令和５年度　特別研究報告書

画像からの身体的特徴取得と
それによるおしゃれアドバイスの検討

龍谷大学　先端理工学部　知能情報メディア課程

Y200102　石原 智規

指導教員　三好 力 教授

内容梗概

おしゃれを重視するような傾向がある今の社会において, おしゃれに自信の無い人には何かアドバイスがあった方がいいのではないかと考えた. そこで, この研究では画像処理によって取得した身体的特徴と服の色の傾向やパターンから対象者におすすめの服の色の提案ができるかを明らかにする. 身体的特徴を選んだ理由としては, 鏡の前で服をあてがうことがあると思うのだが, その際に考慮しているのは自身の身体的特徴との相性なのではないかと感じたためである. そして服をそのまま提案するよりも服の色を提案した方が, 対象者にある程度の選択の幅を持ちつつアドバイスできると考えた. 実験では服選びに影響があると思われる顔立ちや顔の雰囲気に関係する「髪色」と「肌の色」,「髪の長さ」,「目と目の距離」と服の色を複数枚の画像から取得して傾向やパターンを知識ベースとし, 知識ベースから服の色の提案が可能か検証する.

目次

第1章 緒言

今回おしゃれに着目した理由は，若者が身だしなみを重視している傾向にあり，その傾向が年々高くなっているからである．実際，TBS が行なった総合嗜好調査では図1[1]のような結果が出ている．図1では，男女ともに若者の方がおしゃれを重視する傾向にあり，年代によっては傾向が下がるところがあるものの，総合的に見てみるとおしゃれ重視傾向が年々高くなっていることが分かる．

そのため，おしゃれに関するアドバイスを与えることはこの先も需要が高まっていくのではないかと考えた．以下，おしゃれに関するアドバイスを「おしゃれアドバイス」と呼ぶ．

おしゃれアドバイスを行なう上で必要なものが，その人の身体的特徴であると考えた．その理由を次に述べる．

服を選ぶ際に，鏡の前で服をあてがうことがあると思う．大体の場合，これは自分に似合っているかを確認していると思うのだが，その際に考慮しているのは自身の身体的特徴との相性なのではないかと感じた．そこで，身体的特徴はおしゃれアドバイスを行なう上で重要なのではないかと考えたのである．

そこで今回，おしゃれに自信の無い人へ身体的特徴から対象者にあったおすすめの服の色を提供するシステム作成について検討した．このシステムのことを以下，「おしゃれアドバイスシステム」と呼ぶ．服の色をアドバイスする理由としては，色をアドバイスすることで対象者の選択の幅をある程度保持できると考えたためである．

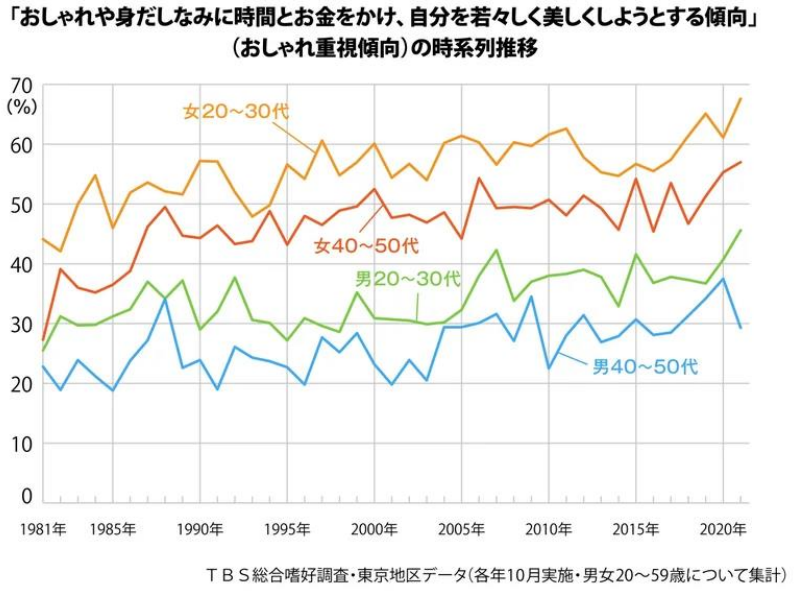

図1．TBS 総合嗜好調査による年代男女別おしゃれ重視傾向のグラフ
（調査情報デジタルより引用）[1]

第2章 既存技術

2.1 パーソナルカラー診断

身体的特徴から似合う色を提供するということにおいて,既存のものとしてパーソナルカラー診断というものがある．これは目の色や肌の色，髪の色などの主に身体的特徴の色部分の情報をアンケートなどから取得し，その人に合った色を提供するというものである[2].

2.2 SENSY CLOSET

ファッション人工知能アプリで，洋服に関することをスマートフォンで管理できる．デジタル・クローゼット機能では自分の持っている洋服の管理ができ，洋服を買いに行く際に自分がどんな服を持っていたのかを確認しながら買い物ができるという利点を持つ.

コーディネート・キャンバス機能では写真を撮って登録した洋服から自在にコーディネートを作ることができる.

他にも洋服に関する様々な機能があり,コーディネートを学習しておすすめの服を提案してくれる機能もあるという[３].

2.3 既存技術の問題点

パーソナルカラー診断はアンケート形式のものが多いが,アンケート形式の場合事実確認が難しく，情報の正確性に疑問が残る．またパーソナルカラー診断ではその人に似合う色を提案するが，これは服の色というわけでなく，化粧品関連などの色も含んだ全体的な色を指す．本研究ではおしゃれに自信の無い人へ服の色のみを提案をしたいため,この技術を使用することは相応しくないと考える．さらにほとんど体格などの色以外の身体的特徴は扱っていないため，色関連以外に関する特徴による服の色への影響について調べたい.

SENSY CLOSET では服を提案しているが，色を提案する場合よりも選択肢の幅が狭まることから，服の好みによっては良いアドバイスにならない可能性がある.

以上のことから,おしゃれに自信の無い人へ身体的特徴から服の色を提案するシステムはまだ無いことが分かった.

第3章 提案手法

3.1 既存技術の問題点の解決

2.3 で挙げた問題点の解決方法について考える．まずパーソナルカラー診断のアンケート形式で情報の正確性に難があることに関しては,ユーザー情報を画像処理で取得する方が正確な情報を得られると考えた.そして色以外の身体的特徴による服の色への影響も調査したいという点に関しては,服を購入する時にまず考慮することはサイズであると思ったため,サイズに関係する身長，体重といった体格が似合う服の色を提案するにあたって適切であると考えた.そのため色以外の身体的特徴については「身長」と「体重」を要素として加え，まずは身長と体重を画像処理で取得できるかについて調査した．そして SENSY CLOSET の服を提案することで選択肢の幅が狭まるという課題点については,今回服ではなく服の色を提案する事で解決する.

3.2 おしゃれアドバイスシステムの全体像

事前に画像データから複数の身体的特徴と服の色を取得し,身体的特徴と服の色に関する知識ベースを作成する．システムを使用するユーザーは，自身の身体的特徴の組み合わせと取得した身体的特徴の組み合わせと知識ベースを照らし合わせる.そして自身と同じ身体的特徴の組み合わせデータから服の色を取得し，その服の色をアドバイスとして推薦してもらう．つまり,画像処理によって対象者の身体的特徴を導き出し,その身体的特徴に対する結果(服の色)を知識ベースからアドバイスとして返すシステムである．知識ベースでは似合う色だけでなく,同じ身体的特徴を持つ人があまり着ていない色も結果として導き出せるため個性を出したい人へもアドバイスが出来るようにする．知識ベース作成にあたっては，身体的特徴と服の色のデータはネット上の服を着た人の画像から画像処理のプログラムにより取得する.

3.3 予備調査

3.3.1 身長

はじめに，画像処理から身長を得る方法を調べた．基本的には長さが分かっている物体を基準として対象の物体の長さを測る手法を取る．この測定方法を用いたツールとして「Leafareacounter Plus」というものがある．このアプリは高さ以外に，距離，サイズ，面積なども測ることができる[４].

そして画像処理では無いが,最近では富士通によって基準対象物が写っていなくても推定できる技術が開発された．その技術についての記事にはこう書かれている.「富士通の AI（人工

知能）技術「行動分析技術 Actlyzer」から抽出した骨格情報を利用し，映像の中で常に同じ身長になるよう，カメラの位置や設置角度を幾何計算から求め，これにより，人物が動く映像からカメラ位置と角度が求まり，身長を推定できる[5].」

3.3.2 体重

次に体重について調べた．体重は「男性では体重＝0.660×腹囲（cm）＋0.702×下腿周囲長（cm）＋0.096×年齢（歳）－26.917（R2＝0.862，p＜0.001），女性では体重＝0.315×腹囲（cm）＋0.183×身長（cm）－28.788（R2＝0.836，p＜0.001）」で求められる[6]．画像から周囲長を推定し，体重推定式を使用した要介護認定者の体重を画像から計測するシステムも研究されている[7].

3.4 予備調査結果

身長と体重をおしゃれアドバイスするための要素として加えるため調査を行なった結果，これらを画像処理で取得することはかなり難しいことが分かった．さらに全身が写っていることや，基準対象物が写っていることなど，画像データとして扱うための画像条件が厳しくなってしまうことから，別の要素の検討を行なうことにした．

3.5 本調査

服の色を提案する際に適切な要素として，同じく色に関係する身体的特徴に加えて，服を選ぶ際に考慮する身長や体重などの体格的特徴の取得を考えたが，予備調査では身長や体重といった体格の特徴を画像から取得することは難しいことが分かった．そこで他に適切な要素であり，服選びに影響があると思われる顔立ちや髪を含めた顔の雰囲気を身体的特徴として選んだ．顔の雰囲気に影響を与える要素として，左目と右目の距離，髪色，髪の長さ，肌の色を考えた．髪色と肌の色は色の身体的特徴とし，左目と右目の距離と髪の長さは色以外の身体的特徴とする．本調査では画像処理によって左目と右目の距離，髪色，髪の長さ，肌の色が取得できるかどうかについてそれぞれ調査した．

3.5.1 左目と右目の距離

左目と右目の距離は，色以外の身体的特徴の特徴量として扱え，さらに顔さえ写っていれば取得できることから画像条件も厳しくない．そして OpenCV には目の検出ができるカスケード分類器があったため簡単に取得できることが分かった．

3.5.2 髪色

色に関係する身体的特徴として髪色を選んだ．こちらも同様に，OpenCV に顔検出のカスケード分類器があり，これと目の検出を行なうカスケード分類器を応用することで髪色が取得できると考えた．手法としては，まず顔検出を行ない，顔画像を切り抜く．そして目の上には髪があるため，目の検出で取得した座標の y 座標を上にずらすことで取得できるのではないかと考えた．

3.5.3 髪の長さ

髪色を取得できた場合，髪の長さも特徴量として取得できるのでは無いかと考えた．取得した目の座標から y 座標をだいたい顎近くまで下にずらし，その y 座標の x 軸上に髪色と同じ色が検出された場合，長髪と判断し，取得しなかった場合は短髪と判断できる．この特徴量は色以外の身体的特徴として扱う．

3.5.4 肌の色

髪色の分類分けができるのであれば，肌の色も分類分けできるのでは無いかと思った．肌の色は取得した目の座標から少しずらすだけで取得できるため，髪色の分類分けと同じ手法を用いることで簡単に取得できると考えた．この特徴量は色の身体的特徴として扱う．

3.6　身体的特徴，服の色を取得する流れ

画像データから，それぞれの情報を取得するための流れを図２にフローチャートで表わす．

まず画像を読み込み，それぞれの情報を取得していく．「顔」，「目」，「服」のいずれかが検出できなかった場合はプログラムを終了し，検出できた場合はそのまま処理を続けていく．それぞれの情報を全て取得できた場合は，それらの情報をまとめ，１つのデータとする．これを，データセットを作るための画像すべてに行なう．

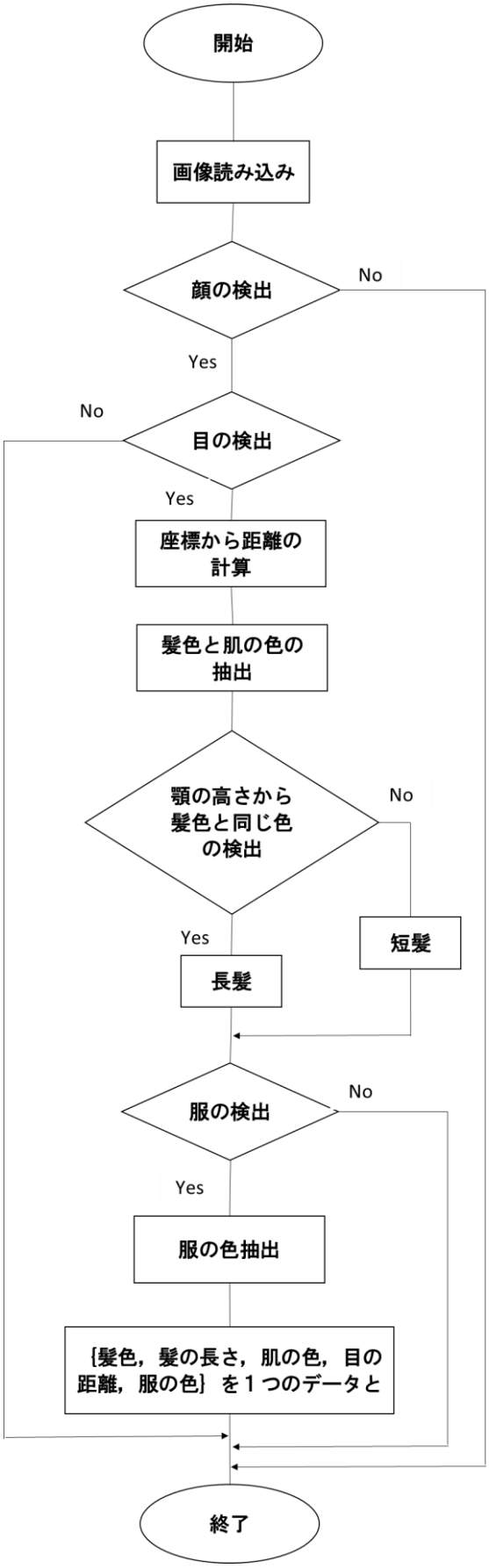

図２．身体的特徴と服の色を取得する流れを示したフローチャート

第4章 実験

4.1 実験目的

ファッション通販サイトから取得した服のサンプルを示すモデル画像から，モデルの身体的特徴と服の色の関係性を調べ，身体的特徴が服の色へ影響を及ぼすのかを明らかにする．そしてシステム作成のために身体的特徴の組み合わせと服の色の割合データを示した知識ベースを作成する．ユーザーと同じ身体的特徴を持つモデルがどんな色の服を着ているのかという情報を得ることさえ出来れば，おしゃれに自信の無い人へのアドバイスになり，個性を出したい人へのアドバイスにもなるため，その結果に基づいたおしゃれアドバイスシステムを作ることができる．身体的特徴は，特徴量として「髪色」13 種，「肌の色」12 種，「髪の長さ」2 種，「目と目の距離」3 種の 4 つで，服の色として 14 種のデータを画像から取得する．

4.2 環境・準備

使用した PC は，iiyama PC の Intel Corei7，NVIDIA GEFORCE RTX 内蔵型である．

開発環境は，OS が Linux（バージョン 5.15.0-84-generic）で Ubuntu 22.04.3 LTS を使用した．ツールは Visual Studio Code（バージョン 1.82.2）を使用し，言語は Python のみで行なった．Python のバージョンは 3.10.12 を使用する．画像処理を行なう際には，OpenCV を使用した．

学習データとする画像は約 500 枚用意した．この 500 枚の画像データはファッション雑誌の画像から取得する予定だったが，500 枚集めることが難しかったためショッピングサイトのモデル画像で代用した．画像は，しっかりと顔が写っていること，メガネや帽子は付けていないこと，できるだけ正面を向いていることを条件として収集していった．

4.3 特徴抽出のアルゴリズム

画像データセットから身体的特徴と服の色を取得するためのプログラムは図 2 で示した通りに動かしていく．そのプログラムについて，図 2 のフローチャートを基に説明していく．プログラム全体は付録で示す．

4.3.1 画像読み込みと顔の検出

画像データセットとして扱うための条件が厳しかったこともあり，今回使用した画像は全て，条件に合う画像が多く載っていた 1 つのサイトから取得した．そのため，全て同じサイズで用意することができた．複数のサイトから取ってくる場合は同じ画像サイズで取得する．ただし目の距離の結果が大幅に変わらないように，リサイズして取得する際に縦と横の比を崩さ

ないようにする必要がある．取得した画像は一つのファイルにまとめ，glob.glob 関数を使用してパスを取得し，for ループを使用してファイル内の画像を 1 枚ずつ imread で読み込んでいく．この画像読み込みの処理を図 3 に示す．図 3 にある count は，今何枚目を読み込んでいるのかを数えている．

```
#パスを取得して拡張子がjpgのものだけ取ってくる
file_list = glob.glob(os.path.join("./output/*.jpg"))
for path in file_list:
    count += 1
    print(count)
# 画像の読み込み
    img = cv2.imread(path)
    print(path)
# 画像のグレースケール化
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# 学習済みモデルの読み込み
    cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
```

図 3．画像の読み込みとカスケード分類器の取得プログラム

読み込んだ画像を，OpenCV ダウンロード時に付いてくる顔検知のカスケード分類器で顔の検出を行なう．まず画像をグレースケール化して処理を軽くする．枚数が多い場合は多くの処理が必要となるため，処理速度を速めるためにも情報量の少ないグレースケール画像に変換する．そして CascadeClassifier クラスでカスケード分類器を用意する．学習済みモデルは「haarcascade_frontalface_default.xml」を使用して顔検出を行なった．

顔検出を行なう際，精度を高い数値から徐々に低くして行なっていく．図 4 に顔検知のプログラムの一部を示す．カスケード分類器を用いた検出は，cascade.detectMultiScale という関数を使用して行なう．この関数にはいくつかのパラメータが設定されており，その中の minNeighbors というパラメータが検出の精度を表わす．このパラメータはデフォルトの状態では 3 が設定されており，高くするほど誤検知が減り精度が上がるが，高すぎると顔が検出されない場合があり，逆に低すぎると顔ではないところまでもが顔と認識されてしまう．そこでまずは高い精度で検出を行ない，検出できなかった場合は 1 つ低い精度で行なうという手順を繰り返していった．今回は minNeighbors の値を 5 から 3 の範囲で行った．いずれかのパラメータで顔検出が出来た場合はその後の処理を行なうが，どうしても出来なかった場合は「検出できませんでした」と表示し，break でプログラムを終了させた．検出できた場合は顔部分を切り抜き，幅 640，高さ 480 でリサイズして face_cut.jpg として保存し，目の検出に進む．

リサイズする理由としては,顔を抜き出した際に画像サイズが小さすぎると目の検出がうまくいかないことや，座標から色や目の距離を抽出するため，サイズの統一をしていない場合，抽出する部位にズレが生じる可能性があるからである．図 3，図 4 のプログラムは Web サイトのプログラムを参考にしている[８]．

```
# 顔を検出する
    lists = cascade.detectMultiScale(img_gray, minNeighbors = 5, minSize=(100, 100)) #min
    if len(lists):
        # 顔を検出した場合
        for (x,y,w,h) in lists:
            #顔部分を切り取る
            face_cut = img[y:y+h, x:x+w]
         #調整後サイズを指定(横幅、縦高さ)
        dst = cv2.resize(face_cut, (640, 480))
        cv2.imwrite('face_cut.jpg',dst)
    else:
        lists = cascade.detectMultiScale(img_gray, minNeighbors = 4, minSize=(100, 100))
```

図 4．顔検出のプログラム（一部）

4.3.2 目の検出と，左目と右目の距離の計算

目の検出もカスケード分類器を使用して行なう．使用する学習済みモデルは「haarcascade_eye.xml」．画像はこの前に作成した face_cut.jpg を使用する．アルゴリズムは顔検出の時とほとんど同じであり，minNeighbors を高く設定してから徐々に低くして検出を行なっていった．minNeighbors の値は，70 から 10 ずつ減らしていき，minNeighbors が 10 目 10 になるまで行なう．ただし目の検出は左目と右目で 2 つ必要であるが，どれだけ minNeighbors の値を高くしても 3 つ以上検出される画像がいくつかあり，３つ以上検出される画像はだいたい口の部分を目と誤検知している傾向にあった．そこで，検出した順番に x 座標と y 座標のデータを配列に格納していき，1 番目と 2 番目に格納した座標データを左目と右目の座標として扱った．格納した時点では 1 番目と 2 番目に格納した座標データはどちらが左目でどちらが右目の座標なのかが不明であるため，それぞれの x 座標の値の大きさで判断する（大きい方が右目で小さい方が左目）．目の検出ができなかった，もしくは目が一つしか検出できなかった場合は「検出できませんでした」と表示し，break でプログラムを終了させた．

うまく検出できた場合,格納した 2 つのデータ(左目と右目の座標)から目の距離を計算し，目の距離を「目の距離「大」」,「目の距離「中」,「目の距離「小」」の 3 つに分類分けした．計算式は$\sqrt{(a[x]-b[x])^2+(a[y]-b[y])^2}$で行なった．a と b は左目と右目の座標を表わす（例：

a[x]はａのｘ座標を表わす)．距離が 240 より大きい場合は「目の距離「大」」，200 以上 240 以下の場合は「目の距離「中」」，200 より小さい場合は「目の距離「小」」と判定した．

4.3.3 髪色と肌の色の抽出

色の判定はあるサイトのプログラムから参考にした[９]．図 5 はそのサイトを参考にして作成したプログラムである．図 5 を基に上から説明していく．

color_judge 関数は色を算出して戻り値として返す関数である．HSV 色空間の H（色相）を扱い，画像の特定の座標から抽出した H の値を引数 value として関数に渡す．OpenCV では、H の範囲は 0~180 となっている．今回この関数で検出する色の種類は 12 色とするため，color_wheel 配列の数もそれに合わせて変化させている．そして引数として渡した H の値が色 1 から色 12 のどの色に属するかを計算で導き出している．髪色と肌の色は対象者に結果として渡すわけではなく，分類分けでのみ使用するため色の名前は番号で付けている．

色の抽出には顔の検出で作成した face_cut.jpg を使用する．図 5 では file_name に face_cut.jpg が代入されている．まず imread で読み込んだ後に，COLOR_BGR2HSV で BGR から HSV に色空間の変換を行なう.これは HSV 色空間の方が色を表現しやすいからである.その後，髪色と肌の色それぞれの座標から HSV それぞれの値を取得する．img_HSV と img_skinHSV にはそれぞれ[H の値，S の値，V の値]の配列が代入されている．髪色は左目の座標から x と y 座標をそれぞれ 100 ずつ引いた座標（だいたい右こめかみあたりで，x 座標=search_posx，y 座標=search_posy）から HSV の値を取得し，肌の色は左目と右目の間の座標（だいたい鼻の上あたりで，x 座標=skin_posx，y 座標=skin_posy）から取得した．肌の色は color_judge 関数をそのまま使用して 12 色に分類分けするが，黒色が多い髪色は色 1 から色 12 とは別に，色 0 を設定する．黒という色は色相で表わすことができず，HSV の V（明度）で表わすことが出来る．OpenCV では，明度は 0~255 の範囲で表わされ値が低いほど黒に近くなる．今回は V が 90 以下の値は全て色 0（黒色）と判定することにした．V が 90 以上と判定されたものは color_judge 関数を使用して髪色を判定している（髪は光を反射し，光の当たり具合によっては色が左右する可能性があるため範囲を広めに設定した)．ただし肌の色と同じ色が髪色として検出された場合，肌部分の座標から髪色を抽出している可能性があるため，x 座標を-10 ずつずらしていき，肌の色と異なる色が検出された部分で髪色の抽出を行なう．

```
#色を算出する関数
def color_judge(value):
    color_wheel=['色1','色2','色3','色4','色5','色6',
                 '色7','色8','色9','色10','色11','色12',
                 '色1']
    index_value=int(round(value/(180/(len(color_wheel)-1)),0))
    return color_wheel[index_value]


#選択した場所の色を出力する
img=cv2.imread(file_name,cv2.IMREAD_COLOR)
imghsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
img_array = np.asarray(imghsv)
img_HSV = img_array[search_posy,search_posx,:]  #髪色
img_skinHSV = img_array[skin_posy,skin_posx,:]  #肌の色
skin_color = color_judge(imghsv[skin_posy,skin_posx,0])
if 90 >= img_HSV[2]:
    color = '色0'  #黒を表す
else:
    color = color_judge(imghsv[search_posy,search_posx,0])
    while skin_color == color:  #髪色と肌の色が同じである間
        search_posx = search_posx - 10
        img_HSV = img_array[search_posy,search_posx,:]  #髪色の値更新
        if 90 >= img_HSV[2]:
            color = '色0'  #黒を表す
        else:
            color = color_judge(imghsv[search_posy,search_posx,0])
```

図 5．髪色と肌の色を抽出するプログラム

4.3.4 髪の長さ判定

まず face_cut.jpg 画像において，x 座標が 0，y 座標が 440（画像の左端下あたり）から，x 座標が 140 になるまで x 座標を 10 ずつ増やしてずらしていく．y 座標 440 はだいたい顎や首あたりの高さを示す．その間に髪色と同じ色が検出された場合，「長髪」と判定し，検出されなかった場合は「短髪」と判定する．右端（x 座標が 640）まで検出を行なわない理由は，首当たりで影になっているところが黒と検出され，黒髪で短髪の人が長髪と判定されてしまうことがあったためである．そこで，おおよそ首まで行かない座標の 140 までで判定を行なうことにした．

4.3.5 服の検出

服の検出は顔検出の時と同じ学習済みモデルを使用して行なう.顔の下には必ず上半身があるため，元画像で顔の検出を行なった後，顔の下部分を切り抜いて服の検出をする．顔部分の面積より服部分の面積の方が大きいため,顔の切り抜きとは異なった大きさで切り抜きを行なった．図 6 に服部分の切り取りプログラムを示す．minNeighbors の値は顔検出の時と同じく 5 から 3 まで行なっている．顔がうまく検出された場合，図 6 の 6 行目のように，切り取る場所やサイズを調整して clothes_cut に代入し，服部分を取得している．図 4 の 7 行目のプログラムと比べてみると調整していることが分かると思う.そして幅 244,高さ 235 でリサイズし，clothes_cut.jpg として保存している．

```
# 顔を検出する
lists = cascade.detectMultiScale(img_gray, minNeighbors = 5, minSize=(100, 100))
if len(lists):
    for (x,y,w,h) in lists:
        #服部分を切り取る
        clothes_cut = img[y-int(h*-1.4):y+int(h*3.7), x-int(w*0.6):x+int(w*1.8)] #img
    dst = cv2.resize(clothes_cut, (244, 235))
    cv2.imwrite('clothes_cut.jpg', dst)
    #cv2.imshow('clothes_cut',dst)
    #cv2.waitKey(0)
else:
    lists = cascade.detectMultiScale(img_gray, minNeighbors = 4, minSize=(100, 100))
    if len(lists):
```

図 6．服の切り取り

4.3.6 服の色の抽出

服の色の抽出には髪色の抽出と同じく，図 5 にある color_judge 関数を使用する．ただし服の色の抽出では番号で分類せず，色の名前を付けた．そのため新しく color_judge 関数を作成し，色はそれぞれ{赤，赤橙，オレンジ，黄橙，黄色，黄緑，緑，青緑，青，青紫，紫，赤紫}の 12 色で名付けた．この色の名前は color_judge 関数を作成する際に参考にしたサイトから引用している[9]．服の色は color_judge 関数で判定する 12 色に，黒色と白色を加えた 14 色で分けることにした．色相だけでは黒色と白色は表せられないため，HSV の範囲を自分で設定し，判定することにした．図 7 に服の色抽出プログラムの一部を示す．

しかし服の色は柄や模様があるため，1 つの座標から服の色を判定することはあまり良くないと考えた．そこで，縦 10 横 10 で合計 100 個の座標から色を抽出した．色抽出は x 座標 50，y 座標 60 の地点からはじめ，x 座標を+10 ずつ動かしていき，x 座標が 150 になるまで色を抽出していく．x 座標 150 まで抽出すると，次は y 座標を+10 動かし，また x 座標 50 から 150 ま

で抽出を行なう．これを y 座標 160 になるまで続け，合計 100 個の色を抽出する．抽出した色は配列に格納していき，その抽出した 100 個の色の内 20 個以上同じ色が判定された場合は服の色として認め，そうで無い場合は服の色として認めず省くことにした．

```
search_posy = 60
for num in range(10):
    search_posx = 50
    for num in range(10):
        color=color_judge(imghsv[search_posy,search_posx,0])
        img_array = np.asarray(imghsv)
        img_HSV = img_array[search_posy,search_posx,:]
        cv2.circle(img,center=(search_posx,search_posy),radius=10,color=(255,255,255),thickness=3)
        img_HSV = img_array[search_posy,search_posx,:]
        hsv_min = np.array([0,0,0])
        hsv_max = np.array([180,255,30])
        if hsv_min[0] <= img_HSV[0] <= hsv_max[0] and hsv_min[1] <= img_HSV[1] <= hsv_max[1] and hsv_min[2] <= img_HSV[2] <= hsv_max[2]:
            color = '黒色'
        hsv_min = np.array([0,0,200])
        hsv_max = np.array([180,20,255])
        if hsv_min[0] <= img_HSV[0] <= hsv_max[0] and hsv_min[1] <= img_HSV[1] <= hsv_max[1] and hsv_min[2] <= img_HSV[2] <= hsv_max[2]:
            color = '白色'
        color_list.append(color)
        search_posx += 10
    search_posy += 10
```

図 7．服の色抽出プログラム（一部）

4.4　結果

身体的特徴毎のそれぞれ取得できた色の数の表を表 1 と表 2 に分けて示す．背景の緑が濃いほど取得できた割合が大きいことを示し，赤が濃いほど取得できた割合が小さいことを示す．

集めた 500 枚の画像の内，結果全てのデータが綺麗に取れた画像は 380 枚だった．それぞれの身体的特徴の内，どれか一つでも取得できなかった画像はデータとして含めないことにした．髪色や肌の色は 12 色以上あるが，ここでは 10 個以上データが取得できたもののみをピックアップしてまとめている．合計してもデータ数が 380 にならないのはそのためである．

表 1．身体的特徴毎のそれぞれ取得できた色の数（1）

	赤	赤橙	オレンジ	黄橙	黄緑	黄色	緑
髪色0	36	59	25	30	5	3	22
髪色1	8	14	5	8	2	1	4
髪色2	3	1	2	2	1	0	1
髪色3	6	8	6	3	2	1	0
短髪	23	29	17	18	6	3	13
長髪	33	56	23	28	5	2	16
肌の色1	5	1	3	3	1	0	1
肌の色2	50	83	37	42	10	5	27
目の距離「小」	11	20	11	10	4	0	9
目の距離「中」	21	42	20	24	4	4	13
目の距離「大」	24	23	9	12	3	1	7

表 2．身体的特徴毎のそれぞれ取得できた色の数（2）

	青緑	青	青紫	紫	赤紫	黒	白	データ数
髪色0	40	52	14	2	8	26	64	259
髪色1	17	10	2	1	0	2	10	51
髪色2	5	5	1	0	0	0	3	19
髪色3	7	7	0	2	2	1	4	29
短髪	36	41	4	1	4	11	44	157
長髪	39	38	15	4	6	20	46	223
肌の色1	9	10	5	0	0	0	5	33
肌の色2	65	69	14	5	10	31	81	344
目の距離「小」	15	24	1	2	1	4	21	90
目の距離「中」	36	31	10	2	6	19	34	171
目の距離「大」	24	24	8	1	3	8	32	119

身体的特徴の組み合わせと服の色との割合データ（知識ベース）の例を表 3 に示す．全ての組み合わせで作成するとかなりの量となってしまうため，表 1 と表 2 の結果から比較的多く取れたデータを選び，その組み合わせを例として 4 通り作成した．3.2 で考えたシステムの全体像において，ここでは取得できたデータの多い量と少ない量をそれぞれ 3 色ずつ推薦するシステムとして考える．そこで，取得した割合データの内，上位 3 色を緑で表わし，下位 3 色を赤で表わした．同じ割合の色がある場合は 4 色以上示されているものもある．結果，「髪色 0，長髪，肌の色 2，目の距離「中」」のユーザーには，赤橙 22%，白 19%，黄橙 11%，青 11%を順に似合っている色として推薦し，個性を出したい人へは紫 0%，黄緑 1%，黄色 1%をアドバイスとして推薦する．「髪色 1，長髪，肌の色 2，目の距離「中」」の人には青緑 25%と赤橙 13%，オレンジ 13%と黒 13%を推薦し，黄緑 0%，黄色 0%，紫 0%，赤紫 0%を，個性を出したい人へ推薦する．「髪色 0，長髪，肌の色 2，目の距離「小」」の人には赤橙 19%，オレンジ 13%，青 12%を推薦し，個性を出したい人には黄色 0%，青紫 0%，赤紫 0%を推薦する．「髪色 1，長髪，肌の色 2，目の距離「小」」の人には青緑 31%，青 19%，白 19%を推薦し，個性を出したい人へはオレンジ 0%，黄緑 0%，黄色 0%，緑 0%，青紫 0%，赤紫 0%，黒 0%を推薦する．ユーザーには色に合わせて割合の数値もアドバイスとして出す．

表 3．身体的特徴の組み合わせと服の色との割合データの例

	赤	赤橙	オレンジ	黄橙	黄緑	黄色	緑	青緑	青	青紫	紫	赤紫	黒	白
髪色0, 長髪, 肌の色2, 目の距離「中」	7%	22%	6%	11%	1%	1%	4%	8%	11%	4%	0%	2%	10%	19%
髪色1, 長髪, 肌の色2, 目の距離「中」	6%	13%	13%	6%	0%	0%	6%	25%	6%	6%	0%	0%	13%	6%
髪色0, 長髪, 肌の色2, 目の距離「小」	10%	19%	14%	10%	2%	0%	7%	7%	12%	0%	2%	0%	7%	10%
髪色1, 長髪, 肌の色2, 目の距離「小」	6%	13%	0%	6%	0%	0%	0%	31%	19%	0%	6%	0%	0%	19%

4.5　考察

表 1 と表 2 から，身体的特徴毎の服の色への影響は類似した関係性が見られた．表 1，2 の

結果を基に表 3 の通り知識ベースの作成に成功した．表 3 を見てみると，黄色や赤紫などの割合が 0 に近いことなど多少似通った部分はあるものの，身体的特徴の組み合わせ毎に異なる結果となっていることが分かり，良いアドバイス結果となっている．しかし長髪や肌の色 2 など，取得できた身体的特徴の偏りが激しく，ほとんどの組み合わせで十分なデータ量を取得できなかったため，改善すべき点はある．

偏りが激しかった理由としては，今回集めた画像データは条件が厳しかったこともあり全て一つのサイトから集めたものであるため，そのブランドの傾向が反映されているという可能性が考えられる．そのため，他のサイトからも画像を集めて改善を図っていくこと，そしてより多くのデータを取得することを今後の課題にしていきたい．

今回は身長や体重といった頭部以外の身体的特徴の画像処理による取得難易度が高く断念することになったが，身体的特徴が服の色に影響を及ぼすのかどうかを明確にするには，今後は顔部分に関する身体的特徴以外についても服の色との関係性を研究していく必要があるだろう．身体的特徴による服の色への影響が見られた場合は，その身体的特徴を組み込んだ上で知識ベースを作成し，よりアドバイスにばらつきを加えてシステムの改善を図りたい．

第5章 まとめ

近年ではおしゃれが重視される傾向にあることから, おしゃれに自信の無い人などへおしゃれアドバイスが必要であると感じ, 本研究ではユーザーの身体的特徴から服の色を提案するおしゃれアドバイスシステムについて考えた.

既存技術では服を提案するものや, アンケートによって取得した色に関する身体的特徴からユーザーに合った色を提案する技術があったが, これらの技術には服を提案することでそのユーザーの選択肢の幅を狭めてしまう問題点や, 服の色では無く全体的に合った色であるため服の色の提案には相応しくないという問題点が考えられた. さらにほとんど色以外の身体的特徴を扱っていない点や, アンケート形式であることによる情報の正確性に疑問が残った. そこで, おしゃれアドバイスシステムでは画像処理によって身体的特徴を取得することで情報の正確性を上げ, 色以外の身体的特徴による服の色への影響についても調査することにした. 選択肢の幅に関する問題点は服の色を提案することで解決することにした. 予備調査では色以外の身体的特徴である身長や体重が画像処理で取得しづらいことが判明したため, 服選びに影響を与えると考えた顔の身体的特徴を要素とすることにした. 本調査では「目の距離」,「髪色」,「髪の長さ」,「肌の色」が画像処理で取得できることが分かった.

実験では 1 つのファッション通販サイトから 500 枚の服を着た人の画像を用意し, それぞれ本調査で取得できると判明した 4 つの顔の身体的特徴と服の色を画像処理で取得した. そして顔の身体的特徴と服の色の関係性に基づく知識ベースを作成した. 知識ベースを用いることで, 個別の身体的特徴毎に異なる服の色をアドバイスできた. しかし知識ベースを見てみると, 取得できた身体的特徴に激しい偏りが見られた. これは 1 つのファッション通販サイトから画像を用意したことで, その通販サイトのブランドの傾向が反映されてしまっていることなどが原因として考えられた. そのため今後は他のサイトからも画像を収集して実験を行ない比較すること, そしてその他の身体的特徴による服の色への影響について調べていくことなどが必要であると考えた.

謝辞

本研究では壁にぶつかることが多く，研究が思ったように進まないことがありましたが，三好 力教授や同研究室のご先輩を含めたメンバーの皆様には多様な場面で多くのアドバイスをご教授いただきましたため，深く感謝いたしております.

参考文献

[1]：データからみえる今日の世相　～今どきの「おしゃれ重視派」～　-調査情報デジタル
https://tbs-mri.com/n/nf75d05c0ca2c
[2]：パーソナルカラー実務検定協会 パーソナルカラーとは
https://acb-color.com/about_personalcolor/
[3]：SENSY CLOSET
https://closet.sensy.ai/
[4]：写真や画像から、長さ・角度・面積を測定できるフリーソフト
https://constupper.com/gsokute-kensoft-247/
[5]：富士通、防犯カメラ画像から位置や角度を算出して対象人物の身長を推定する技術
https://it.impress.co.jp/articles/-/22221
[6]：大西 玲子, 藤井 弘二, 津田 博子, 今井 克己, 寝たきり要介護高齢者における体重推定式の作成, 日本老年医学会雑誌, 49 巻, pp.746~751(2012).
https://www.jstage.jst.go.jp/article/geriatrics/49/6/49_746/_pdf/-char/ja
[7]：画像を用いた簡便な体重推定を行うシステム
https://web.wakayama-u.ac.jp/~yoshino/lab/research/kiwa_tanaka_2020/
[8]：OpenCV による顔認識 画像から顔を検出する方法
https://office54.net/python/opencv/image-face-authentication
[9]：【python-openCV】カラー画像における指定位置の色を判定する方法！-ヒザカラ
https://www.higashisalary.com/entry/cv2-color-judge

付録（プログラム全体）

```
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt #画像を表示するためのモジュール
import cv2
import numpy as np
import os
import glob
import sys
import pandas as pd
import openpyxl
from openpyxl.styles.borders import Border, Side
from openpyxl.styles.fonts import Font
from openpyxl.styles.alignment import Alignment
from PIL import Image
import math
title_txt = '身体的特徴（髪の色と目と目の距離）と服の色の表'
data = [
    ['髪の色','髪の長さ','肌の色', '目と目の距離', '服の色']
]
try:
    # ファイルを開く
    wbook = openpyxl.load_workbook('./osyare.xlsx')
except FileNotFoundError as e:
    print(e)
    exit()
# ワークシートは集計
sheet = wbook['集計']
count = 0                #何枚目を読み込んでいるかの確認
#パスを取得して拡張子が jpg のものだけ取ってくる
file_list = glob.glob(os.path.join("./output/*.jpg"))
for path in file_list:
    count += 1
    print(count)
# 画像の読み込み
    img = cv2.imread(path)
    print(path)
# 画像のグレースケール化
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# 学習済みモデルの読み込み
    cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
# 顔を検出する
    lists = cascade.detectMultiScale(img_gray, minNeighbors = 5, minSize=(100, 100)) #minNeighbors を３から５にすることで検出精度を上げるが、未検出も増える可能性がある
    if len(lists):
        # 顔を検出した場合
        for (x,y,w,h) in lists:
            #顔部分を切り取る
            face_cut = img[y:y+h, x:x+w]
         #調整後サイズを指定(横幅、縦高さ)
        dst = cv2.resize(face_cut, (640, 480))
        cv2.imwrite('face_cut.jpg',dst)
    else:
        lists = cascade.detectMultiScale(img_gray, minNeighbors = 4, minSize=(100, 100)) #minNeighbors を３から５にすることで検出精度を上げるが、未検出も増える可能性がある
        if len(lists):
            # 顔を検出した場合
            for (x,y,w,h) in lists:
                #顔部分を切り取る
                face_cut = img[y:y+h,x:x+w]
            #調整後サイズを指定(横幅、縦高さ)
            dst = cv2.resize(face_cut, (640, 480))
            cv2.imwrite('face_cut.jpg',dst)
        else:
            lists = cascade.detectMultiScale(img_gray, minNeighbors = 3, minSize=(100, 100)) #minNeighbors を３から５にすることで検出精度を上げるが、未検出も増える可能性がある
```

```
        if len(lists):
            # 顔を検出した場合
            for (x,y,w,h) in lists:
                face_cut = img[y:y+h,x:x+w]
            dst = cv2.resize(face_cut, (640, 480))
            cv2.imwrite('face_cut.jpg',dst)
        else:
            print('顔が見つかりませんでした。')
            break
#情報を入力する
file_name='face_cut.jpg'
#ここから目の検出を行う
eye_cascade = cv2.CascadeClassifier("haarcascade_eye.xml")                    #目検出のカスケード分類機を使用
eye_list = []
l = []
face_cut_gray = cv2.cvtColor(dst, cv2.COLOR_BGR2GRAY)
eyes = eye_cascade.detectMultiScale(face_cut_gray, minNeighbors = 70)
if len(eyes) >= 2:
    for (x,y,w,h) in eyes:
        #
        cv2.circle(dst, (x+w//2, y+h//2), 5, (0,0,255), thickness=-1)
        l.append(x+w//2)
        l.append(y+h//2)
    a = {'x':l[0], 'y':l[1]}
    b = {'x':l[2], 'y':l[3]}
    dist = math.sqrt((a['x'] - b['x']) ** 2 + (a['y'] - b['y']) ** 2)
    #print(dist)
    cv2.imwrite('face_cut.jpg',dst)
    skin_posx = int((a['x'] + b['x']) / 2)
    skin_posy = int((a['y'] + b['y']) / 2)
    #左目の座標を取得する
    if a['x'] > b['x']:
        search_posx = b['x']-100            #髪の色を取得する座標をここで決める
        search_posy = b['y']-100
    else:
        search_posx = a['x']-100            #髪の色を取得する座標をここで決める
        search_posy = a['y']-100
else:
    eyes = eye_cascade.detectMultiScale(face_cut_gray, minNeighbors = 60)
    if len(eyes) >= 2:
        for (x,y,w,h) in eyes:
            cv2.circle(dst, (x+w//2, y+h//2), 5, (0,0,255), thickness=-1)
            l.append(x+w//2)
            l.append(y+h//2)

        a = {'x':l[0], 'y':l[1]}
        b = {'x':l[2], 'y':l[3]}
        dist = math.sqrt((a['x'] - b['x']) ** 2 + (a['y'] - b['y']) ** 2)
        cv2.imwrite('face_cut.jpg',dst)
        skin_posx = int((a['x'] + b['x']) / 2)
        skin_posy = int((a['y'] + b['y']) / 2)
        if a['x'] > b['x']:
            search_posx = b['x']-100            #髪の色を取得する座標をここで決める
            search_posy = b['y']-100
        else:
            search_posx = a['x']-100            #髪の色を取得する座標をここで決める
            search_posy = a['y']-100
    else:
        eyes = eye_cascade.detectMultiScale(face_cut_gray, minNeighbors = 50)
        if len(eyes) >= 2:
            for (x,y,w,h) in eyes:
                cv2.circle(dst, (x+w//2, y+h//2), 5, (0,0,255), thickness=-1)
                l.append(x+w//2)
                l.append(y+h//2)
```

```
        a = {'x':l[0], 'y':l[1]}
        b = {'x':l[2], 'y':l[3]}
        dist = math.sqrt((a['x'] - b['x']) ** 2 + (a['y'] - b['y']) ** 2)
        cv2.imwrite('face_cut.jpg',dst)
        skin_posx = int((a['x'] + b['x']) / 2)
        skin_posy = int((a['y'] + b['y']) / 2)
        if a['x'] > b['x']:
            search_posx = b['x']-100            #髪の色を取得する座標をここで決める
            search_posy = b['y']-100
        else:
            search_posx = a['x']-100            #髪の色を取得する座標をここで決める
            search_posy = a['y']-100
    else:
        eyes = eye_cascade.detectMultiScale(face_cut_gray, minNeighbors = 40)
        if len(eyes) >= 2:
            for (x,y,w,h) in eyes:
                cv2.circle(dst, (x+w//2, y+h//2), 5, (0,0,255), thickness=-1)
                l.append(x+w//2)
                l.append(y+h//2)
            a = {'x':l[0], 'y':l[1]}
            b = {'x':l[2], 'y':l[3]}
            dist = math.sqrt((a['x'] - b['x']) ** 2 + (a['y'] - b['y']) ** 2)
            cv2.imwrite('face_cut.jpg',dst)
            skin_posx = int((a['x'] + b['x']) / 2)
            skin_posy = int((a['y'] + b['y']) / 2)
            if a['x'] > b['x']:
                search_posx = b['x']-100            #髪の色を取得する座標をここで決める
                search_posy = b['y']-100
            else:
                search_posx = a['x']-100            #髪の色を取得する座標をここで決める
                search_posy = a['y']-100
        else:
            eyes = eye_cascade.detectMultiScale(face_cut_gray, minNeighbors = 30)
            if len(eyes) >= 2:
                for (x,y,w,h) in eyes:
                    cv2.circle(dst, (x+w//2, y+h//2), 5, (0,0,255), thickness=-1)
                    l.append(x+w//2)
                    l.append(y+h//2)
                a = {'x':l[0], 'y':l[1]}
                b = {'x':l[2], 'y':l[3]}
                dist = math.sqrt((a['x'] - b['x']) ** 2 + (a['y'] - b['y']) ** 2)
                cv2.imwrite('face_cut.jpg',dst)
                skin_posx = int((a['x'] + b['x']) / 2)
                skin_posy = int((a['y'] + b['y']) / 2)
                if a['x'] > b['x']:
                    search_posx = b['x']-100            #髪の色を取得する座標をここで決める
                    search_posy = b['y']-100
                else:
                    search_posx = a['x']-100            #髪の色を取得する座標をここで決める
                    search_posy = a['y']-100
            else:
                eyes = eye_cascade.detectMultiScale(face_cut_gray, minNeighbors = 20)
                if len(eyes) >= 2:
                    for (x,y,w,h) in eyes:
                        cv2.circle(dst, (x+w//2, y+h//2), 5, (0,0,255), thickness=-1)
                        l.append(x+w//2)
                        l.append(y+h//2)
                    a = {'x':l[0], 'y':l[1]}
                    b = {'x':l[2], 'y':l[3]}
                    dist = math.sqrt((a['x'] - b['x']) ** 2 + (a['y'] - b['y']) ** 2)
                    cv2.imwrite('face_cut.jpg',dst)
                    skin_posx = int((a['x'] + b['x']) / 2)
                    skin_posy = int((a['y'] + b['y']) / 2)
```

```
                    if a['x'] > b['x']:
                        search_posx = b['x']-100        #髪の色を取得する座標をここで決める
                        search_posy = b['y']-100
                    else:
                        search_posx = a['x']-100        #髪の色を取得する座標をここで決める
                        search_posy = a['y']-100
                else:
                    eyes = eye_cascade.detectMultiScale(face_cut_gray, minNeighbors = 10)
                    if len(eyes) >= 2:
                        for (x,y,w,h) in eyes:
                            cv2.circle(dst, (x+w//2, y+h//2), 5, (0,0,255), thickness=-1)
                            l.append(x+w//2)
                            l.append(y+h//2)

                        a = {'x':l[0], 'y':l[1]}
                        b = {'x':l[2], 'y':l[3]}
                        dist = math.sqrt((a['x'] - b['x']) ** 2 + (a['y'] - b['y']) ** 2)
                        cv2.imwrite('face_cut.jpg',dst)
                        skin_posx = int((a['x'] + b['x']) / 2)
                        skin_posy = int((a['y'] + b['y']) / 2)
                        if a['x'] > b['x']:
                            search_posx = b['x']-100        #髪の色を取得する座標をここで決める
                            search_posy = b['y']-100
                        else:
                            search_posx = a['x']-100        #髪の色を取得する座標をここで決める
                            search_posy = a['y']-100
                    else:
                        print('目が見つかりませんでした')
                        break
if 0 >= search_posx:
    search_posx = 0
if not dist:
    print('目が見つからないので判断できません')
    break
else:
    if dist == 0:
        eyes_lt = '目の距離測定不能'
    else:
        if dist > 240:
            eyes_lt = '目の距離「大」'
            print(eyes_lt)
        else :
            if dist <= 240 and dist >= 200:
                eyes_lt = '目の距離「中」'
                print(eyes_lt)
            else:
                if dist < 200:
                    eyes_lt = '目の距離「小」'
                    print(eyes_lt)
#色を算出する関数
def color_judge(value):
    color_wheel=['色 1','色 2','色 3','色 4','色 5','色 6',
                 '色 7','色 8','色 9','色 10','色 11','色 12',
                 '色 1']
    index_value=int(round(value/(180/(len(color_wheel)-1)),0))
    return color_wheel[index_value]
#選択した場所の色を出力する
img=cv2.imread(file_name,cv2.IMREAD_COLOR)
imghsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
img_array = np.asarray(imghsv)
img_HSV = img_array[search_posy,search_posx,:]  #髪色
img_skinHSV = img_array[skin_posy,skin_posx,:]  #肌の色
skin_color = color_judge(imghsv[skin_posy,skin_posx,0])
if 90 >= img_HSV[2]:
```

```
        color = '色 0'    #黒を表す
    else:
        color = color_judge(imghsv[search_posy,search_posx,0])
        while skin_color == color:    #髪色と肌の色が同じである間
            search_posx = search_posx - 10
            img_HSV = img_array[search_posy,search_posx,:]    #髪色の値更新
            if 90 >= img_HSV[2]:
                color = '色 0'    #黒を表す
            else:
                color = color_judge(imghsv[search_posy,search_posx,0])
    #髪の長さを測る
    hair_posx = 0
    hair_posy = 440
    img_hairHSV = img_array[hair_posy,hair_posx,:]    #だいたい y 軸は顎あたり x 軸は一番右から
    if 90 >= img_hairHSV[2]:
        hair_search = '色 0'
    else:
        hair_search = color_judge(imghsv[hair_posy,hair_posx,0])
    while 140 > hair_posx and hair_search != color:
        hair_posx = hair_posx + 10
        img_hairHSV = img_array[hair_posy,hair_posx,:]    #だいたい y 軸は顎あたり x 軸は一番右から
        if 90 >= img_hairHSV[2]:
            hair_search = '色 0'
        else:
            hair_search = color_judge(imghsv[hair_posy,hair_posx,0])
    if hair_search == color:
        hair_length = '長髪'
    else:
        hair_length = '短髪'
    #cv2.imwrite('img_point_'+str(search_posx)+'_'+str(search_posy)+'.jpg',img)
    print("髪",color)
    #print(img_HSV[0],img_HSV[1],img_HSV[2])
    print("肌の",skin_color)
    #print(img_skinHSV[0],img_skinHSV[1],img_skinHSV[2])
    print(hair_length)
    hair_cl = color
    hair_lt = hair_length
    skin_cl = skin_color
    #選択した場所の色を確認するための画像を出力
    #heck=np.zeros((256,256,3),np.uint8)
    #ここから服の検出を行う
    #  画像の読み込み
    img = cv2.imread(path)
    #w,h,c = img.shape
    #  画像のグレースケール化
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #  学習済みモデルの読み込み
    cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
    #  顔を検出する
    lists = cascade.detectMultiScale(img_gray, minNeighbors = 5, minSize=(100, 100))
    if len(lists):
        for (x,y,w,h) in lists:
            #顔部分を切り取る
            clothes_cut = img[y-int(h*-1.4):y+int(h*3.7), x-int(w*0.6):x+int(w*1.8)] #img[y:y+h, x:x+w]マイナスにするとずらすことができる
        dst = cv2.resize(clothes_cut, (244, 235))
        cv2.imwrite('clothes_cut.jpg', dst)
        #cv2.imshow('clothes_cut',dst)
        #cv2.waitKey(0)
    else:
        lists = cascade.detectMultiScale(img_gray, minNeighbors = 4, minSize=(100, 100))
        if len(lists):
            for (x,y,w,h) in lists:
                #顔部分を切り取る
                clothes_cut = img[y-int(h*-1.4):y+int(h*3.7), x-int(w*0.6):x+int(w*1.8)] #img[y:y+h, x:x+w]マイナスにするとずらすことができる
```

```
            dst = cv2.resize(clothes_cut, (244, 235))
            cv2.imwrite('clothes_cut.jpg', dst)
            #cv2.imshow('clothes_cut',dst)
            #cv2.waitKey(0)
        else:
            lists = cascade.detectMultiScale(img_gray, minNeighbors = 3, minSize=(100, 100))
            if len(lists):
                for (x,y,w,h) in lists:
                    #顔部分を切り取る
                    clothes_cut = img[y-int(h*-1.4):y+int(h*3.7), x-int(w*0.6):x+int(w*1.8)] #img[y:y+h, x:x+w]マイナスにするとずらすことができる
                dst = cv2.resize(clothes_cut, (244, 235))
                cv2.imwrite('clothes_cut.jpg', dst)
                #cv2.imshow('clothes_cut',dst)
                #cv2.waitKey(0)
            else:
                print('服が見つかりませんでした')
                break
file_name='clothes_cut.jpg'

search_posx=50
search_posy=60
color_list = []
def remove_specified_values(arr, value):
    while value in arr:
        arr.remove(value)
#色を算出する関数
def color_judge(value):
    color_wheel=['赤','赤橙','オレンジ',
                 '黄橙','黄色','黄緑',
                 '緑','青緑','青',
                 '青紫','紫','赤紫','赤']
    index_value=int(round(value/(180/(len(color_wheel)-1)),0))
    return color_wheel[index_value]
#選択した場所の色を出力する
img=cv2.imread(file_name,cv2.IMREAD_COLOR)
cv2.circle(img,center=(search_posx,search_posy),radius=10,color=(255,255,255),thickness=3)
cv2.imwrite('img_point_'+str(search_posx)+'_'+str(search_posy)+'.jpg',img)
imghsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
img = cv2.imread("clothes_cut.jpg") #画像の読み込み
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #色配置の変換  BGR→RGB
def remove_specified_values(arr, value):
    while value in arr:
        arr.remove(value)
color_list = []
img_array = np.asarray(img) #numpy で扱える配列をつくる
#print(img_array)
#plt.imshow(img_array)
#plt.show()
search_posy = 60
for num in range(10):
    search_posx = 50
    for num in range(10):
        color=color_judge(imghsv[search_posy,search_posx,0])
        img_array = np.asarray(imghsv)
        img_HSV = img_array[search_posy,search_posx,:]
        cv2.circle(img,center=(search_posx,search_posy),radius=10,color=(255,255,255),thickness=3)
        img_HSV = img_array[search_posy,search_posx,:]
        #print(img_RGB)
        hsv_min = np.array([0,0,0])
        hsv_max = np.array([180,255,30])
        if hsv_min[0] <= img_HSV[0] <= hsv_max[0] and hsv_min[1] <= img_HSV[1] <= hsv_max[1] and hsv_min[2] <= img_HSV[2] <= hsv_max[2]:
            color = '黒色'
            #color_list.append('黒色')
        hsv_min = np.array([0,0,200])
```

```
                hsv_max = np.array([180,20,255])
                if hsv_min[0] <= img_HSV[0] <= hsv_max[0] and hsv_min[1] <= img_HSV[1] <= hsv_max[1] and hsv_min[2] <= img_HSV[2] <= hsv_max[2]:
                    color = '白色'
            #cv2.imwrite('img_point_'+str(search_posx)+'_'+str(search_posy)+'.jpg',img)
                color_list.append(color)
                #print(color)
                search_posx += 10
            search_posy += 10
        if 0 < color_list.count('赤') < 20:
            remove_specified_values(color_list, '赤')
        if 0 < color_list.count('赤橙') < 20:
            remove_specified_values(color_list, '赤橙')
        if 0 < color_list.count('オレンジ') < 20:
            remove_specified_values(color_list, 'オレンジ')
        if 0 < color_list.count('黄橙') < 20:
            remove_specified_values(color_list, '黄橙')
        if 0 < color_list.count('黄色') < 20:
            remove_specified_values(color_list, '黄色')
        if 0 < color_list.count('黄緑') < 20:
            remove_specified_values(color_list, '黄緑')
        if 0 < color_list.count('緑') < 20:
            remove_specified_values(color_list, '緑')
        if 0 < color_list.count('青緑') < 20:
            remove_specified_values(color_list, '青緑')
        if 0 < color_list.count('青') < 20:
            remove_specified_values(color_list, '青')
        if 0 < color_list.count('青紫') < 20:
            remove_specified_values(color_list, '青紫')
        if 0 < color_list.count('紫') < 20:
            remove_specified_values(color_list, '紫')
        if 0 < color_list.count('赤紫') < 20:
            remove_specified_values(color_list, '赤紫')
        if 0 < color_list.count('白色') < 20:
            remove_specified_values(color_list, '白色')
        if 0 < color_list.count('黒色') < 20:
            remove_specified_values(color_list, '黒色')
        print(set(color_list)) #set をつけることでかぶりを省く
        #print(color_list)
        cv2.circle(img,center=(search_posy,search_posx),radius=10,color=(255,255,255),thickness=3)
        cv2.imwrite('img_point_'+str(search_posy)+'_'+str(search_posx)+'.jpg',img)
        #cv2.imwrite("clothes_cut_result.jpg", img_array)
        #','.join(set(color_list))
        #data の配列に配列を追加する
        data.append([hair_cl, hair_lt, skin_cl, eyes_lt, ','.join(set(color_list))])
# 書き込む範囲の指定
#data_cell_range = sheet.iter_rows(min_row=2, min_col=1, max_row=len(data), max_col=3)
data_cell_range = sheet['A2':'E382']
# 表タイトルを書き込む
sheet['A1'].value = title_txt
sheet.merge_cells('A1:E1')          # セルの結合
sheet['A1'].alignment = Alignment(horizontal="center")   # 中央揃え
#書き込み
for i in range(len(data_cell_range)):
    for j in range(len(data_cell_range[0])):
        data_cell_range[i][j].value = data[i][j]
line_style = Side(style='thin', color='000000')
border = Border(top=line_style, bottom=line_style, left=line_style, right=line_style)
for row_number in range(2, 9):
    for col_number in range(1, 5):
        sheet.cell(row=row_number, column=col_number).border = border
# 表のタイトル,フォントのサイズ
font_style = Font(name='Noto Sans CJK JP', size=10, bold=True)
sheet['A1'].font = font_style
sheet['A2'].font = font_style
```

```
sheet['B2'].font = font_style
sheet['C2'].font = font_style
sheet['D2'].font = font_style
sheet['E2'].font = font_style
# 上書き保存
wbook.save('./osyare.xlsx')
# ワークシートの読み込みと内容を表示
df = pd.read_excel('./osyare.xlsx')
for row in df.values:
    print(row[0], row[1], row[2],row[3],row[4])
# ワークブックを閉じる
wbook.close()
```