１.課題の目的・内容

□目的

「色相 H」を用いることで、入力画像から顔を検出・抽出し、抽出した顔の「目」の部分にモザイク加工を施す。

□内容

我々が設定した課題の内容は、以下の 3 つに大別できる。

(1) 色相や色差、輝度を計算し、画像内から「顔」を抽出する。
(2) (1)で検出した顔の範囲内で、「目」の位置を特定する。
(3) (2)で特定した目の部位を、長方形型のモザイク加工を施す。

２.概要

□課題解決のためのアイデア

(1) 顔の検出・抽出

・各画素の色相 H・輝度 Y、色差 Cb、Cr を計算し、それらの値が肌色の条件を満たすかを判定し、検出や抽出を行う。

・顔と顔でない肌色領域の区別は、検出した肌色領域の図形的特徴を計算し、顔であるかを判定する。判定の際に用いる顔の図形的特徴は、「画像全体に対する比率、丸型であるか、縦横比、領域の色の範囲、左右対称であるか、抽出領域の上部分に目の空白（孤立）領域があるか」が挙げられる。

(2) 目の検出

・顔の抽出範囲内で、さらに各画素の色相 H・輝度 Y、色差 Cb、Cr を計算することで、「人間の目」の候補領域を抽出。

・顔の上部分に限定して、目や瞳の色相を計算し、検出する。

・検出した目・瞳の図形的特徴を計算し、目であるかどうかを判定する。判定の際に用いる目・瞳の図形的特徴は、「丸型であるか、領域の色の範囲」が挙げられる。

・顔の範囲内でさらに抽出することが難しい場合、単純に検出した顔のどの位置に目が存在するか（一般的に人間の目が、顔のどの位置にあるか）を計算することで、目を特定することができると考えられる。

(3) モザイク加工

・抽出した目の部分を、長方形の形で塗りつぶす。

・塗りつぶす際の色は、黒か、モザイク柄にする。

・モザイク加工を施すには、塗りつぶす領域の各画素の色を、乱数を用いてランダムに指定することで、「モザイク風」になると考えられる。

□着眼点

・「肌色」という色は、画像の明度や彩度によってはかなり異なる RGB 値となる。
R、G、B の各範囲を指定して、「RGB 空間」を直方体で切り出すだけでは、適切に「肌色の画素だけ」を画像から取り出すことはできない。
これに対し、色相 H、輝度 Y、色差 Cb、Cr の範囲を指定して、「HSV 空間」及び「YCbCr 空間」で考えることで、どのような入力画像に対しても、画像内の色の明るさによらず、比較的安定して肌色を指定して抽出することができるという点。

・「顔認識」を行うためには、画像から肌色領域を検出し、その検出した領域から「顔」を選別しなければならない。
今回は、検出した顔候補肌色領域から、

1) 画像全体に対する孤立領域の面積比
2) 外接四角形中の特定の色の画素の面積比
3) 外接四角形の縦横比
4) 色の範囲
5) 左右対称性

の計 5 つの図形特徴を計算し、顔であるかを判定することにより、より顔認識の精度を高めることに成功した。

□アピール点
・ソースコードの総行数が 677 行。
・本プログラムは二人のみで完成させた。
・班員各々が用意した計 13 個の関数をうまく繋げることにより、顔の抽出及びモザイク加工処理を成功できた。
・出力画像は、顔の候補領域の検出画像、顔と判定した領域の抽出画像、原画像に顔と認識した部分を緑の枠線で囲った画像、原画像をモザイク加工した画像の計 4 枚の画像を得られる。
・プログラム中では、色相計算や領域のラベリングといった技術を利用している。
・コマンドライン上でも入力画像を指定することができる上、どのような画素数の画像でも顔の抽出とモザイク加工を施すことができ、汎用性の高いプログラムを完成できた。
・画像の背景色等の条件が良ければ、画像内の顔の抽出可能人数は無制限である。
・顔検出をする際に、首の部分を「顔」と認識しないように、顔領域を認識する際は縦横比を 1.3 にすることで顔のみを外接四角形で囲むことに成功している。
・手や腕などの顔と同じ大きさの肌色領域を誤認識しないように、顔の判定条件を工夫した。

３.アルゴリズムの説明

顔の検出・抽出及びモザイク加工は、次に示す 6 つの処理の順に行う。

①処理 1：肌色画素の検出
②処理 2：肌色領域のラベリング
③処理 3：顔候補領域の図形特徴の算出および選別
④処理 4：選別した顔領域のみを抽出
⑤処理 5：顔の周りを緑の外接四角形で囲む
⑥処理 6：顔のモザイク加工

それぞれの処理について、以下に説明する。

① 処理 1：肌色画素の検出（ 関数名：skin_color_detection ）

本プログラムは RGB の値ではなく、HSV と YCbCr を用いて肌色かどうかを判別する。
入力画像の各画素の RGB データから、色相 H・輝度 Y、色差 Cb、Cr を計算し、それらの値が設定した肌色の条件を満たすかを判定し、検出を行う。

色相 H は次式で計算する。[1)]

$$H = \begin{cases} 60 \times (G - B)/(Max - Min) & (Max = R\text{のとき}) \\ 60 \times (B - R)/(Max - Min) + 120 & (Max = G\text{のとき}) \\ 60 \times (B - G)/(Max - Min) + 240 & (Max = G\text{のとき}) \end{cases} \quad \text{—①}$$

ここで、RGB 値は 0.0～1.0 の範囲（通常は 0～255 の値であるが、255 で割っている）で定義され、Max は RGB 値の最大値、Min は RGB 値の最小値を表す。
また H は 0~360 の範囲で定義され、0 未満の時は 360 を足した数値、360 以上の時は 360 を引いた値となり、Max=Min の時は、H は定義されない。

次に、Y、Cb、Cr を次式で計算する。

$$\begin{cases} Y = 0.299 \times R + 0.587 \times G + 0.114 \times B \\ Cb = -0.172 \times R - 0.339 \times G + 0.511 \times B \\ Cr = 0.511 \times R - 0.428 \times G - 0.083 \times B \end{cases} \quad —②$$

ここで RBG は 0 ~ 255 の範囲で定義される。Y は色の輝度、Cb は輝度と R 成分の色差、Cr は輝度と B 成分の色差を表す[2)]。なお、Y の式の右辺で RGB に掛かっている係数は、人間の視覚度特性から決定されている。

以上より、上記の式で色相 H、輝度 Y、色差 Cb、Cr の値を求め、その値が以下の条件を満たす時に肌色と判定しそのままの RGB 値を保持して出力し、それ以外の画素は白色（画素値 255）として出力することで、肌色領域を検出している。

【H<50 または H>210 】かつ【-50<Cb<-20】かつ【20<Cb<40】かつ【Y>100】

ここで、図 2 を用いて、処理 1 を説明すると、色相・色差・輝度の計算によって、「肌色」であると判定された「手」や、光加減によって肌色に映った「髪」、「首元を含む顔」が検出されており、それ以外の「青い服」や「背景」はすべて白色で出力されている。

よって、処理 1 により、図 2, 7, 12, 17 に示す、肌色画素領域の検出結果を得ることができる。

②処理 2：肌色領域のラベリング（ 関数名：labeling ）

処理 1 によって肌色部分をそれぞれ孤立領域として抽出した後、ラベリングを行うことにより、肌色部分の孤立図形を構成する画素に、孤立図形領域毎の通し番号を振っている。

具体的に説明すると、まず、入力画像と同じ縦と横の大きさを持つ配列 label を定義して、そのすべてを 0 に初期化する。
次に、入力画像の全画素について、左上から右下へ走査する。
その際、はじめて白色ではない画素（ラベル番号が 0 の画素）すると、その画素に 0 に 1 を加えたものをラベル値とする。またそれと同時に、注目画素についての上下左右の 4 近傍のラベル番号を調べ、その中で最大値のラベル番号をその注目画素のラベル値とする。

その処理が画像の右下まで終了したら、もう一度左上から走査を行う。そのとき、もしラベル値が 0 でないなら上下左右のラベル値を調べ、その 4 つの要素の中の最大値を画像中にあるすべての同じラベル値を置換する（各孤立領域のラベリング完了）。

以上の処理を、画像の右下まで繰り返した後、最後にもう一度左上から画像を走査することで、最終的な肌色領域の個数（total_area）をラベル値の値をもとにして求め、最終的な肌色領域の個数を返している。

③処理 3 : 顔候補領域の図形特徴の算出および選別（ 関数名 : cal_area_label ）

処理 2 で求めたラベル番号を持つ各孤立領域の図形的特徴を求める。
具体的に説明すると、各ラベルの番号の孤立領域の図形特徴を算出し、顔かどうかを判定する。孤立領域の図形的特徴として、次に挙げる図形特徴を算出し、顔であるかを判定する。

a) 画像全体に対する孤立領域の面積率
b) 外接四角形中の肌色の画素の面積率
c) 外接四角形の縦横比
d) 色の範囲
e) 左右対称性

以下に、a)~e)までの判定方法を説明する。

a) 画像全体に対する孤立領域の面積率
孤立領域に対して、同じラベル番号を持つ画素の個数（sum_area）を数えることで領域の面積を求めている。そして、その面積と入力画像全体の面積(row*cols)の比が、ある一定の値（AREARATIO = 0.002）以上であるならば、顔であると判別する。

b) 外接四角形中の肌色の画素の面積率
孤立領域の左上と右下の座標を求めることで、外接四角形の横幅 W と縦幅 H を決めることができ、孤立領域に外接するように囲んだ四角形の面積を求めることができる。肌色画素の面積（sum_area）を、この外接四角形の面積（W*H）で割った面積率がある一定の値（AREARATE = 0.3）以上であるならば、顔であると判定する。

c) 外接四角形の縦横比
外接四角形が横長でなく、正方形か少し縦長である場合、すなわち、外接四角形の横幅 W を縦幅 W で割った縦横比が、ある一定の範囲内（HWMIN 以上 HWMAX 以下）であるならば顔であると判定している。

d) 色の範囲
顔の候補領域の外接四角形内には目鼻口や髪の毛の一部などが含まれると考えられるので、原画像中の外接四角形内の画素の色の範囲を調べている。具体的には、孤立領域中の各 RGB 値の最大値と最小値の差を求め、各 RGB 値の差の合計値(colrange)がある一定の値 (= 300) 以上であれば顔であると判定している。その差が小さければ（色の範囲が狭すぎれば）壁などの可能性があると考えられる。

e) 左右対称性
顔はほぼ左右対称なので、原画像の外接四角形内の画像を中心線で折り返して重なる画素の色差から左右対称度を求めている。具体的には、外接四角形の横幅の中点を境界として、左端の座標に対称の位置にある座標の各 RGB 値の差を求め、その処理を外接四角形全ての画素について行う。そして、その差の合計値がある一定の値（SYMMAX＝0.7）以下であれば、顔であると判定している。

以上の a) ~ e)の全ての条件を満たす領域のみが顔であると判定し、領域を選別している。

④処理 4：選別した顔領域のみを抽出（ 関数名：face ）

処理 3 で選別したラベル番号の領域のみを別ファイルに抽出（出力）している。
それ以外の画素は白色（画素値 255）として出力することで、「顔領域のみ」の肌色領域を抽出している。

ここで、図 3 を用いて、処理 2～4 を説明する。処理 1 で得られた肌色孤立領域には大きく「髪」、「手」、「首元を含む顔」の三つであると分かる。（厳密にいえば、より多くの小さい孤立領域が存在する。）この各孤立領域にラベリング処理を施し、各番号の候補領域の図形特徴を計算することで、「髪」及び「手」の領域が「顔でない」と判定され、図 3 に示す「顔」のみの画像となっている。

以上の処理 2 ~ 4 により、図 3, 8, 13, 18 に示す、顔領域のみの抽出結果を得ることができる。

⑤処理 5：顔の周りを緑の外接四角形で囲む（ 関数名：draw_rect ）

外接四角形を描くために、関数 edge を利用して、外接四角形の左上の座標と右下の座標を得る。
まず左上の座標を求める。入力画像の左から右を、上から下にかけて走査していき、最初に検知した白以外の画素の y 座標（すなわち、孤立領域の左端かつ一番上側の座標）を、左上の座標の y 座標とする。
次に、入力画像の上から下を、左から右にかけて走査していき、最初に検知した白以外の画素の x 座標（すなわち、孤立領域の上端かつ一番左側の座標）を、左上の x 座標とする。

同様に、右下の座標を求めるために、入力画像の右から左を、下から上にかけて走査していき、最初に検知した白以外の画素の y 座標（すなわち、孤立領域の右端かつ一番下側の座標）を、右下の座標の y 座標とする。
次に、入力画像の下から上を、右から左にかけて走査していき、最初に検知した白以外の画素の x 座標（すなわち、孤立領域の下端かつ一番右側の座標）を、右下の座標の x 座標とする。

その後、得られた外接四角形の座標の縦横比（縦 / 横）が 1.3 以内かをチェックし、もし 1.3 以上であれば、縦横比が 1.3 になるように補正する。具体的には、左上の y 座標と横幅の 1.3 倍の値を加算した値を、右下の y 座標に代入している。このように工夫することで、顔とつながっている「首」などの肌色領域を外接四角形から排除して「顔のみ」を囲むことができる。

ここで、図 4 を用いて説明すると、顔領域として抽出した領域に首の部分が含まれており、その領域の外接四角形を調べると、縦横比が 1.3 以上であるので、補正した座標で外接四角形を描いている。よって、外接四角形内には「顔」のみが含まれており、抽出されていた「首の部分」は外接四角形から排除されている。

以上より、関数 edge から得られた左上の座標（x1, y1）と右下の座標（x2, y2）を用いることで、外接四角形を構成する座標（x1, y1），（x2, y1），（x2, y2），（x1, y2）に沿って幅が 4(=bold)の緑色の外接四角形を描く。

よって、処理 5 により、図 4, 9, 14, 19 に示す、顔領域を緑の外接四角形で囲んだ出力結果を得ることができる。

⑥処理 6：顔のモザイク加工（ 関数名：draw_rect_mosaic ）

関数 edge から得られた左上の座標（x1, y1）と右下の座標（x2, y2）を用いることで、外接四角形を構成する座標（x1, y1），（x2, y1），（x2, y2），（x1, y2）を得ることができる。
この外接四角形の上からどの位置に目があるかを外接四角形の大きさから推定し、モザイク処理を施す。なお、抽出した顔の大きさによって、場合分けをすることによりどのような顔の大きさにも対応できるようにしている。
具体的には、外接四角形の高さ（= y2 – y1）が 75 よりも大きい場合（画像に対する顔の大きさが標準的な大きさの場合)、外接四角形の上から高さの 1/4 の位置から、幅 25 の範囲に目があると仮定して、その部分をモザイク加工している。
外接四角形の高さ（= y2 - y1）が 75 よりも小さい場合（画像に対する顔の大きさが小さい場合)、外接四角形の上から、高さの 1/5 の位置から幅 15 の範囲に目があると仮定して、その部分をモザイク加工している。

モザイク加工処理は、塗りつぶす領域の各画素の RGB 値をそれぞれ平均し、-48~48 の乱数を用いて、ランダムに色を指定することで、「モザイク風」の色合いになるように工夫してある。

なお、今回の課題では、抽出した顔の範囲内で、さらに目を抽出し、抽出した目をモザイク加工する計画であったが、「抽出した顔の肌色領域画像には『目』が含まれておらず、目の色相を単純に検出することが不可能」であり、技術的に困難であったため、顔の外接四角形を用いて、人間の顔の縦幅の 1/4 または 1/5 の位置に「目」があると仮定してその位置にモザイク加工を施すことにより、本課題を解決した。

ここで、図 5 を用いて説明すると、外接四角形の高さが「75」よりも大きいため、外接四角形の上から高さの 1/4 の位置から、幅 25 の範囲をモザイク加工しており、目の部分を完全にモザイク加工処理することに成功している。

処理 6 により、図 5, 10, 15, 20 に示す、外接四角形から目の位置を特定し、モザイク加工を施した出力結果を得ることができる。

４.結果

入力画像から顔を検出・抽出し、抽出した顔の「目」の部分にモザイク加工を施すプログラムソースをソースコード１に示す。

□検証１（1 人の写真）

・検証１における１人の女性が映った入力用のカラー画像[3)]を図１に示す。
図１の画像を入力画像とし、図１の肌色画素を検出した画像を図２に示す。
図 1 の画像を入力画像とし、図 2 の肌色画素領域から選別した顔領域のみを抽出した画像を図 3 に示す。
図１の画像を入力画像とし､図３の顔領域を緑の外接四角形で囲んだ画像を図４に示す。
図 1 の画像を入力画像とし、図 4 の外接四角形から目の位置を特定し、モザイク加工を施した画像を図 5 に示す。

ソースコード 1 を用いて、図 1 のカラー画像を入力画像とし、色相計算を用いてプログラムを実行すると、図 2 ~ 5 の出力画像を得られた。

なお、この時の色相 H、色差 Cb、Cr、輝度 Y の肌色の条件及び、顔の画像全体の面積に対する下限の条件、モザイクの縦幅の太さは以下のように設定した。

○条件

- 色差 H：　H < 50 または 210 < H
- 色相 Cb：　-70 < Cb < -15
- 色相 Cr：　10 < Cr < 30
- 輝度 Y：　Y > 100

- 顔の画像全体の面積に対する下限：0.5%
- モザイクの縦幅の太さ：
  外接四角形の高さが 75 よりも大きい場合：75
  外接四角形の高さが 75 よりも小さい場合：65

□検証 2（5 人の写真）

・検証 2 における 5 人の人物が映った入力用のカラー画像[4]を図 6 に示す。
図 6 の画像を入力画像とし、図 6 の肌色画素を検出した画像を図 7 に示す。
図 6 の画像を入力画像とし、図 7 の肌色画素領域から選別した顔領域のみを抽出した画像を図 8 に示す。
図 6 の画像を入力画像とし､図 8 の顔領域を緑の外接四角形で囲んだ画像を図 9 に示す。
図 6 の画像を入力画像とし、図 9 の外接四角形から目の位置を特定し、モザイク加工を施した画像を図 10 に示す。

ソースコード 1 を用いて、図 6 のカラー画像を入力画像とし、色相計算を用いてプログラムを実行すると、図 7 ~ 10 の出力画像を得られた。

なお、この時の色相 H、色差 Cb、Cr、輝度 Y の肌色の条件及び、顔の画像全体の面積に対する下限の条件、モザイクの縦幅の太さは以下のように設定した。

○条件

・色差 H：　$H < 120$ または $190 < H$
・色相 Cb：　$-70 < Cb < -16$
・色相 Cr：　$10 < Cr < 55$
・輝度 Y：　$Y > 100$

・顔の画像全体の面積に対する下限：0.2%
・モザイクの縦幅の太さ：
　外接四角形の高さが 75 よりも大きい場合：25
　外接四角形の高さが 75 よりも小さい場合：15

□検証 3（10 人の写真）

・検証 3 における 10 人の子供が映った入力用のカラー画像[5]を図 11 に示す。
図 11 の画像を入力画像とし、図 11 の肌色画素を検出した画像を図 12 に示す。
図 11 の画像を入力画像とし、図 12 の肌色画素領域から選別した顔領域のみを抽出した画像を図 13 に示す。
図 11 の画像を入力画像とし、図 13 の顔領域を緑の外接四角形で囲んだ画像を図 14 に示す。
図 11 の画像を入力画像とし、図 14 の外接四角形から目の位置を特定し、モザイク加工を施した画像を図 15 に示す。

ソースコード 1 を用いて、図 11 のカラー画像を入力画像とし、色相計算を用いてプログラムを実行すると、図 12 ~ 15 の出力画像を得られた。

なお、この時の色相 H、色差 Cb、Cr、輝度 Y の肌色の条件及び、顔の画像全体の面積に対する下限の条件、モザイクの縦幅の太さは以下のように設定した。

○条件

- 色差 H：　$H < 60$ または $200 < H$
- 色相 Cb：　$-70 < Cb < -16$
- 色相 Cr：　$10 < Cr < 30$
- 輝度 Y：　$Y > 100$

- 顔の画像全体の面積に対する下限：0.1%
- モザイクの縦幅の太さ：
  外接四角形の高さが 75 よりも大きい場合：15
  外接四角形の高さが 75 よりも小さい場合：6

□検証 4（33 人の写真）

・検証 4 における 33 人の人物が映った入力用のカラー画像[6]を図 16 に示す。
図 16 の画像を入力画像とし、図 16 の肌色画素を検出した画像を図 17 に示す。
図 16 の画像を入力画像とし、図 17 の肌色画素領域から選別した顔領域のみを抽出した画像を図 18 に示す。
図 16 の画像を入力画像とし、図 18 の顔領域を緑の外接四角形で囲んだ画像を図 19 に示す。
図 16 の画像を入力画像とし、図 19 の外接四角形から目の位置を特定し、モザイク加工を施した画像を図 20 に示す。

ソースコード 1 を用いて、図 16 のカラー画像を入力画像とし、色相計算を用いてプログラムを実行すると、図 17 ~ 20 の出力画像を得られた。

なお、この時の色相 H、色差 Cb、Cr、輝度 Y の肌色の条件及び、顔の画像全体の面積に対する下限の条件、モザイクの縦幅の太さは以下のように設定した。

○条件

- 色差 H：　H < 48 または 202 < H
- 色相 Cb：　-60 < Cb < -20
- 色相 Cr：　20 < Cr < 40
- 輝度 Y：　画像全体が暗く、肌色領域が濃い色となっているため、設定していない。

- 顔の画像全体の面積に対する下限：0.05%
- モザイクの縦幅の太さ：
  外接四角形の高さが 75 よりも大きい場合：25
  外接四角形の高さが 75 よりも小さい場合：10

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <ctype.h>
#include "pnmimg.h"

/* 各パラメータの割合の定義 [ ()内はデフォルトの値 ] */
#define AREARATIO 0.002 /* 顔の画像全体の面積に対する下限(0.5%) */
#define AREARATE  0.3   /* 外接四角形における占める割合の下限(30%) */
#define HWMIN     0.8   /* 縦横比の下限(0.8) */
#define HWMAX     1.8   /* 縦横比の上限 (1.8)*/
#define SYMMAX    0.7   /* 左右対称度の上限(0.3) */
#define NOP         10            /*検出最大人数(10)*/
#define FACE_RATE 1.3             /*顔の縦横比(1.3)*/


/*3つの引数の内、最小値を返す関数*/
double return_min(double a,double b,double c){
      double min=a;
      if(min>b){
            min=b;
      }
      if(min>c){
            min=c;
      }
      return min;
}
/*ここまで*/


/*3つの引数の内、最大値を返す関数*/
double return_max(double a,double b,double c){
      double max=a;
      if(max<b){
            max=b;
      }
      if(max<c){
            max=c;
      }
      return max;
}
/*ここまで*/


/**********処理１．肌色画素の検出**********/
/*＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊肌色画素の検出＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
void skin_color_detection(RGB_PACKED_IMAGE *input,RGB_PACKED_IMAGE *output){
      int j,k;                  //for文カウンター変数
      double R=0,G=0,B=0;             //各画素のRGB値を格納する変数
      int H;                        //色相H
      double Y=0.0,Cb=0.0,Cr=0.0;               //Y…輝度,Cb,Cr…色差
            for(k=0;k<input->rows;k++){
                  for(j=0;j<input->cols;j++){           //画像中の1つの画素のRBG値を読み取る
                        R=(double)input->p[k][j].r;
                        G=(double)input->p[k][j].g;
                        B=(double)input->p[k][j].b;

                        Y = (0.299*R)+(0.587*G)+(0.114*B);     //Y,Cb,Crの計算
                        Cb= (-0.172*R)-(0.399*G)+(0.511*B);
                        Cr= (0.511*R)-(0.428*G)-(0.083*B);

                        R=R/(double)255;            //RGB値を[0.0~1.0]の間で再定義する
                        G=G/(double)255;
                        B=B/(double)255;

                        //色相Hの計算
                        if(return_max(R,G,B)==R){
                              H=(int)(60*(G-B)/(return_max(R,G,B)-return_min(R,G,B)));
                        }
                        if(return_max(R,G,B)==G){
                              H=(int)(60*(B-R)/(return_max(R,G,B)-return_min(R,G,B)))+120;
                        }
                        else{
                              H=(int)(60*(B-G)/(return_max(R,G,B)-return_min(R,G,B)))+240;
                        }
                        //ここまで

                        //H<0, またはH>360の場合、色相Hの値を0~360の間で再定義
                        if(H<0){
                              H=H+360;
                        }
                        else if(H>360){
                              H=H-360;
                        }

                  //ここまで
                  //
                  //肌色画素の判定:条件[ (H<50||H>210)&&(-50<Cb&&Cb<-20)&&(20<Cr&&Cr<40) ]
                  //◀◀条件：(色相<50 または210<色相) かつ (-50<色差<-20) かつ (20<色差<40) かつ (輝度>100)▶▶
                        if(H!=-1000){
                              if((H<120||H>190)&&(-70<Cb&&Cb<-16)&&(10<Cr&&Cr<55)&&(Y>100)){
                                    output->p[k][j]=input->p[k][j];
                              }
                              else{
                                    output->p[k][j].r=255;          //肌色画素ではないのは白色に変換
                                    output->p[k][j].g=255;
                                    output->p[k][j].b=255;
                              }
                        }
                        R=0;G=0;B=0;
                        Y=0;Cb=0;Cr=0;                  //使用した変数の初期化
                        H=0;
                  }
            }
            printf("＊＊＊¥t¥t¥t肌色画素を抽出しました¥t¥t¥t＊＊＊¥n¥n");
}

/*＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊こ　こ　ま　で＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/

```

```
110
111 /*ラベル番号を修正する関数*/
112 void modify_label(int num1,int num2,RGB_PACKED_IMAGE *input, int label[input->rows][input->cols]){
113     int j,k;                                    //for文カウンター
114     for(k=0;k<input->rows;k++){
115         for(j=0;j<input->cols;j++){
116             if(label[k][j]==num1){              //ラベル番号num1をすべてnum2に置換する
117                 label[k][j]=num2;
118             }
119         }
120     }
121 }
122 /*ここまで*/
123
124
125 /*4近傍のラベルの最大値を返す関数*/
126 int search_4neighbors(int y,int x,RGB_PACKED_IMAGE *input,int label[input->rows][input->cols]){
127     int max=0;          //最大値
128
129     if(y-1>=0&&label[y-1][x]>max){                  //上のラベルを見る
130         max=label[y-1][x];
131     }
132     if(x-1>=0&&label[y][x-1]>max){                  //左のラベルを見る
133         max=label[y][x-1];
134     }
135     if(y+1<input->rows&&label[y+1][x]>max){             //下のラベルを見る
136         max=label[y+1][x];
137     }
138     if(x+1<input->cols&&label[y][x+1]>max){             //右のラベルを見る
139         max=label[y][x+1];
140     }
141     return max;
142 }
143 /*ここまで*/
144
145
146 /**********処理２．肌色領域のラベリング**********/
147 /＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊ラベリング関数＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
148 int labeling(RGB_PACKED_IMAGE *input,int R,int G,int B,int label[input->rows][input->cols]){
149     int j,k,num;                                //for文カウンター
150     int count=0;                                //ラベル最大値
151     int total_area;                             //領域の数
152
153     for(k=0;k<input->rows;k++){                     //ラベルを初期化
154         for(j=0;j<input->cols;j++){
155             label[k][j]=0;
156         }
157     }                                   //ここまで
158
159     for(k=0;k<input->rows;k++){
160         for(j=0;j<input->cols;j++){                 //位置(j,k)の画素が白でないかつラベル番号が0なら
161             if((input->p[k][j].r!=R||input->p[k][j].g!=G||input->p[k][j].b!=B)&&label[k][j]==0){
162                 num=search_4neighbors(k,j,input,label);     //4近傍のラベルの値をnumに調べる
163                 if(num==0){
164                     label[k][j]=++count;            //ラベル番号が0なら,ラベル番号に1加えて、位置(j,k)のラベル番号をcountにする
165                 }
166                 else{
167                     label[k][j]=num;            //それ以外なら4近傍のいずれかの値を代入
168                 }
169
170             }
171         }
172     }
173     printf("＊＊＊¥t¥t肌色領域のラベリングを終了しました¥t¥t＊＊＊¥n¥n");     //メッセージ出力
174     if(count>0){                                //ラベル番号が0より大きい(肌色領域が1つ以上存在する)
175         for(k=0;k<input->rows;k++){
176             for(j=0;j<input->cols;j++){
177                 if(label[k][j]!=0){
178                     num=search_4neighbors(k,j,input,label);     //4近傍のラベル値をもとに戻す
179                     if(num>label[k][j]){
180                         modify_label(num,label[k][j],input,label);
181                     }
182                 }
183             }
184         }
185         total_area=0;
186         for(k=0;k<input->rows;k++){
187             for(j=0;j<input->cols;j++){
188                 if(label[k][j]>total_area){
189                     total_area++;
190                     modify_label(label[k][j],total_area,input,label);
191                 }
192             }
193         }
194         printf("肌色領域の数：¥t¥t%d個¥n",total_area);              //メッセージ出力
195
196         return total_area;                      //最終的なラベルの最大値（肌色領域の個数）を返す
197     }
198     else{
199         printf("肌色領域の数：0個¥n");
200         return 0;
201     }
202 }
203
204
205 /＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊こ　こ　ま　で＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
206
```

```
207
208 /* 画像No.n の(x1,y1)<-->(x2,y2)の左右対称度を返す関数 */
209 /* 戻り値の範囲は [0, 1] で、0 に近いほど左右対称　　*/
210 double symmetry( RGB_PACKED_IMAGE *input, int x1, int y1, int x2, int y2 ){
211    int xcenter;   /* 横幅の中点　　*/
212    int xrange;    /* xを動かす範囲 */
213    double sum=0.0; /* 色差を表す量　*/
214    int counter=0;  /* 総画素数　　*/
215    int x,y;   /* 作業変数　　*/
216    double dif;    /* 色差（の二乗） */
217    double dr, dg, db, d;      /* 作業変数　　*/
218
219    xcenter = ( x2 + x1 ) / 2;  /* 横幅の中点 */
220    xrange = xcenter - x1;
221    /* 画像を折り返して調べます */
222    for(x=0;x<xrange;x++){
223       for(y=y1;y<=y2;y++){
224          counter++;  /* 画素数のカウンターに 1 を足す */
225          dif=0;      /* 初期化 */
226
227          dr = (double)( input->p[y][x1+x].r - input->p[y][x2-x].r );      //      R成分
228          if ( dr < 0 ) dr = - dr;
229
230          dg = (double)( input->p[y][x1+x].g - input->p[y][x2-x].g );      //      G成分
231          if ( dg < 0 ) dg = - dg;
232
233          db = (double)( input->p[y][x1+x].b - input->p[y][x2-x].b );      //      B成分
234          if ( db < 0 ) db = - db;
235
236          d = dr + dg + db;
237
238          dif += d / 255.0;
239
240          sum += dif / 3.0;
241       }
242    }
243    return sum / counter;
244 }
245 /*ここまで*/
246
```

```
247
248 /**********処理３．図形特徴の算出及び顔の領域の選別**********/
249 /＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊ラベル番号ごとの図形特徴の算出関数＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊/
250 int cal_area_label(RGB_PACKED_IMAGE *input,int lab[input->rows][input->cols],int n){
251     int j,k;            /* 制御変数 */
252     int sum_area=0;             /* 領域の面積 */
253     int rgbmin[3],rgbmax[3];  /* 範囲を調べる変数 */
254     int xmin=0,ymin=0,xmax=0,ymax=0;  /* 領域の左上と右下の座標 */
255       double W,H;     /* 横と縦 */
256       int colrange;     /* 色の範囲に関する変数 */
257       double symratio;  /* 左右対称度 */
258
259     //printf("¥n処理３と４：領域ごとの特徴を判定します¥n");
260       /* ラベルNo.n の領域を順番に調べます */
261
262     /* 各変数の初期化 */
263
264     /* 範囲の初期化 */
265     xmin=input->cols;   xmax=0;
266     ymin=input->rows;  ymax=0;
267
268     /* 面積の初期化 */
269     sum_area=0;
270
271     /* 色の範囲の初期化 */
272     for(j=0;j<3;j++){
273       rgbmin[j]=255;
274       rgbmax[j]=0;
275     }
276     /* ラベル(大域変数 label[][])を走査します */
277     for(k=0;k<input->rows;k++){                 //ラベル番号nの面積を計算する
278       for(j=0;j<input->cols;j++){
279         if(lab[k][j]==n){
280         /* 面積（領域）の更新 */
281           sum_area++;
282
283           /* 左上と右下の点の更新 */
284             if ( j < xmin ) xmin = j;
285             if ( j > xmax ) xmax = j;
286             if ( k < ymin ) ymin = k;
287              if ( k > ymax ) ymax = k;
288
289             /*色の範囲の更新*/
290           if(input->p[k][j].r<rgbmin[0])          // R成分
291             rgbmin[0]=input->p[k][j].r;
292           if(input->p[k][j].r>rgbmax[0])
293             rgbmax[0]=input->p[k][j].r;
294
295           if(input->p[k][j].g<rgbmin[1])          //G成分
296             rgbmin[1]=input->p[k][j].g;
297           if(input->p[k][j].g>rgbmax[1])
298             rgbmax[1]=input->p[k][j].g;
299
300           if(input->p[k][j].b<rgbmin[2])          //B成分
301             rgbmin[2]=input->p[k][j].b;
302           if(input->p[k][j].b>rgbmax[2])
303             rgbmax[2]=input->p[k][j].b;
304         } /*if文終了*/
305       }
306     } /*ラベル走査終了*/
307
308     /* 1) 面積の画像全体に対する比率＞AREARATIO の確認 */
309     if ( (double)(sum_area)/input->cols/input->rows >= AREARATIO ){
310         printf("=== No.%d ===¥n",n);
311         printf("  画像における面積率の条件をクリア：");
312          printf("面積率＝%f¥n",(double)(sum_area)/input->cols/input->rows);
313           W = xmax - xmin + 1;
314           H = ymax - ymin + 1;
315
316         /* 2) 外接四角形における面積率>AREARATE の確認 */
317         if ( ( sum_area / ( W * H ) ) > AREARATE ){
318           printf("  外接四角形における面積率の条件をクリア：");
319           printf("四角形面積率＝%f¥n",sum_area/(W*H));
320
321           /* 3) 縦横比の計算 */
322           if ( ( H/W > HWMIN ) && ( H/W < HWMAX )){
323             printf("  縦横比の条件をクリア：");
324             printf("縦横比＝%f¥n",H/W);
325
326             /* 4) 色の範囲の検査 */
327             colrange=0;
328             colrange = (rgbmax[0] - rgbmin[0]) + (rgbmax[1] - rgbmin[1]) + (rgbmax[2] - rgbmin[2]);
329             if ( colrange > 300 ){
330               printf("  色の範囲の条件もクリア：");
331               printf("色の範囲を表す量：%d¥n",colrange);
332
333               /* 5) 左右対称性の検査 */
334               symratio = symmetry( input, xmin, ymin, xmax, ymax );
335               if ( symratio < SYMMAX ){
336                 printf("  左右対称度の条件もクリア：");
337                 printf("左右対称度＝%f¥n",symratio);
338                 printf("★ No.%d の領域を顔の候補と判定します★¥n¥n",n);
339                 printf("¥t¥tラベル番号->%2d¥t¥t面積は%d¥t(%lf%)¥n¥n",n,sum_area,((double)sum_area/(input->rows*input->cols))*100.0);
340
341                 return n;     // 1) ~ 5)の条件を全て満たす場合、ラベル番号を顔領域としてラベル番号を返す（領域の選別）
342               }
343             }
344           }
345         }
346     }
347     else{
348         return 0;
349     }
350     printf("◆ No.%d の領域は顔の候補の条件をクリアできませんでした（顔候補から削除）◆¥n¥n",n);
351     return 0;
352 }
353 /＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊こ　こ　ま　で＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊/
354
```

```
355
356 /*＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊画像をすべて白色にする関数＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
357
358 void init(RGB_PACKED_IMAGE *input){
359     int k,l;
360     for(l=0;l<input->rows;l++){
361         for(k=0;k<input->cols;k++){
362             input->p[l][k].r=255;
363             input->p[l][k].g=255;
364             input->p[l][k].b=255;
365         }
366     }
367 }
368
369 /*＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊こ　こ　ま　で＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
370
371
372 /**********処理４．処理３の結果から、顔領域のみを抽出**********/
373 /*＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊顔領域のみを抽出する関数＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
374 void face(RGB_PACKED_IMAGE *input,RGB_PACKED_IMAGE *output,int lab[input->rows][input->cols],int n){
375     int k,l;
376     for(l=0;l<input->rows;l++){
377         for(k=0;k<input->cols;k++){
378             if(lab[l][k]==n){
379                 output->p[l][k].r=input->p[l][k].r;
380                 output->p[l][k].g=input->p[l][k].g;
381                 output->p[l][k].b=input->p[l][k].b;
382             }
383             else{
384                 if(input->p[l][k].r!=255&&input->p[l][k].g!=255&&input->p[l][k].b!=255){     /*何もしない*/     }
385                 else{
386                     output->p[l][k].r=255;
387                     output->p[l][k].g=255;
388                     output->p[l][k].b=255;
389                 }
390             }
391         }
392     }
393 }
394 /*＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊こ　こ　ま　で＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
395
396
397 /*＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊領域の左上端,右下端を判定する関数＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
398 void edge(RGB_PACKED_IMAGE *input,int lab[input->rows][input->cols],int num,int ul[2],int lr[2]){
399     int j,k;                                //ul[0]=左上のx座標,ul[1]=左上のy座標
400     int flag=0;                             //lr[0]=右下のx座標 lr[1]=右下のy座標
401     ul[0]=0;lr[0]=0;
402     ul[1]=0;lr[1]=0;
403     for(k=0;k<input->rows;k++){                     //左上のy座標の決定
404         for(j=0;j<input->cols;j++){
405             if(lab[k][j]==num){
406                 if(flag==0){
407                     ul[1]=k;
408                     flag=1;
409                 }
410             }
411         }
412     }
413
414     flag=0;
415
416     for(k=0;k<input->cols;k++){                     //左上のx座標の決定
417         for(j=0;j<input->rows;j++){
418             if(lab[j][k]==num){
419                 if(flag==0){
420                     ul[0]=k;
421                     flag=1;
422                 }
423             }
424         }
425     }
426     flag=0;
427     for(k=input->rows-1;k>=0;k--){                   //右下のy座標の決定
428         for(j=0;j<input->cols;j++){
429             if(lab[k][j]==num){
430                 if(flag==0){
431                     lr[1]=k;
432                     flag=1;
433                 }
434             }
435         }
436     }
437     flag=0;
438     for(k=input->cols-1;k>=0;k--){                   //右下のx座標の決定
439         for(j=0;j<input->rows;j++){
440             if(lab[j][k]==num){
441                 if(flag==0){
442                     lr[0]=k;
443                     flag=1;
444                 }
445             }
446         }
447     }
448
449     if((double)(lr[1]-ul[1])/(lr[0]-ul[0])>FACE_RATE){      //顔の縦横比を補正する
450         printf("¥t＜＜＜＜ラベル番号->%dの顔の縦横比が異常です!!!!＞＞＞＞¥n",num);
451
452         lr[1]=ul[1]+(int)(FACE_RATE*(lr[0]-ul[0]));
453
454         printf("¥t¥t_____顔の縦横比を修正しました_____¥t¥n¥n");
455     }
456     printf("ラベル番号%dの領域の左上の座標は(%d,%d),右下の座標は(%d,%d)です¥n¥n",num,ul[0],ul[1],lr[0],lr[1]);
457 }                                       //メッセージ出力
458
459 /*＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊こ　こ　ま　で＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
460
```

```
461
462 /**********処理５．領域の左上端,右下端の座標を用いて、顔の周りを緑の外接四角形で囲む**********/
463 /＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊四角形を書く関数＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
464 void draw_rect(RGB_PACKED_IMAGE *input,int x1,int y1,int x2,int y2,int bold){
465     int i,j;
466
467     //水平方向に直線を書く
468     for(j=x1;j<=x2;j++){
469         for(i=-bold/2;i<=bold/2;i++){
470             if(0<=y1+i){
471                 input->p[y1+i][j].r=0;
472                 input->p[y1+i][j].g=255;
473                 input->p[y1+i][j].b=0;
474             }
475         }
476     }
477     for(j=x1;j<=x2;j++){
478         for(i=-bold/2;i<=bold/2;i++){
479             if(input->rows>y2+i){
480                 input->p[y2+i][j].r=0;
481                 input->p[y2+i][j].g=255;
482                 input->p[y2+i][j].b=0;
483             }
484         }
485     }
486
487     //垂直方向に直線を書く
488     for(j=y1;j<=y2;j++){
489         for(i=-bold/2;i<=bold/2;i++){
490             if(0<=x1+i){
491                 input->p[j][x1+i].r=0;
492                 input->p[j][x1+i].g=255;
493                 input->p[j][x1+i].b=0;
494             }
495         }
496     }
497     for(j=y1;j<=y2;j++){
498         for(i=-bold/2;i<=bold/2;i++){
499             if(input->cols>x2+i){
500                 input->p[j][x2+i].r=0;
501                 input->p[j][x2+i].g=255;
502                 input->p[j][x2+i].b=0;
503             }
504         }
505     }
506 }
507
508 /＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊こ　こ　ま　で＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
509
510
511 /**********処理６．領域の左上端,右下端の座標を用いて、顔の大きさから目の部位を計算し、モザイク加工**********/
512 /＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊モザイク型の四角形を書く関数＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
513 void draw_rect_mosaic(RGB_PACKED_IMAGE *input,int x1,int y1,int x2,int y2,int bold){
514     int i,j;
515     int rr, gg, bb;
516     int s = 5;
517     int Y1, Y2;
518
519     if(y2 - y1 > 75){
520         Y1 = (int)( y1 + (y2 - y1) /4);
521         Y2 = Y1 + 25;
522     }
523     else{           // 顔の枠が小さい場合
524         Y1 = (int)( y1 + (y2 - y1) / 5);
525         Y2 = Y1 + 15;
526     }
527
528     for(j=x1;j<=x2;j++){
529         for(i=Y1;i<=Y2;i++){
530             rr=input->p[i][j].r;
531             gg=input->p[i][j].g;
532             bb=input->p[i][j].b;
533
534             rr=rr/(s*s)+(rand() % 96)-48; // // 各ピクセルの色を平均し-48～48の乱数加算
535              gg=gg/(s*s)+(rand() % 96)-48;
536              bb=bb/(s*s)+(rand() % 96)-48;
537
538             input->p[i][j].r=(BYTE)rr; // ブロックの色計算
539             input->p[i][j].g=(BYTE)gg;
540             input->p[i][j].b=(BYTE)bb;
541
542             rr = 0; gg = 0; bb = 0;
543         }
544     }
545 }
546
547 /＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊こ　こ　ま　で＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
548
```

```
549
550 /*＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊　　メイン処理　　＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊*/
551
552 #ifdef __STDC__
553 int
554 main( int argc, char *argv[] )
555 #else
556 int
557 main( argc, argv )
558     int argc ;
559     char *argv[] ;
560 #endif
561 {
562
563     char *name_img_in = "image2.ppm" ;                    /* 入力画像ファイル名 */
564     char *name_img_out_1 = "output1_Flesh_color_area.ppm" ;       /* 肌色領域の画像ファイル名 */
565     char *name_img_out_2 = "output2_face_area.ppm" ;              /* 顔画像ファイル名 */
566     char *name_img_out_3 = "output3_result_detect.ppm" ;       /* 顔抽出結果画像ファイル名 */
567     char *name_img_out_4 = "output4_result_mosaic.ppm" ;       /* モザイク処理結果画像ファイル名 */
568
569     RGB_PACKED_IMAGE *image_in ;      /* カラー画像用構造体(顔抽出用) */
570     RGB_PACKED_IMAGE *image_in_1 ;      /* カラー画像用構造体（モザイク加工用） */
571     RGB_PACKED_IMAGE *image_Skin_P ;/*肌色画素検出用*/
572     RGB_PACKED_IMAGE *image_face ;      /*結果出力画像用*/
573
574     /* コマンドラインでファイル名が与えられた場合の処理 */
575     if ( argc == 1 ){
576         printf("■■■¥t¥t入力画像をコマンドラインで入力してください。¥t¥t■■■¥n");
577         printf("■■■¥t現在はファイル名：%sが入力画像に設定されています¥t■■■¥n¥n¥n",name_img_in);
578     }
579     if ( argc >= 2 ) name_img_in = argv[1] ;
580     if ( argc >= 3 ) name_img_out_1 = argv[2] ;
581
582     /* 入力画像ファイルのオープンと画像データ獲得 */
583     if (!( image_in = readRGBPackedImage( name_img_in ))) {
584         printError( name_img_in ) ;
585         return(1) ;
586     }
587
588     /* 入力画像ファイルのオープンと画像データ獲得 */
589     if (!( image_in_1 = readRGBPackedImage( name_img_in ))) {
590         printError( name_img_in ) ;
591         return(1) ;
592      }
593
594     /*各種構造体や変数の宣言*/
595     image_Skin_P = allocRGBPackedImage(image_in->cols,image_in->rows);      //肌色画素検出用
596      image_face = allocRGBPackedImage(image_in->cols,image_in->rows);      //結果出力画像用
597      int label[image_in->rows][image_in->cols];      //ラベル配列
598      int UL[2],LR[2];            //領域の端を格納する配列
599      int a,cnt=0,temp=0;            //カウンター
600      int face_label_num[NOP]={0};      //顔の候補の領域のラベル番号を格納する配列
601      int SOP=0;      //検出できた人数をカウント用
```

```
602
603     /*処理開始*/
604     skin_color_detection(image_in,image_Skin_P);          //処理１．入力画像から肌色画素を抽出する
605
606     int number = labeling(image_Skin_P,255,255,255,label);          //処理２．肌色領域の数を調べ、numeberに代入
607
608     if ( writeRGBPackedImage( image_Skin_P, name_img_out_1 ) == HAS_ERROR ) {          //肌色領域をすべて画像ファイルに出力
609         printError( name_img_out_1 ) ;
610         return(1) ;
611     }
612     printf("¥n＊＊＊¥t¥t肌色領域をラベリングした結果を出力しました¥t＊＊＊¥n＊＊＊¥t¥tファイル名：%s¥t¥t＊＊＊¥n¥n",name_img_out_1);
613
614     printf("**************************¥t顔の候補¥t**************************¥n");     //メッセージ出力
615
616     for(a=1;a<=number;a++){                                 //顔候補の領域のラベル番号を調べる。
617         temp=cal_area_label(image_Skin_P,label,a);                  //処理３．図形特徴の算出及び顔の領域の選別
618         if(temp!=0){
619             if(cnt==NOP){}
620             else{
621                 face_label_num[cnt]=temp;                   //顔候補の領域のラベル番号を代入
622                 cnt++;
623             }
624         }
625
626     }
627     printf("********************************************************************************¥n");     //メッセージ出力
628
629     init(image_face);                                   //出力画像の初期化
630
631     for(a=0;a<NOP;a++){
632         if(face_label_num[a]!=0){
633             face(image_Skin_P,image_face,label,face_label_num[a]);     //処理４．処理３の結果から、顔領域のみを抽出し、書き込む
634         }
635     }
636     printf("¥n＊＊＊¥t¥t¥t顔候補の領域を出力しました¥t¥t＊＊＊¥n＊＊＊¥t¥t¥tファイル名：%s¥t¥t＊＊＊¥n¥n",name_img_out_2);
637                                         //メッセージ出力
638
639     for(a=0;a<NOP;a++){                                 //顔候補の領域の端を調べる。
640         if(face_label_num[a]!=0){
641             SOP++;
642             edge(image_Skin_P,label,face_label_num[a],UL,LR);     //領域の左上端,右下端を判定
643             draw_rect(image_in,UL[0],UL[1],LR[0],LR[1],4);          //処理５．顔の周りを緑の外接四角形で囲む
644             draw_rect_mosaic(image_in_1,UL[0],UL[1],LR[0],LR[1],4);     //処理６．顔の大きさから目の部位を計算し、モザイク加工
645         }
646     }
647     printf("¥n＊＊＊¥t¥t¥t%d人分の顔の判定を判定できました¥t¥t＊＊＊¥n",SOP);
648
649
650     if ( writeRGBPackedImage( image_face, name_img_out_2 ) == HAS_ERROR ) {          //顔候補の領域を画像ファイルに出力
651         printError( name_img_out_2 ) ;
652         return(1) ;
653     }
654
655
656     /* 出力画像ファイルのオープンと画像データの書込み */
657
658      if ( writeRGBPackedImage( image_in, name_img_out_3 ) == HAS_ERROR ) {          //最終結果を出力
659             printError( name_img_out_3 ) ;
660             return(1) ;
661      }
662
663      printf("¥n＊＊＊¥t¥t¥t顔抽出結果の画像を出力しました¥t¥t＊＊＊¥n＊＊＊¥t¥t¥tファイル名：%s¥t¥t＊＊＊¥n¥n",name_img_out_3);
664         //メッセージ出力
665
666      if ( writeRGBPackedImage( image_in_1, name_img_out_4 ) == HAS_ERROR ) {          //最終結果を出力
667             printError( name_img_out_3 ) ;
668             return(1) ;
669      }
670
671      printf("¥n＊＊＊¥t¥t¥tモザイク加工結果の画像を出力しました¥t¥t＊＊＊¥n＊＊＊¥t¥t¥tファイル名：%s¥t¥t＊＊＊¥n¥n",name_img_out_4);
672         //メッセージ出力
673
674      return(0);
675
676 }
677
```

ソースコード 1. 入力画像の顔を検出・抽出し、モザイク加工するソースコード

図 1. 1 人の女性が映った入力用のカラー画像 [3)]

図 2. 図 1 の肌色画素を検出した画像

図 3. 図 2 の肌色画素領域から選別した顔領域のみを抽出した画像

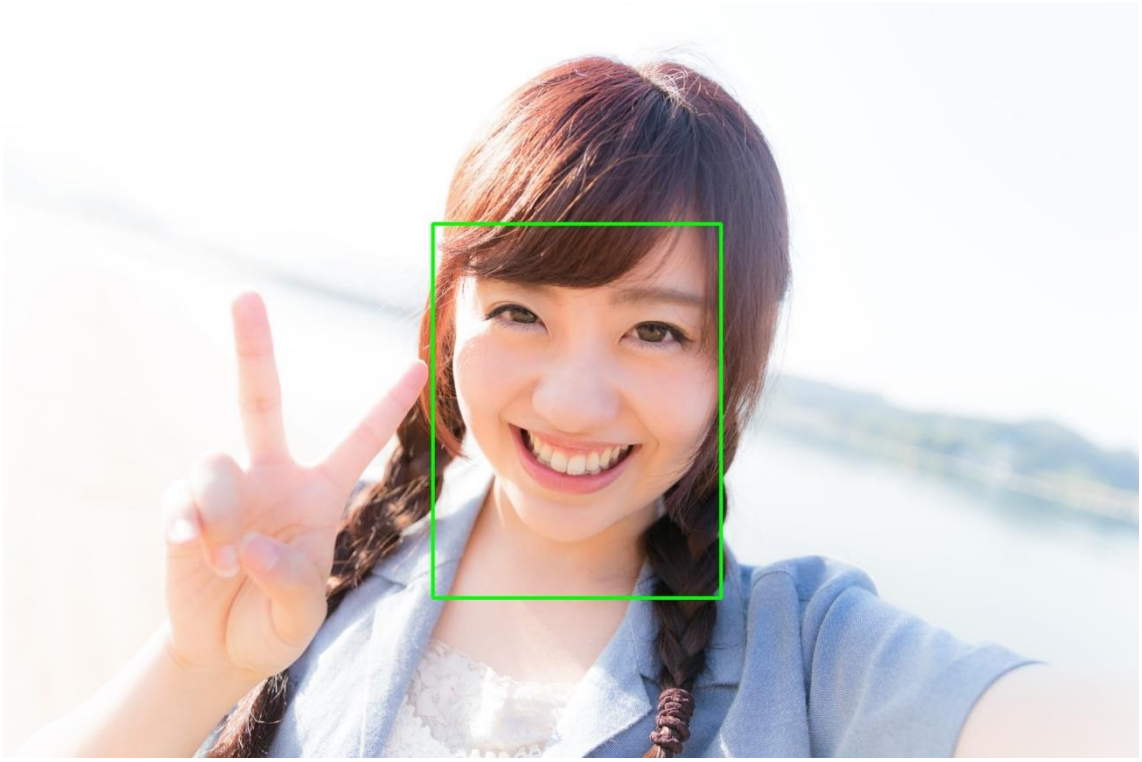

図 4. 図 3 の顔領域を緑の外接四角形で囲んだ画像

図 5. 図 4 の外接四角形から目の位置を特定し、モザイク加工を施した画像

図 6. 5 人の人物が映った入力用のカラー画像 [4)]

図 7. 図 6 の肌色画素を検出した画像

図 8. 図 7 の肌色画素領域から選別した顔領域のみを抽出した画像

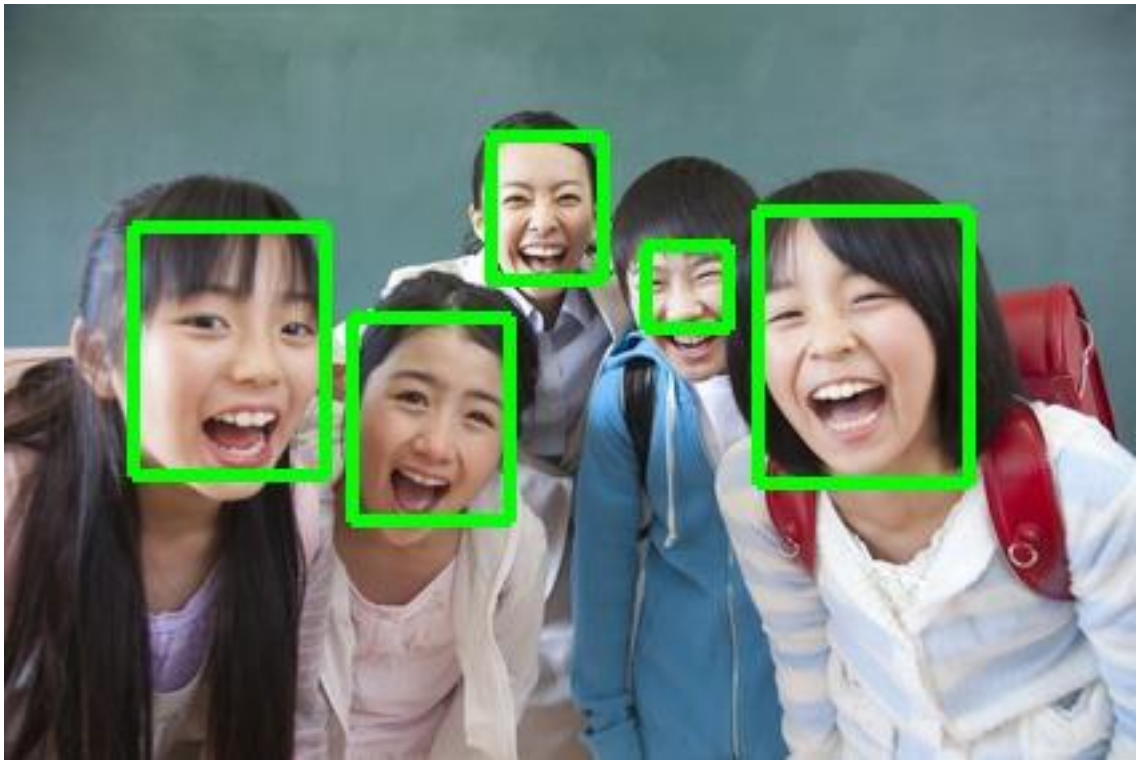

図 9. 図 8 の顔領域を緑の外接四角形で囲んだ画像

図 10. 図 9 の外接四角形から目の位置を特定し、モザイク加工を施した画像

図 11. 10 人の子供が映った入力用のカラー画像 [5)]

図 12. 図 11 の肌色画素を検出した画像

図 13. 図 12 の肌色画素領域から選別した顔領域のみを抽出した画像

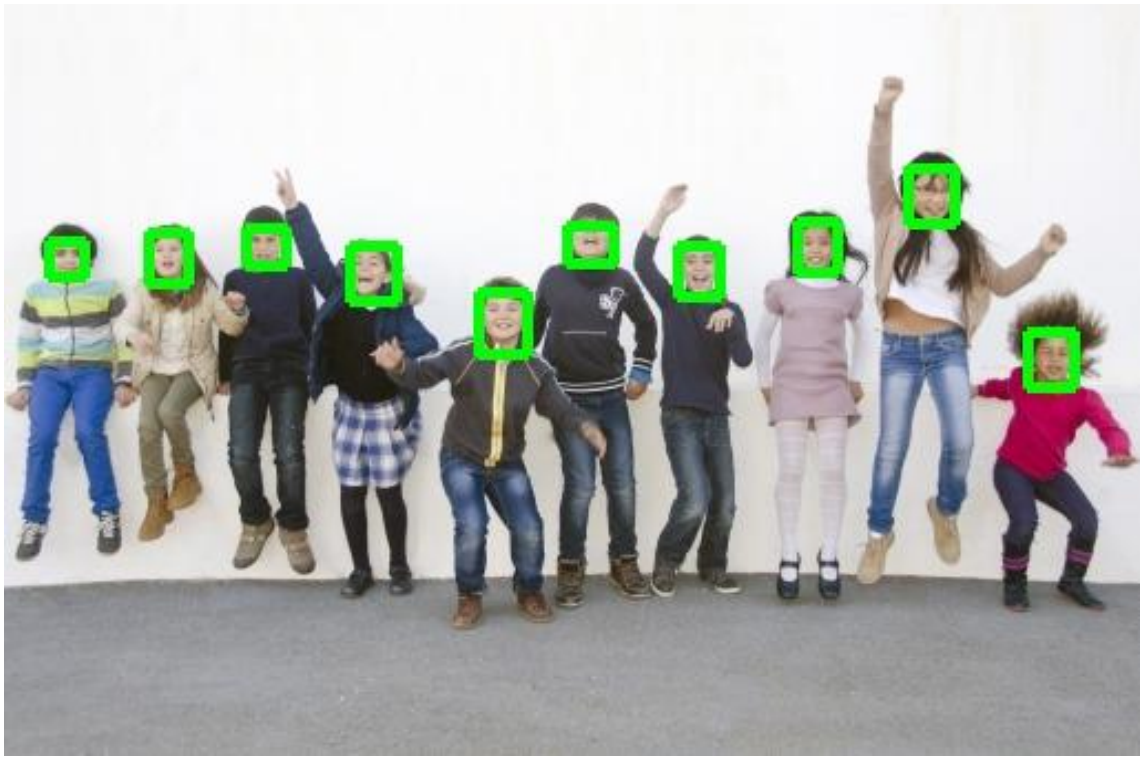

図 14. 図 13 の顔領域を緑の外接四角形で囲んだ画像

図 15. 図 14 の外接四角形から目の位置を特定し、モザイク加工を施した画像

図 16. 33 人の人物が映った入力用のカラー画像 [6)]

図 17. 図 16 の肌色画素を検出した画像

図 18. 図 17 の肌色画素領域から選別した顔領域のみを抽出した画像

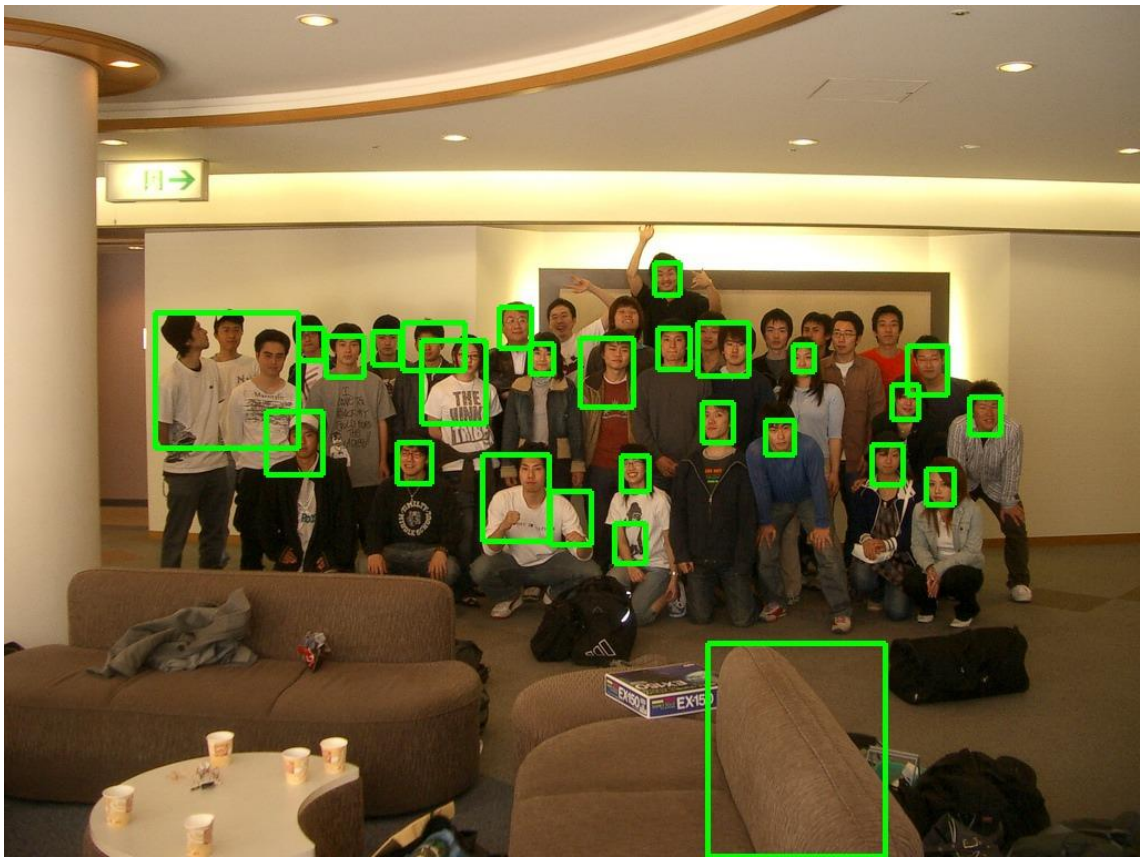

図 19. 図 18 の顔領域を緑の外接四角形で囲んだ画像

図 20. 図 19 の外接四角形から目の位置を特定し、モザイク加工を施した画像

５.考察

□本実験についての検討

本実験における課題では、「色相 H」を用いることで、入力画像から顔を検出・抽出し、抽出した顔の「目」の部分にモザイク加工を施すことが目的である。

図1, 6, 11, 16のそれぞれ条件の異なるカラー画像を入力画像とし、全ての入力画像に対し、画像内の人物の目の部分にモザイク加工を施し、出力することができれば目的を達成できたことになる。

各入力画像での「成功・失敗」について、入力画像として図 1 を用いた際の結果説明を「検証 1」で行い、同様に、入力画像として図 6, 11, 16 を用いた際の検討を「検証 2～4」で行う。

□検証 1（成功例）

この検証では、図 1 の画像のような「顔の近くに顔と同じ大きさの肌色領域（手）が存在する場合」に、手を誤抽出することなく、顔のみを検出し、モザイク加工を施すことができれば、目的を達成できたことになる。

図 2～5 から、図 1 の入力画像の女性の顔のみにモザイク加工を施せているので、検証 1 における目的は達成できているといえる。

目的が達成できた理由は、顔候補領域の図形特徴の算出において、「手」の肌色孤立領域の外接四角形を考えると、「外接四角形中の肌色画素の面積率」が小さく、また、「手の肌色孤立領域の左右対称性がない」ため、手を含む、顔以外の肌色孤立領域は顔ではないと判定されたからであると考察できる。

また、図 1 の画像の女性の顔は大きく、くっきりと映っているため、顔の肌色領域を広く、かつ正確に検出することができたことも目的達成の理由であると考察できる。

□検証 2（成功例）
この検証では、図 6 の画像のような「顔が大きく映った人と小さく映った人が存在する場合・顔が少し隠れて映った人が存在する場合」に、画像内の顔の大きさ・映った際の悪条件に関わらず、各顔の大きさに合わせて顔を認識し（外接四角形で囲む)、顔の大きさに合わせて適切にモザイク加工を施すことができれば、目的を達成できたことになる。
図 7~ 10 から、図 6 の入力画像の左から 3 番目の奥にいる顔の小さく映った人物や、左から 4 番目にいる顔が少し隠れた人物の顔を的確に認識し、外接四角形を描けている。
さらに、顔の大きさに関わらず、適切なサイズのモザイク加工を施せているので、検証 2 における目的は達成できているといえる。

目的が達成できた理由は、画像内の小さな顔を検出させるために、顔の画像全体の面積に対する下限の条件を 0.2%に下げたことで、小さな顔の肌色領域も顔候補領域として図形特徴算出の条件で「顔」として選別することができたからであると考察できる。

また、画像の背景色が「濃い緑」であり、図 6 の人物の周りに肌色候補となる領域が存在しない。そして、顔が少し隠れている人物も、隣の人物の黒髪の色で、顔領域の境界が明確となっている。このため、図 7 の抽出した肌色領域も各顔の領域が一体化せずに、孤立領域となったため、各孤立領域が「顔」として判定されたと考察することができる。

さらに、顔の外接四角形の高さに応じてモザイクをかける位置の指定及びモザイクの縦幅の太さを微調整したことにより、適切な位置・サイズで、「目」の部分にモザイク加工を施せたと考察することができる。

□検証 3（成功例）
この検証では、図 11 の画像のような、「大人数が映っており、映った顔のサイズが極端に小さい場合」に、体全体が映ったことで、映った体の方が顔よりも大きいという悪条件に関わらず、画像内の小さな顔を全て検出し、適切にモザイク加工を施すことができれば、目的を達成できたことになる。
図 12~ 15 から、図 11 の入力画像中の全ての人物（10 人）に対して、的確に顔を認識し、外接四角形を描くことができ、適切なサイズのモザイク加工を施せているので、検証 3 における目的は達成できているといえる。

目的が達成できた理由は、画像内の小さな顔を検出させるために、顔の画像全体の面積に対する下限の条件を 0.1%に下げたことで、極端に小さな顔の肌色領域も顔候補領域として図形特徴算出の条件で「顔」として選別することができたからであると考察できる。

また、画像の背景色が「白」であり、図 11 の人物の周りに顔候補となる「顔の図形特徴を持つ肌色領域」が存在しない。そして、画像内の人物の顔は全て一定の大きさ・間隔の距離を保って映っており、顔領域の境界が明確となっている。このため、図 12 の抽出した肌色領域も各顔の領域が一体化せずに、孤立領域となり、さらに、全ての顔の大きさに差がないため、顔領域のみがが「顔」として判定されたと考察することができる。

□検証 4（失敗例）
この検証では、図 16 のような「大人数が映っており、映った顔のサイズが極端に小さく、かつ、画像全体の輝度が低く、背景色が肌色に近い色の場合」に、背景色の悪条件に関わらず、画像内の小さな顔を全て検出し適切にモザイク加工を施すことができれば、目的を達成できたことになる。
図 17 ~ 20 から、図 16 の入力画像中の全ての人物（33 人）に対して、的確に顔を認識し、適切なサイズのモザイク加工を施すことができたのは 20 人で、図 19 を見ると、複数人を一つの顔と認識して外接四角形を描いており、さらに、手前のソファーを「顔」として認識しているため、検証 4 における目的は達成できなかったいえる。

目的が達成できなかった原因は、主に 3 つあると考察できる。

1 つ目の原因は、「肌色の色相抽出条件を広げてしまった」ことである。画像全体の輝度が低いため、画像内の顔の色も暗くなっており、検証 1～3 の色相の条件では顔を抽出することができない。このため、色相抽出条件を広げることで、全ての顔領域を抽出することには成功したが、顔以外の「肌色領域の壁・柱」や「肌色よりも濃い色のソファー」、さらに、「複数の人物」や「人物と背景色が肌色に似た色の壁」を一つにまとまった領域として抽出してしまっている。その結果、図 17 のような、複数の人物が存在する孤立領域や、色相条件を満たす人ではない多くの肌色領域を検出してしまった。

2 つ目の原因は、「顔の画像全体の面積に対する下限の条件を 0.05%に下げた」ことである。33 人の人物が映っているので、必然的に画像内に対する顔の大きさが極端に小さくなる。このため、顔の画像全体の面積に対する下限の条件を極限まで下げることで、極端に小さな顔を抽出することができたが、「顔以外の肌色領域」を多く検出してしまったことである。その結果、本来抽出されるべきでない大きさの肌色領域が「顔の条件」を満たし、顔として検出してしまった。図 19 における手前のソファーや、画像中央の人物の手や腕が誤検出されていることが分かる。

3 つ目の原因は、画像内における要因で、「人物の背景色が肌色に近い色である」という悪条件である。入力画像の図 16 と図 17 を見ると、画像内における右上の複数人の人物と壁の色がほとんど同色であり、「一つのまとまった領域」として抽出されている。その結果、この領域が顔かどうかを選別されるので、図 18 を見ると、「顔でない」と判断され、排除されていることが分かる。

□理由に基づく改善案

・検証１～３の成功例から、さらなる性能向上の案を検討する。

検証１～３では、結果としては、的確に顔を認識し、モザイク加工を施すことができたが、現在のプログラムでは、顔の入力画像全体の面積に対する下限の条件を画像毎に変更する必要がある。このため、顔の入力画像全体に対する下限をプログラム内で自動的に決定するように改良すると実用性が向上すると考えた。
具体的には、「厳密な肌色の色相条件」を定義しておき、その条件を満たす肌色領域の面積の和を全て計算し、その和を孤立領域の個数（すなわち画像中の人数）で割ることで、一人あたりの顔の大きさの平均を求めることができる。その平均値から、検出する顔の大きさを推測することができ、条件設定の自動化が可能になるのではないかと考察できる。

しかし、検証４のような、画像全体の輝度が低い場合、画像内の顔領域が「厳密な肌色領域の色相条件」を満たさない可能性もある。すなわち、読み込む画像の明るさによって、色相や色差の判定でのパラメータ調節が必要になるという課題が生じてくる。この問題を解決するために、画像中央部の各画素の輝度の和を計算し、その平均値をとることで、画像全体の暗さを求めることができる。その平均値から、検出する際の画像の暗さを推測することができ、「厳密な肌色領域の色相条件」の調節を自動化することが可能になるのではないかと考察できる。

その他の性能向上・改善点としては、本プログラムの検証では取り扱わなかった「画像に映った顔が傾いている場合の的確なモザイク加工処理の実現」がある。
この改善策としては、顔判別の際に用いた「顔の左右対称性の算出」を応用し、「抽出した顔の対称度を複数の角度の軸で計算し、一番対称性が高い軸に合わせてモザイク加工を施す」という案を考察することができる。

・検証 4 の失敗例から、失敗原因の解決策を検討する。

検証 4 では、3 つの原因により目的を達成することができなかった。1 つ目と 3 つ目の原因である「肌色画素の色相抽出条件を広げてしまったこと」により、「人物の背景色が肌色に近い色である」場合に背景と顔を一つのまとまった領域として抽出してしまったが、これは、性能向上案で述べた方法により、「画像内の輝度に依存しない厳密な肌色の色相条件」を設けることによりで解決できる。
ここでは、2 つ目の原因「顔の画像全体の面積に対する下限の条件を極限まで下げたこと」について、解決策を検討する。

2 つ目の原因によって、顔以外の肌色領域を多く検出してしまっており、本来抽出されるべきでない大きさの肌色領域が「顔の条件」を満たす場合、顔として検出してしまう。
本実験で作成したプログラムはスマートフォン等に用いられている顔検出技術に比べると極めて簡易なものであるので、図 16 のような悪条件の場合に「顔でない領域」を誤抽出してしまう場合がある。例えば、正座をしている人間の「膝」などは、色は肌色で、形状も四角形であり、外接四角形内での面積も許容範囲内に収まり、左右対称度も高いため、本プログラムにおける顔候補領域としての条件をすべて満たすものは無条件で「顔である」と判定してしまう。

よって、本実験で作成したプログラムの顔の検出精度を改善するためには、より多くの・より精度の高い「顔らしさ」の図形特徴を用いて領域を厳密に選択する必要がある。
顔領域選択の厳密化を解決する手法として、平均的な顔パターンとマッチングを行って違いを求める「テンプレートマッチング法」がある。
具体得的には、まず、入力用のカラー画像をグレイ化し、それを二値化する。次に、テンプレートの白黒画像と、二値化した白黒濃淡画像の一部を比較して一致する画素ごとに点数をつけ、一定値以上、ずなわち、最も一致度の高いテンプレートの形状であれば、顔として検出する。[7)]
この「テンプレートマッチング法」を本プログラムの「顔の図形特徴の判定条件」に組み込むことで、より精度の高い顔検出が可能になり、検証 4 における失敗原因を解決できると考えられる。
さらに、テンプレートマッチング法により、「目のテンプレート画像」を用意することで、今回は技術的に不可能であった「目の検出」も、可能になるのではないかと考察できる。
これにより、本プログラムにおいて、「目の位置を仮定してモザイク加工」を施すのではなく「目の位置を検出し、検出部分をモザイク加工」を施すことができるので、モザイク処理の精度・性能向上に繋がると考察できる。

その他の方法として、これまでに知られている顔特有の図形特徴量（Haar-like 特徴など）を積極的に用いる方法が考えられる。[1)] また、複数の特徴量を基にした認識（分類器）に、より分類精度が高いサポートベクターマシンやブースティングなどの方式を使う方法、発展的には、大量の顔のサンプルデータを用いてパターン認識を行う機械学習の手法が考えられる。

６.参考文献・画像出典

1) 長尾智晴　「図解入門よくわかる最新　画像処理アルゴリズムの基本と仕組み」
株式会社 秀和システム (2012 年 10 月 1 日　発行) [98 ページ] [104 ページ]

2) 晶達慶仁　「詳解　画像処理プログラミング」
ソフトバンククリエイティブ株式会社 (2008 年 3 月 30 日) [253 ページ]

3) 「PAKUTASO」
＜ https://www.pakutaso.com/20160947249post-8923.html ＞
（2017 年 11 月 9 日閲覧）

4) 「Fotolia Fan!」
＜ http://giveusgvdesktop.com/sample/%E5%86%99%E7%9C%9F%E7%B4%A0%E6%9D%90%E5%AD%90%E4%BE%9B/ ＞
（2017 年 11 月 9 日閲覧）

5) 「photoAC」
＜ https://www.photo-ac.com/main/search?q=mdfk033&srt=dlrank ＞
（2017 年 11 月 9 日閲覧）

6) 「商用無料の写真検索さん」
＜ http://www.igosso.net/se.cgi?q=%E9%9B%86%E5%90%88%E5%86%99%E7%9C%9F ＞
（2017 年 11 月 9 日閲覧）

7) 「テンプレートマッチングによる顔検出」
＜ http://leo.ec.t.kanazawa-u.ac.jp/~nakayama/edu/ind_res_2009-x/045baba.pdf ＞
（2017 年 11 月 22 日閲覧）