ビットフォーマットで成り立っている。この BGR は RGB の赤、緑、青の順を青、赤、緑の順に変えたもので、内容は同一のものである。

4.4.2 HSV

HSV [7] とは色相、彩度、明度の 3 チャンネルから成り立つカラーモデルである。

図 7　入力した正面からの画像データ

図 8　入力した左側からの画像データ

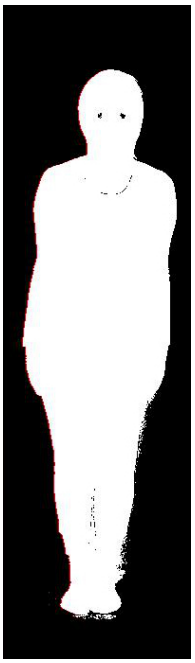

図 9　2 値化した正面からの画像データ

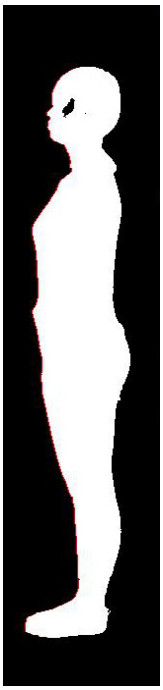

図 10　2 値化した左側からの画像データ

4.4.3 バストの高さ

ユーザーの正面から撮影した画像データから肩の位置がどの高さにあるかを測定する。肩の高さの測定方法は、画像データの左上を原点とし順番に BGR 値を読み取っていく処理を画像データの右下まで行い、列ごとに x 軸上ではじめに発見された BGR 値がすべて 255 の位置から、y 軸方向に 20pixel プラスした位置までで BGR 値がすべて 255 の位置が x 軸方向にマイナスに一番働いた y 軸の位置を肩の高さとする。

その後、ユーザーの左側から撮影した画像データからバストの高さを測定する。肩の高さより y 軸方向にプラスの位置で、列ごとに x 軸上ではじめに発見された BGR 値がすべて 255 の位置から、y 軸方向に 1 プラスした位置で BGR 値がすべて 255 の位置が x 軸方向にプラスに働いた y 軸の位置をバストのトップとする。バストのトップの位置から肩の高さをマイナスした数値をバストの高さの数値とする。

4.4.4 首の長さ

首の長さを測定するためにまず、首と顔の境目を調べる。首と顔の境目は、ユーザーの左側から撮影した画像データから調べる。画像データの左上の原点からバストの高さで測定した肩の高さの位置まで BGR 値を順番に読み取っていく処理を行い、列ごとに x 軸上ではじめに発見された BGR 値がすべて 255 の位置から、y 軸方向に 1pixel プラスした位置で BGR 値がすべて 255 の位置が x 軸方向にプラスに一番働いた y 軸の位置を首と顔の境目の位置とする。

肩の高さから首と頭の境目の位置をマイナスした数値を首の長さの数値とする。

4.4.5 身長に対しての足の比率

身長に対しての足の比率を出すために足のサイズと身長を求める。足のサイズを測定するために、つま先と踵を調べる。つま先は、ユーザーの左側から撮影した画像データから調べる。画像データの左下を原点とし、その位置からバストのトップの位置まで BGR 値を順番に読み取っていく処理を行い、列ごとに x 軸上ではじめに発見された BGR 値がすべて 255 の位置から、y 軸方向に 1pixel プラスした位置で BGR 値がすべて 255 の位置が x 軸方向にプラスに働いた y 軸の位置をつま先の位置とする。

踵は、つま先の y 軸の高さから x 軸方向に BGR 値を読み取っていく処理を行い、BGR の値がすべて 255 になる x 軸上の最後の位置を踵の位置とする。

踵の位置からつま先の位置をマイナスした数値を足のサイズとする。

身長は、画像データの左下を原点とし、列ごとに x 軸上ではじめに発見された BGR 値がすべて 255 の数をカウントしていく。

足のサイズから身長を割り、この数値に 100 をかけた数値を身長に対しての足の比率とする。

4.4.6 身長に対しての手の比率

ユーザーの左側から撮影した画像データの手の部分をトリミングする。トリミング後の画像を図 11 に示す。トリミング後、BGR を HSV に変換する。HSV を使って肌色を検出し [8]、肌色の部分は H と V の値を 255 に,S の値を 0 に設定、それ以外のところは H と S と V の値を 0 に設定し、2 値化処理をする。2 値化処理後の画像を図 12 に示す。

2 値化処理をした手の画像データから、手のサイズを測定する。身長に対しての手の比率を出すために手のサイズと身長を求める。手のサイズを図るため、手のつけ根と手の先を調べる。手のつけ根は、画像データの左上の原点から BGR 値を順番に読み取っていく処理を行い、最初に BGR の値がすべて 255 がくる y 軸の位置を手のつけ根の位置とする。手の先は、、画像データの左上の原点から BGR 値を順番に読み取っていく処理を行い、最後に BGR の値がすべて 255 がくる y 軸の位置を手の先の位置とする。

手の先の位置から手のつけ根の位置をマイナスした数値を手のサイズとする。

身長は、身長に対しての足の比率で求めた身長の数値を利用する。
手のサイズから身長を割り、この数値に 100 をかけた数値を身長に対しての手の比率とする。

図 11　入力した手の画像データ

図 12　2 値化した手の画像データ

4.4.7　後頭部の突出

後頭部の突出度合いを測定するため、首の後ろの位置と後頭部の一番突出している位置を調べる。首の後ろの位置は、首の長さで測定した y 軸の首と頭の境目の位置を BGR 値を x 軸方向に読み取っていく処理を行い、BGR の値がすべて 255 が x 軸上で最後にくるところを首の後ろの位置とする。後頭部の一番突出している位置は、首の後ろの位置から一番 x 軸方向にプラスに働いている位置を後頭部の一番突出している位置とする。

後頭部の一番突出している位置から首の後ろの位置をマイナスした数値を後頭部の突出度合いの数値とする。

4.5　ストレートタイプの適正比率

ユーザーごとの身体のパーツによってストレートタイプの適性比率を計測する。

ストレートタイプの適正比率の計測方法は、項目に当てはまった個数× 20 で比率を出す。項目はストレートタイプの特徴を基に作成した。項目は以下に示す。項目の閾値は以下の測定結果のデータから平均した。計測結果のデータは図 13 に示す。

- バストの高さの数値が 49 以下
- 首の長さの数値が 27.5
- 身長に対しての足の比率が 13.75 以下
- 身長に対しての手の比率が 10.25 以下
- 後頭部の突出度合いの数値が 5.2 以上

サンプル番号	首の長さ	足の比率	手の比率	バストの高さ	後頭部
1	38	14.04	10.75	50	18
2	31	13.33	11.00	41	4
3	23	13.68	11.28	56	3
4	32	12.31	9.28	50	4
5	24	14.15	9.94	45	0
6	23	13.85	9.87	42	3
7	28	14.71	11.03	57	15
8	28	14.34	10.85	49	5
9	25	14.04	10.11	44	0
10	23	13.09	8.40	55	0
平均	27.5	13.75	10.25	48.9	5.2

図 13　測定結果のデータ。足と手の比率は身長を 100 とした比率である。

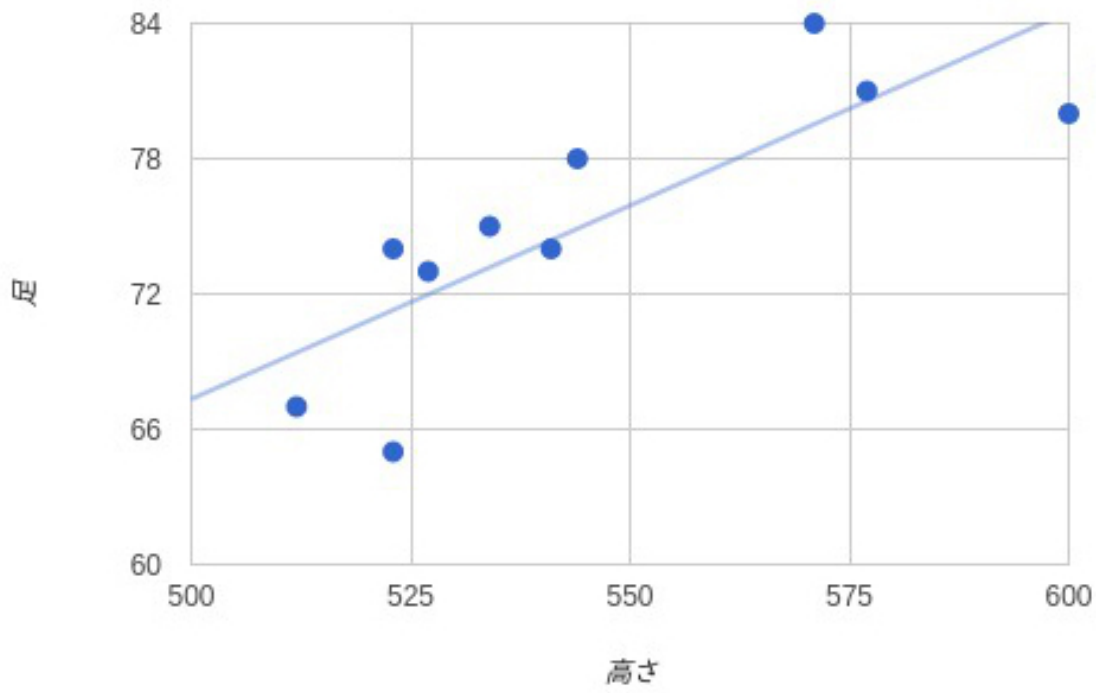

図 14　身長に対しての足の比率のグラフ。直線は最小二乗法を用いて導き出した身長に対して足の比率の平均の値。

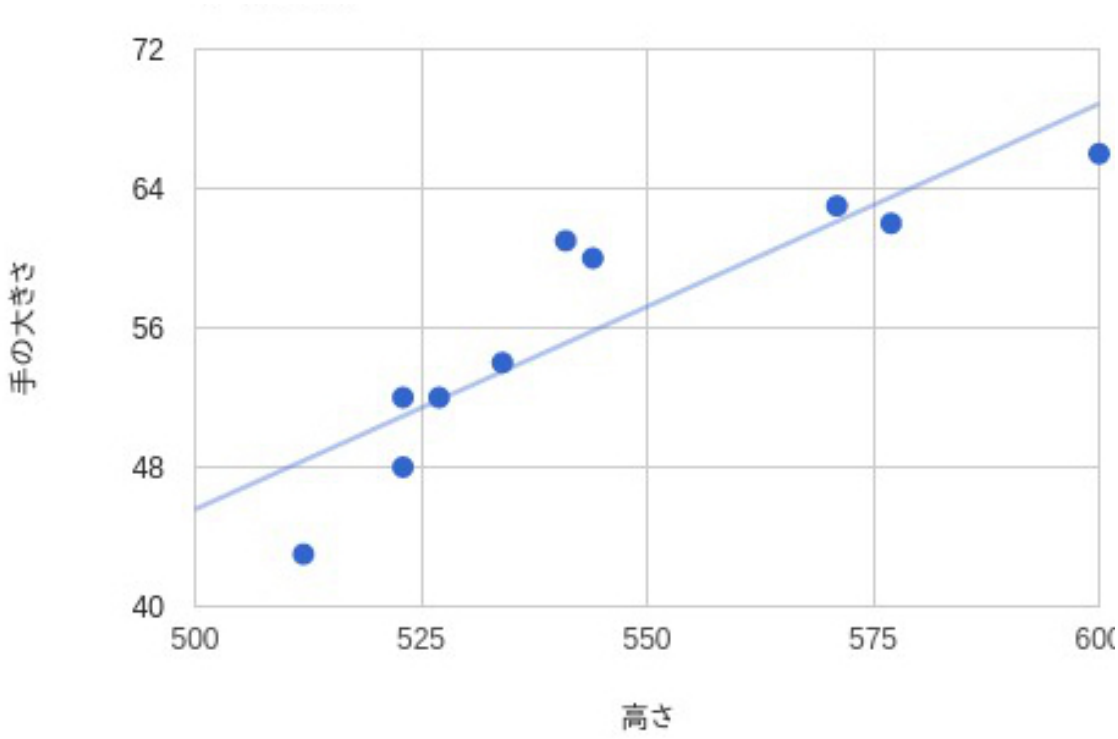

図 15　身長に対しての手の比率のグラフ。直線は最小二乗法を用いて導き出した身長に対して足の比率の平均の値。

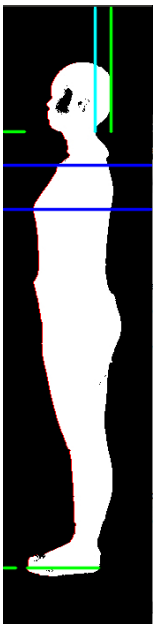

図 16　出力された画像データ。横に伸びる直線は上から、顎の位置を表す直線、肩の位置を表す直線、胸のトップの位置を表す直線、足の位置を表す直線である。縦に伸びる直線は左から、首の後ろの位置を表す直線、後頭部の一番突出している位置を表す直線である。

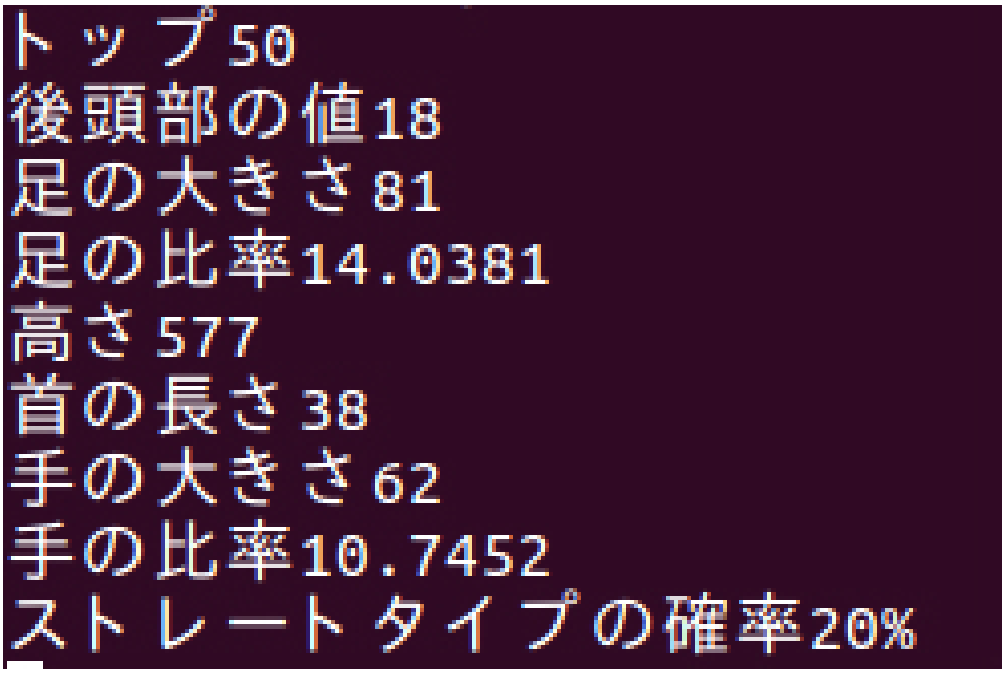

図 17　サンプル番号 1 の出力結果

5　結果と考察

ストレートタイプの身体の特徴が現れやすい、バストの高さ、首の長さ、身長に対しての足の大きさ、身長に対しての手の大きさ、後頭部の突出度合いを測定し数値で表現することができた。この数値を基にストレートタイプにあたる割合を導き出すシステムを構築した。このシステムの利点は、身体の特徴を人間の目で見るよりも正確に測定できることである。

5.1　今後の課題

今後の課題は、実装できなかった要素を実装することと精度を上げることである。現在の骨格タイプ判別システムでより似合うファッションスタイルを導き出すため、以下の課題が考えられる。

5.1.1　精度を上げる

今回、測定結果のデータ量が少なかったためストレートタイプの適正比率の結果に偏りが生じている可能性がある。今後、測定結果のデータを多数収集し、システムの精度を上げることが求められる。

5.1.2　残りの骨格タイプの判別

本稿ではストレートタイプにあたる割合を導き出すことができたが、今後はウェーブタイプやナチュラルタイプにあたる割合を導き出すこともできるよう改良していくことが望ましい。また、割合を出すのではなくユーザーがどのタイプかの結果を出すように改良していくべきである。

5.1.3　パーソナルカラー診断

ユーザーの似合う色を判別するパーソナルカラー診断を追加していく。パーソナルカラー診断とは、ユーザーの肌の色味を判別し似合う色を診断することである。この機能を追加することで、より一層似合うファッションスタイルを導き出せると考えられる。

6　まとめ

本稿では、ユーザーの似合うファッションスタイルを客観的に判別するシステムの開発を行った。そのため、ユーザーが本来持っている体型のラインの特徴を基にファッションアイテムを導き出す骨格診断を基にしたシステムを検討した。

その結果、骨格診断の中の 1 つのタイプであるストレートタイプにあたる割合を導き出すシステムを開発することができた。しかし、残り 2 つのタイプであるウェーブタイプとナチュラルタイプにあたる割合を導き出すことができていないため、今後この 2 つのタイプにあたる割合を導き出す機能の開発を行う必要があると考えられる。また、パーソナルカラー診断を行う機能を追加することと、測定結果のデータを多数収集し現在のシステムの精度を上げることが望ましい。

本システムを用いることで似合うファッションスタイルを提案できるため、今後のアパレル業界などに貢献することが期待できる。

謝辞

本研究を進めていく上で、担当教員である大垣斉准教授に御指導及び御協力を戴きました。また、情報教育システム研究室のメンバー及び卒業生の方々、team.andrew ML に参加されている皆様方に御助言を賜りました。

ここに深く感謝の意を表します。

参考文献

[1] iqon. https://www.iqon.jp/.

[2] Wear. http://wear.jp/.

[3] 山崎真理子. 骨格診断初めての公式本. 株式会社すばる舎リンケージ, 2014.

[4] Shining energy/骨格診断とは. http://p-brand.com/fashion-type-detail/.

[5] Opencv 公式ポータルサイト. http://opencv.org/.

[6] Opencv とは? http://www.buildinsider.net/small/opencv/001.

[7] 永田雅人. 実践 OpenCV2.4 映像処理&解析. 株式会社カットシステム, 2013.

[8] Opencv hsv で肌色検出. http://d.hatena.ne.jp/mintsu123/20111123/1322065624, 2011.

付録 A　付録 1

付録 B　ソースコード

```
#include<stdio.h>
#include<opencv2/opencv.hpp>

#define B(IMG , X , Y) ((uchar*)((IMG) -> imageData + (IMG) -> widthStep*(Y)))[(X)*3]
#define G(IMG , X , Y) ((uchar*)((IMG) -> imageData + (IMG) -> widthStep*(Y)))[(X)*3+1]
#define R(IMG , X , Y) ((uchar*)((IMG) -> imageData + (IMG) -> widthStep*(Y)))[(X)*3+2]

using namespace std;

int h1, h2 , w1 , w2 , x_first, x_second, y_first, y_second , st = 0 ,
  counter = 0 ,
  shoulder_max = 0 , shoulder = 0 ,
  kubi = 0 , kubi_max = 0 , kubi_back = 0 ,
  top = 0 , foot = 0 , foot_count = 0, foot_f = 0,
  occipital = 0 , occipital_max = 0;

double handhi = 0 , foothi = 0 , hand_count = 0 ,count_height = 0;

IplImage *img_niti , *img_niti_mae , *img_hand , *img_hand_niti ,
*img_mae = cvLoadImage("new_yuki_mae.jpg") ,
*img_yoko = cvLoadImage("new_yuki.jpg");

void Mouse(int event, int x, int y, int flags, void *param);

int main(int argc , char *argv[])
{

  cvNamedWindow("入力画像");
  cvNamedWindow("出力画像");
  cvNamedWindow("入力画像前");
  cvNamedWindow("出力画像前");

  if(img_mae == NULL || img_yoko == NULL){
    cout << "画像を取得できません" << endl;
    cvWaitKey(0);
    return -1;
  }
```

```
  cvShowImage("入力画像" , img_yoko);

  cvSetMouseCallback("入力画像" , Mouse);

  w1 = img_yoko -> width;
  h1 = img_yoko -> height;

  w2 = img_mae -> width;
  h2 = img_mae -> height;

  cout << "横の横" << w1 << endl;
  cout << "前の横" << w2 << endl;
  cout << "横の縦" << h1 << endl;
  cout << "前の縦" << h2 << endl;


  img_niti = cvCreateImage(cvSize(w1 , h1) , IPL_DEPTH_8U , 3);
  img_niti_mae = cvCreateImage(cvSize(w2 , h2) , IPL_DEPTH_8U , 3);

  //二値化
  for(int y = 0 ; y < h1 ; y++){ //ピクセル値変換領域垂直方向
    for(int x = 0 ; x < w1 ; x++){ //ピクセル値変換領域水平方
      if(B(img_yoko , x , y) >= 140){
  R(img_niti , x , y) = 0;
  G(img_niti , x , y) = 0;
  B(img_niti , x , y) = 0;
       }
       else{
R(img_niti , x , y) = 255;
G(img_niti , x , y) = 255;
B(img_niti , x , y) = 255;
counter ++ ;
       }
     }
  }

  //img_mae 二値化
  for(int y = 0 ; y < h2 ; y++){ //ピクセル値変換領域垂直方向
    for(int x = 0 ; x < w2 ; x++){ //ピクセル値変換領域水平方
      if(B(img_mae , x , y) >= 180){
  R(img_niti_mae , x , y) = 0;
  G(img_niti_mae , x , y) = 0;
  B(img_niti_mae , x , y) = 0;
```

```
       }
       else{
R(img_niti_mae , x , y) = 255;
G(img_niti_mae , x , y) = 255;
B(img_niti_mae , x , y) = 255;
       }
    }
  }

  //赤線前
  for(int y = 0 ; y < h2 ; y++){ //ピクセル値変換領域垂直方向
    for(int x = 0 ; x < w2 ; x++){ //ピクセル値変換領域水平方
      if(R(img_niti_mae , x , y) == 255 &&
 G(img_niti_mae , x , y) == 255 &&
 B(img_niti_mae , x , y) == 255 ){
        G(img_niti_mae , x , y) = 0;
B(img_niti_mae , x , y) = 0;
break;
       }
     }
  }
  //肩認識
  for(int y = 0 ; y < h2 / 2 ; y++){ //ピクセル値変換領域垂直方向
    for(int x = 0 ; x < w2 ; x++){
      if(B(img_niti_mae , x , y) == 0 &&
 G(img_niti_mae , x , y) == 0 &&
 R(img_niti_mae , x , y) == 255){
for(int a = y; a <= y + 20 ; a++){
  for(int b = 0; b <= x ; b++){
    if(B(img_niti_mae , b , a) == 0 &&
       G(img_niti_mae , b , a) == 0 &&
       R(img_niti_mae , b , a) == 255 ){
      int dfe = x-b ;
      if(shoulder_max < dfe){
shoulder_max = dfe ;
shoulder = a;
      }
    }
  }
}
      }
    }
  }
```

```
   cvLine(img_niti , cvPoint(0 ,shoulder) , cvPoint(200 , shoulder) ,
    CV_RGB(0, 0 , 255) , 2 , CV_AA , 0);
  cout << "肩の値" << shoulder << endl;

  //高さ（横）
  for(int y = 0 ; y < h1 ; y++){ //ピクセル値変換領域垂直方向
    for(int x = 0 ; x < w1 ; x++){ //ピクセル値変換領域水平方
      if(R(img_niti , x , y) == 255 &&
 G(img_niti , x , y) == 255 &&
 B(img_niti , x , y) == 255 ){
        G(img_niti , x , y) = 0;
B(img_niti , x , y) = 0;
count_height ++;
break;
       }
     }
  }

  //top
  for(int y = shoulder + 20 ; y < h1 / 2  ; y++){
    for(int x = 0 ; x < w1 ; x++){
      if(B(img_niti , x , y) == 0 &&
 G(img_niti , x , y) == 0 &&
 R(img_niti , x , y) == 255){
for(int a = y + 1; a <= y + 5 ; a++){
  for(int b = 0; b <= w1 ; b++){
    if(B(img_niti , b , a) == 0 &&
       G(img_niti , b , a) == 0 &&
       R(img_niti , b , a) == 255 && x < b){
top = y;
goto topend ;
    }
  }
}
      }
    }
  }

 topend:

  cout << "トップ" << top - shoulder << endl;
  cvLine(img_niti , cvPoint(0 ,top) , cvPoint(200 , top) ,
```

```
    CV_RGB(0, 0 , 255) , 2 , CV_AA , 0);

  if(top - shoulder < 27.5){
    st++;
  }

  //首のはじまり
  for(int y = 0 ; y < shoulder-20 ; y++){
    for(int x = 0 ; x < w1 ; x++){
      if(B(img_niti , x , y) == 0 &&
 G(img_niti , x , y) == 0 &&
 R(img_niti , x , y) == 255){
for(int a = y + 1; a <= y + 2 ; a++){
  for(int b = x + 1; b <= w1 ; b++){
    if(B(img_niti , b , a) == 0 &&
       G(img_niti , b , a) == 0 &&
       R(img_niti , b , a) == 255 ){
      int dfe = b - x ;
      if(kubi_max < dfe){
kubi_max = dfe ;
kubi = a;
      }
    }
  }
}
      }
    }
  }

  cout << "首の値" << shoulder - kubi << endl;
  cvLine(img_niti , cvPoint(0 ,kubi) , cvPoint(25 , kubi) ,
 CV_RGB(0, 255 , 0) , 2 , CV_AA , 0);

  //首の後ろ
  for(int x = 0; x < w1; x++){
    if(B(img_niti , x , kubi) == 255 &&
       G(img_niti , x , kubi) == 255 &&
       R(img_niti , x , kubi) == 255 &&
       B(img_niti , x + 1 , kubi) == 0 &&
       G(img_niti , x + 1 , kubi) == 0 &&
       R(img_niti , x + 1 , kubi) == 0 ){
      kubi_back = x;
      break;
```

```
    }
  }

  cout << "首の後ろ" << kubi_back  <<endl;
  cvLine(img_niti , cvPoint(kubi_back ,0) , cvPoint(kubi_back , kubi) ,
 CV_RGB(0, 255 , 255) , 2 , CV_AA , 0);

  //後頭部
  for(int y = 0 ; y < kubi - 50 ; y++){ //ピクセル値変換領域垂直方向
    for(int x = 0 ; x < w1 ; x++){
      if(B(img_niti , x , y) == 255 &&
 G(img_niti , x , y) == 255 &&
 R(img_niti , x , y) == 255 &&
 B(img_niti , x + 1 , y) == 0 &&
 G(img_niti , x + 1 , y) == 0 &&
 R(img_niti , x + 1 , y) == 0 &&
   occipital <= x){
   occipital = x;
      }
    }
  }

  cout << "後頭部の値" << occipital - kubi_back << endl;
  cvLine(img_niti , cvPoint(occipital ,0) , cvPoint(occipital , kubi) ,
    CV_RGB(0, 255 , 0) , 2 , CV_AA , 0);

  if(occipital - kubi_back > 5){
    st++;
  }

  //足
  for(int y = h1; y > top; y--){
    for(int x = 0; x <= w1; x++){
      if(B(img_niti , x , y) == 0 &&
 G(img_niti , x , y) == 0 &&
 R(img_niti , x , y) == 255){
for(int x2 = x + 1; x <= w1; x++){
   if(B(img_niti , x2 , y - 1) == 0 &&
      G(img_niti , x2 , y - 1) == 0 &&
      R(img_niti , x2 , y - 1) == 255){
     foot = y;
     foot_f = x;
     goto footend;
```

```
   }
}
      }
    }
  }

 footend:

  for(int x = foot_f; x < w1; x++){
    if(B(img_niti , x , foot) == 255 &&
       G(img_niti , x , foot) == 255 &&
       R(img_niti , x , foot) == 255 &&
       B(img_niti , x + 1 , foot) == 0 &&
       G(img_niti , x + 1 , foot) == 0 &&
       R(img_niti , x + 1 , foot) == 0){
      foot_count =  x;
    }
  }

  /*for(int x = 0;x < w1;x++){
    if(B(img_niti , x , foot) == 0 &&
       G(img_niti , x , foot) == 0 &&
       R(img_niti , x , foot) == 0){
      foot_count++;
    }
  }
  foot_count += 1;*/

  foothi = (foot_count - foot_f) / count_height * 100 ;

  cout << "足の値" << foot << endl;
  cvLine(img_niti , cvPoint(0 ,foot) , cvPoint(15 , foot) ,
    CV_RGB(0, 255 , 0) , 2 , CV_AA , 0);
  cout << "足の大きさ" << foot_count - foot_f << endl;
  cout << "足の比率" << foothi  << endl;
  cvLine(img_niti , cvPoint(foot_f ,foot) , cvPoint(foot_count , foot) ,
   CV_RGB(0, 255 , 0) , 2 , CV_AA , 0);

  if(foothi <= 13.75415){
    st++;
  }

  cvShowImage("入力画像" , img_yoko);
```

```
  cvShowImage("出力画像" , img_niti);
  cvShowImage("入力画像前" , img_mae);
  cvShowImage("出力画像前" , img_niti_mae);

  cout << counter << endl;
  cout << "高さ"  << count_height << endl;
  cout << "首の長さ" << shoulder - kubi << endl ;


  if(shoulder - kubi < 27){
    st++;
  }

  cvWaitKey(0) ;

  cvDestroyAllWindows();
  cvReleaseImage(&img_mae);
  cvReleaseImage(&img_niti);
  cvReleaseImage(&img_yoko);
  cvReleaseImage(&img_niti_mae);
  return 0 ;
}

void Mouse(int event , int x , int y , int flags , void *param = NULL)
{
  //int x1, x2, y1, y2;

  //img =cvLoadImage("new_chieri_yoko.jpg");
  switch(event){
  case CV_EVENT_LBUTTONDOWN :
    x_first = x;
    y_first = y;
    //cout << x_first << "," << y_first << "ここに 1 つ目"  << endl;
    break;

  case CV_EVENT_RBUTTONDOWN :
    x_second = x;
    y_second = y;
    // cout << x_second << "," << y_second << "ここに 2 つ目"  << endl;

    cvSetImageROI(img_yoko ,
   cvRect(x_first , y_first , x_second - x_first , y_second - y_first));
    cvSaveImage("hand.png" , img_yoko);
```

```
    img_hand = cvLoadImage("hand.png");
    cvNamedWindow("image_hand");

    w1 = img_hand -> width ;
    h1 = img_hand -> height;

    img_hand_niti = cvCreateImage(cvSize(w1 , h1) , IPL_DEPTH_8U , 3);
    cvCvtColor(img_hand , img_hand_niti , CV_BGR2HSV);
    for(int y = 0 ; y < h1 ; y++){
      for(int x = 0 ; x < w1 ; x++){
if(B(img_hand_niti , x , y) >= 0 &&
   B(img_hand_niti , x , y) <= 30 &&
   G(img_hand_niti , x , y) >= 50 &&
   R(img_hand_niti , x , y) >= 50){
  B(img_hand_niti , x , y) = 255;
  G(img_hand_niti , x , y) = 0;
  R(img_hand_niti , x , y) = 255;
}else{
  B(img_hand_niti , x , y) = 0;
  G(img_hand_niti , x , y) = 0;
  R(img_hand_niti , x , y) = 0;
}
      }
    }

    cvCvtColor(img_hand_niti , img_hand_niti , CV_HSV2RGB);
    //赤線
    for(int y = 0 ; y < h1 ; y++){ //ピクセル値変換領域垂直方向
      for(int x = 0 ; x < w1 ; x++){ //ピクセル値変換領域水平方
if(R(img_hand_niti , x , y) == 255 &&
   G(img_hand_niti , x , y) == 255 &&
   B(img_hand_niti , x , y) == 255 ){
  G(img_hand_niti , x , y) = 0;
  B(img_hand_niti , x , y) = 0;
  hand_count ++;
  break;
}
      }
    }

    handhi = hand_count / count_height * 100 ;

    cout << "手の大きさ" << hand_count << endl;
```

```
    cout << "手の比率" << handhi  << endl;

    if(handhi <= 10.250041){
      st++;
    }
    cvShowImage("image_hand" , img_hand_niti);


    cout << "ストレートタイプの確率" << st * 20 << "%" << endl;

    cvReleaseImage(&img_hand_niti);
    cvReleaseImage(&img_hand);
    break;

  default:
    break;
  }

}
```