

## 1 実験目的・課題

次の画像 (図 1) について、以下の 3 つの処理を行う。

- 画像に対して輪郭検出を行う
- 画像に対してフィルタ処理を行う
- 画像に対して自動検出によるマスク処理を行う



図 1 画像処理を行う画像

## 2 実装方法

### 2.1 画像の二値化

画像の二値化とは、画素の特定の要素において閾値を与え、0 か 1 に分類することである。二値化された画像は、図 2 のように白黒画像なる。

python の OpenCV では `threshold` という関数で画像の二値化を行うことが出来る。第 1 引数に入力画像、第 2 引数に閾値、第 3 引数は閾値以上の値を持つ画素に割り当てられる値、第 4 引数にいくつかある閾値処理のどれを使用するかフラグを受け取る。返り値は 2 つあり、使用した閾値と、二値化を適用した画像である。入力画像はグレースケール画像でなければならず、画像を読み込むときに `imread(name,0)` のように第 2 引数に 0 を与えると図 3 のようにグレースケール画像として読み込むことが出来る。



図 2 大津の二値化を適用した図 1



図 3 グレースケール画像

## 2.2 輪郭検出

輪郭とは、同じ色などについて境界に沿った連続する点をつなげることによって形成される曲線のことである。精度よく輪郭を検出するために、二値化画像を使用する。以下のコードで輪郭を検出し、元画像に描画することが出来る。

---

```

1  import cv2
2  img = cv2.imread('tapu.jpg')
3  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
4  ret, img_thres = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU)
5  img_cont, contours, hierarchy = cv2.findContours(img_thres, cv2.RETR_EXTERNAL, cv2.
      CHAIN_APPROX_SIMPLE)
6  cv2.drawContours(img, contours, -1, color=(0, 0, 255), thickness=2)

```

---

`findContours` の第二引数には、全ての輪郭を同等に扱う `RETR_LIST`、入れ子構造になっている輪郭のもっとも外側の輪郭のみを検出する `RETR_EXTERNAL` などを指定する。第三引数では、全ての点を検出する `CHAIN_APPROX_NONE`、冗長な点を削除する `CHAIN_APPROX_SIMPLE` 等を指定する。



図 4 `RETR_LIST` で指定した場合

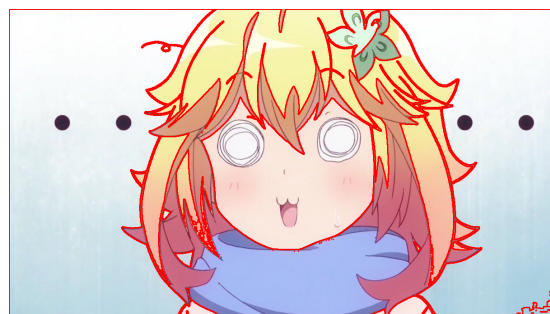


図 5 `RETR_EXTERNAL` で指定した場合

図 4 に、`findContours` の第二引数に `RETR_LIST` を指定したもの、図 5 に、`RETR_EXTERNAL` を指定したものを示す。図 5 では、図 4 では描画されていた顔の中の輪郭などが検出されていないことがわかる。

## 2.3 フィルタ処理

画像に対してローパスフィルタ (LPF) やハイパスフィルタ (HPF) によるフィルタリングをすることが出来る。LPF ではノイズの除去や画像のぼかしが、HPF ではエッジの強調ができる。

filter2D 関数は、入力画像とカーネル (フィルタ) の畳み込みを計算する。式 1 は、画像の平滑化を表す。

$$K = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (1)$$

式 1 を使ったフィルタリングを行うと、各画素に対してその画素を中心に  $5 \times 5$  の画素を選択し、その合計を 25 で割ったものを出力画像でのその画素の値にすることが出来る。

OpenCV ではエッジ検出が出来る Canny や、エッジを劣化させずに平滑化できる bilatetalFilter など多くのフィルタが利用できる。

## 2.4 マスク処理

読み込んだ画像は通常、画素ごとに [G,B,R] の array でデータを持つ。これに対して図 6 のようなマスク画像を用意し bitwise\_and をとると、図 7 のような画像が生成できる。

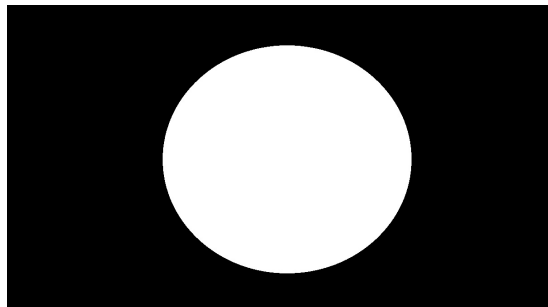


図 6 マスク画像



図 7 bitwise\_and を行った画像

これはマスク画像の白い部分の画素は [0b11111111,0b11111111,0b11111111] で表されるため、どんな画素と and をとっても元の画像の値が残り、逆に、黒い部分は [0b00000000,0b00000000,0b00000000] で表されるためどんな値と and をとっても [0,0,0] つまり黒になってしまうためである。

## 3 結果と考察

### 3.1 輪郭検出

以下のコードによって輪郭検出をし描画したものを図 8 に示す。6 行目の処理では contourArea という関数によって検出した輪郭の面積が小さいものはリムーブするようにしている。これによって図 5 で見られたあご周辺や髪先での点が消えていることがわかる。

---

```
1 import cv2
2 img = cv2.imread('tapu.jpg')
3 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
4 ret, img_thres = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU)
5 img_cont, contours, hierarchy = cv2.findContours(img_thres, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)
6 contours = list(filter(lambda x: cv2.contourArea(x) > 100, contours))
7 cv2.drawContours(img, contours, -1, color=(0, 0, 255), thickness=2)
```

---



図 8 面積 100 以上の領域のみ輪郭検出

次に、画像の二値化の閾値を大津のアルゴリズムによる自動決定から手動で設定したものを図 9 に示す。図 8 では検出できなかったマフラーや左側の髪が検出できている。これは大津の二値化では髪やマフラーがすべて閾値以下に判断されてしまい、塗りつぶされてしまったためである。



図 9 手動で二値化をして輪郭検出をしたもの

### 3.2 フィルタ処理

### 3.3 自動検出によるマスク処理

## 参考文献

- [1] OpenCV-Python チュートリアル文書  
<http://labs.eecs.tottori-u.ac.jp/sd/Member/oyamada/OpenCV/html/index.html>
- [2] OpenCV / findContours を使用して画像中のオブジェクトの輪郭を検出する方法  
<https://axa.biopapyrus.jp/ia/OpenCV/detect-contours.html>