画像認識資料　つくばろぼっとサークル用

2023 年 11 月

石渡真之亮

１．　この資料について

この資料は、つくばろぼっとサークル内で、画像認識についてゼロから学びたい人を対象に作成したものである。本資料作成者はソフトについてはまだ俄かなので質は高くはない。改善点があった場合は適宜変えていってほしい。また、この資料は以下のサイトなどを参考にしているため、詳しく知りたい人は各自参照するとよい。

https://shikaku-mafia.com/python-opencv-function/
https://www.klv.co.jp/corner/python-opencv-image-output.html
https://piccalog.net/python/detect-airplane-opencv
https://shikaku-mafia.com/opencv-circle/
https://shikaku-mafia.com/cv2-houghcircles/

２．　画像認識について

人間は目を使うことで視覚情報から状況を判断する。コンピュータ上でこれを行おうとすると目の代わりに Web カメラや USB カメラなどを用いて画像認識という形で画像から特徴を検出して状況を認識する。画像認識は深層学習の登場により飛躍的に発展していったが、今回はその基本となる色の識別やグレースケール化などの表現方法、動体検知などについて学ぶ。

３．　環境構築について

使用する環境について以下に示す。コードは VSCode を用いて作成していく。各モジュールのインストール方法は下記を参考にした。（アナコンダプロンプトで行った）

NumPy：https://deepage.net/features/numpy-install.html
$ pip install numpy
OpenCV：https://qiita.com/fiftystorm36/items/1a285b5fbf99f8ac82eb

$ pip install opencv-python

(1)Windows で使用する場合の例
Windows 10 22H2(OS ビルド 19045.3448)
Python 3,9,13
NumPy 1.23.5
OpenCV 4.6.0

(2)ubuntu で使用する場合の例
Ubuntu 20.04.6 LTS (Focal Fossa)
Python 3.8.10
NumPy 1.24.3
OpenCV 4.8.1

Python のバージョン確認はターミナル上で
$ python -V
などのようにコマンドを打って確認する。
NumPy や OpenCV については

```
import numpy as np
import cv2

print(np.__version__)
print(cv2.__version__)
```

のようにサンプルコードを作成するとバージョンが表示される。
過去にインストールしていてバージョンアップしたい場合は
$ pip uninstall ○○
$ pip install ○○
と打つ。

４．　基本的な関数について

OpenCV には様々な関数があり、画像に対して処理を積み重ねることで目的の動作を実現することができる。今回はその基本となる関数について紹介する。(cv2.□□のように、インポートしたモジュールの中の関数（≒メソッド）を使いたいときは、(インポートしたモジュール).(使う関数)のように記述して用いること。)

① 画像を読み込む→cv2.imread()
② 画像をディスプレイに出力する→cv2.imshow()
③ 画像をグレースケール化する→cv2.cvtColor()
④ 画像を保存する→cv2.imwrite()
⑤ 画像に円を描画する→cv2.circle()
⑥ 画像に四角形を描画する→cv2.rectangle()
⑦ 画像から円を検出する→cv2.HoughCircles()
⑧ 画像から輪郭を検出する→cv2.findContours()
⑨ [発展]
現在フレームと移動平均との差を計算する→cv2.accumulateWeighted()

① 画像を読み込みたいときは cv2.imread()関数を使う。

```
cv2.imread("パス")
```

第 1 引数に読み込みたい画像のパスを入れると任意の画像を読み込むことができる。

② 画像をディスプレイに出力したいときは cv2.imshow()関数を使う。

```
cv2.imshow("任意の表示名", 表示したい画像)
```

但し、このままでは空のウィンドウが一瞬だけ表示されてプログラムが終了してしまうため、cv2.waitKey()関数をその後に記述する必要がある。これは第 1 引数の数値（ms）だけ処理を待つ関数である。しかし、

```
cv2.waitKey(0)
```

だけは例外で、キーボードが押されるまでずっと処理を待つという意味になる。

```
if cv2.waitKey(1) == 13
```

と記述すると、「13 に対応する Enter キーが押されたら」という意味になる。
各番号に対応するキーは OpenCV_Key_num.pdf（下記サイトのエクセルデータを pdf 化したもの）を参照すること。
参考文献：https://www.s-toki.net/it/cv2-key-num/

さて、ここまで終わったら①と②を使った簡単なコードを書いてみよう。
ワークスペース内に test.py というコードを作成して、その中に新規フォルダを作成して画像ファイルを置く。次にこのコードに以下の内容を記述する。すると、画像ファイルがディスプレイに出力される。このとき Enter キーを押すとディスプレイウィンドウが消されてプログラムが終了する。~~私は画像ファイルとして月姫 R のとあるシーンを選んで出力されるウィンドウ名を Arcueid & Shiki としたが、こ~~

~~れは任意である。この写真について興味ある人は是非調べてみてください。この作品は非常に良ｋ~~

```
import numpy as np # Numpy モジュールを np というあだ名を付けてインポートする
import cv2 # OpenCV モジュールをインポートする

#print(np.__version__) # Numpy のバージョン確認
#print(cv2.__version__) # OpenCV のバージョン確認

pic = cv2.imread("dataset/tsukihime.jpg") # tsukihime.jpg という画像ファイルを読み込んで pic に格納


while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki",pic) # pic を Arcueid & Shiki という名前で画面に出力
    if cv2.waitKey(1) == 13: # Enter キー(ここでは 13 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki") # ウィンドウを破棄
        break # While 文からの脱出。つまり画面出力をやめる
```

③ 画像をグレースケール化したいときは cv2.cvtColor()関数を使う。

```
cv2.cvtColor(対象の画像, cv2.COLOR_BGR2GRAY)
```

cv2.cvtColor()関数は取得した画像の色空間を変換する関数である。第 2 引数では色空間の指定を示しており、どの色空間にしたいかを記述する。ここでは cv2.COLOR_BGR2GRAY によりグレースケール化を行っている。他にも色空間の種類は存在しており、

cv2.COLOR_BGR2HSV：HSV に変換

cv2.COLOR_BGR2YCrCb：YCrCb に変換

cv2.COLOR_BGR2XYZ：XYZ に変換

などがある。~~(BGR to ○○…？)~~詳しく知りたい人は以下を参照すること。

参考文献：https://qiita.com/spc_ehara/items/5c35d3c9900ad5e18e5c

④ 画像に処理を施した後どのようになったのかを確認するためには保存する必要がある。このとき使う関数が cv2.imwrite()関数である。

```
cv2.imwrite("保存するときの名前", 対象の画像)
```

第 1 引数では保存したい場所のパスを入れた名前とすること。

さて、ここで①～④までの内容に関するコードを書いてみよう。以下の内容を記述する。すると、新たにグレースケール化された画像ファイルが指定したディレクトリ下に保存されているので確認してほしい。また、元の画像ファイルのウィンドウを消した後にはグレースケール化された画像がディスプレイに表示されるようにした。スペースキーを押すとそのウィンドウが消されてプログラムが終了する。

```
import numpy as np # Numpy モジュールを np というあだ名を付けてインポートする
import cv2 # OpenCV モジュールをインポートする

#print(np.__version__) # Numpy のバージョン確認
#print(cv2.__version__) # OpenCV のバージョン確認

pic = cv2.imread("dataset/tsukihime.jpg") # tsukihime.jpg という画像ファイルを読み込んで pic に格納

gray = cv2.cvtColor(pic, cv2.COLOR_BGR2GRAY) # pic をグレースケール化

cv2.imwrite("dataset/gray_of_tsukihime.jpg", gray) # gray を保存


while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki", pic) # pic を Arcueid & Shiki という名前で画面に出力

    if cv2.waitKey(1) == 13: # Enter キー(ここでは 13 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("gray of Arcueid & Shiki", gray) # pic を gray of Arcueid & Shiki という名前で画面に出力

    if cv2.waitKey(1) == 32: # space キーを(ここでは 32 が対応)押したら
        cv2.destroyWindow("gray of Arcueid & Shiki") # ウィンドウを破棄
        break
```

次からは画像に対して描画を行ったり検出していったりしていく。ここから後

半戦なので各自ここで少し休憩することをおすすめする。

⑤ 画像に円を描画するには cv2.circle()関数を使う。

```
cv2.circle(対象の画像, 中心座標(x 座標, y 座標), 半径, 円周の色(B, G, R),
線の太さ, 描画するアルゴリズムの種類, 各座標における小数点以下の桁を表すビット数)
```

第 6,7 引数は基本特に指定はしなくてもよく、違いはほとんど見られない。引数を渡さなかったらデフォルトの値が入力される。また、第 5 引数である線の太さだが、マイナスの値を入れるとその描画内を塗りつぶすことができる。

##

余談ではあるが、以下のコードを実行すると、

```
import numpy as np # Numpy モジュールを np というあだ名を付けてインポートする
import cv2 # OpenCV モジュールをインポートする

#print(np.__version__) # Numpy のバージョン確認
#print(cv2.__version__) # OpenCV のバージョン確認

pic = cv2.imread("dataset/tsukihime.jpg") # tsukihime.jpg という画像ファイルを
読み込んで pic に格納

gray = cv2.cvtColor(pic, cv2.COLOR_BGR2GRAY) # pic をグレースケール化

circle = cv2.circle(pic, (500, 300), 150, (0, 255, 0), 10, cv2.LINE_AA,
0) #

cv2.imwrite("dataset/gray_of_tsukihime.jpg", gray) # gray を保存

cv2.imwrite("dataset/tsukihime_with_circle.jpg", circle) # circle を保存

cv2.imwrite("dataset/tsukihime2.jpg", pic) # pic を保存

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki", pic) # pic を Arcueid & Shiki という名前で画
面に出力

    if cv2.waitKey(1) == 13: # Enter キー(ここでは 13 が対応)を押したら
```

```
        cv2.destroyWindow("Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("gray of Arcueid & Shiki", gray) # pic を gray of Arcueid & Shiki という名前で画面に出力

    if cv2.waitKey(1) == 32: # space キーを(ここでは 32 が対応)押したら
        cv2.destroyWindow("gray of Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with a circle", circle) # circle を Arcueid & Shiki with a circle という名前で画面に出力

    if cv2.waitKey(1) == 97: # a キー(ここでは 97 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with a circle") # ウィンドウを破棄
        break
```

QMimeDatabase: Error loading internal MIME data
An error has been encountered at line 1 of <internal MIME data>: Premature end of document.:
というエラーが出て、本来円が描画されないはずの pic に円が描画されてしまった。このエラーが何を意味しているのか分からなかったので Chat GPT くんに聞いてみたら、

提供されたソースコードを実行した際に発生したエラーは、`QMimeDatabase` 関連の内部データの読み込みエラーです。このエラーは通常、MIME データベースに問題があることを示しています。しかし、それが円の描画に影響を与えるかどうかは明確ではありません。

円の描画に関する問題について、`cv2.circle()` 関数は元の画像を変更し、新しい画像を返すのではなく、元の画像自体を変更します。そのため、`pic` に対して円を描画した場合、`pic` 自体が変更されます。

したがって、円の描画が `dataset/tsukihime_with_circle.jpg` に反映されるのは正常な動作です。もし円の描画を `pic` のコピーに行いたい場合は、新しい変数に `pic` をコピーしてから `cv2.circle()` を適用する必要があります。

と答えてくれた。どうやら、cv2.circle()関数の第 1 引数に pic を入れてしまったためらしい。~~色空間の変換と描画ではやっていることは違うから…？~~しかし、エラーメッセージはそのままなので、もし解決できたら書き足します。それか教えてください泣

##

これまでの内容について⑤に関することを付け加えてコードを書いてみよう。以下の内容を記述する。これにより、円の描画が上手くいっていることを確認してほしい。

```
import numpy as np # Numpy モジュールを np というあだ名を付けてインポートする
import cv2 # OpenCV モジュールをインポートする

#print(np.__version__) # Numpy のバージョン確認
#print(cv2.__version__) # OpenCV のバージョン確認

pic = cv2.imread("dataset/tsukihime.jpg") # tsukihime.jpg という画像ファイルを読み込んで pic に格納
pic2 = pic.copy() # pic2 に pic をコピーする。後でこれに円を描画するため

gray = cv2.cvtColor(pic, cv2.COLOR_BGR2GRAY) # pic をグレースケール化

circle = cv2.circle(pic2, (500, 300), 150, (0, 255, 0), 10, cv2.LINE_AA, 0) # 円を描画

cv2.imwrite("dataset/gray_of_tsukihime.jpg", gray) # gray を保存

cv2.imwrite("dataset/tsukihime_with_circle.jpg", circle) # circle を保存

cv2.imwrite("dataset/tsukihime2.jpg", pic) # pic を保存

while True: # 無限ループ
```

```
    cv2.imshow("Arcueid & Shiki", pic) # pic を Arcueid & Shiki という名前で画面に出力

    if cv2.waitKey(1) == 13: # Enter キー(ここでは 13 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("gray of Arcueid & Shiki", gray) # pic を gray of Arcueid & Shiki という名前で画面に出力

    if cv2.waitKey(1) == 32: # space キーを(ここでは 32 が対応)押したら
        cv2.destroyWindow("gray of Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with a circle", circle) # circle を Arcueid & Shiki with a circle という名前で画面に出力

    if cv2.waitKey(1) == 97: # a キー(ここでは 97 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with a circle") # ウィンドウを破棄
        break
```

⑥ 画像に四角形を描画するには cv2.rectangle()関数を使う。

```
cv2.rectangle(対象の画像，左上の頂点の座標(x 座標，y 座標)，右下の頂点の座標(x 座標，y 座標)，円周の色(B，G，R)，線の太さ，描画するアルゴリズムの種類，各座標における小数点以下の桁を表すビット数)
```

円を描画するときと同様に第 5 引数でマイナスの値を入力すると塗りつぶしを行うことができる。

ここまでの内容を再びコードに書く。(分かりやすく整理もしました) rect という四角形を描画するための変数を置いて、四角形の描画を行う。実際に描画がされていることを確認してほしい。

```
################################################
################################################
################################################
```

```
#モジュールのインポートを行う
################################################
################################################
################################################

import numpy as np # Numpy モジュールを np というあだ名を付けてインポートする
import cv2 # OpenCV モジュールをインポートする

################################################
################################################
################################################
#バージョン確認を行う
################################################
################################################
################################################

print("NumPy のバージョンは", np.__version__) # Numpy のバージョン確認
print("OpenCV のバーージョンは", cv2.__version__) # OpenCV のバージョン確認

################################################
################################################
################################################
#画像ファイルを読み込んだりコピーしたりする
################################################
################################################
################################################

pic = cv2.imread("dataset/tsukihime.jpg") # tsukihime.jpg という画像ファイルを
読み込んで pic に格納
pic2 = pic.copy() # pic2 に pic をコピーする。後でこれに円を描画するため
pic3 = pic.copy() # pic3 に pic をコピーする。後でこれに四角形を描画するため


################################################
################################################
################################################
```

```
#色変換や描画を行う
################################################
################################################
################################################

gray = cv2.cvtColor(pic, cv2.COLOR_BGR2GRAY) # pic をグレースケール化

circle = cv2.circle(pic2, (500, 300), 150, (0, 255, 0), 10, cv2.LINE_AA,
0) # 円を描画

rect = cv2.rectangle(pic3, (1300, 40), (1550, 330), (0, 0, 255), 10,
cv2.LINE_AA, 0) # 四角形を描画

################################################
################################################
################################################
#画像保存を行う
################################################
################################################
################################################

cv2.imwrite("dataset/gray_of_tsukihime.jpg", gray) # gray を保存

cv2.imwrite("dataset/tsukihime_with_circle.jpg", circle) # circle を保存

cv2.imwrite("dataset/tsukihime_with_square.jpg", rect) # rect を保存

cv2.imwrite("dataset/tsukihime2.jpg", pic) # pic を保存

################################################
################################################
################################################
#画面出力を行う
################################################
################################################
################################################
```

```
while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki", pic) # pic を Arcueid & Shiki という名前で画面に出力

    if cv2.waitKey(1) == 13: # Enter キー(ここでは 13 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("gray of Arcueid & Shiki", gray) # pic を gray of Arcueid & Shiki という名前で画面に出力

    if cv2.waitKey(1) == 32: # space キーを(ここでは 32 が対応)押したら
        cv2.destroyWindow("gray of Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with a circle", circle) # circle を Arcueid & Shiki with a circle という名前で画面に出力

    if cv2.waitKey(1) == 97: # a キー(ここでは 97 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with a circle") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with a square", rect) # rect を Arcueid & Shiki with a square という名前で画面に出力

    if cv2.waitKey(1) == 115: # s キー(ここでは 115 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with a square") # ウィンドウを破棄
        break
```

⑦ 画像から円を検出するには cv2.HoughCircles()関数を使う。

ここまでやってきて、NumPy をわざわざインポートしたのに一度も使ってないじゃないかッ！と思う人もいるかもしれないが、ここから使っていくので

安心してほしい。

```
cv2.HoughCircles(対象の画像, cv2.HOUGH_GRADIENT, dp, minDist, param1, param2, minRadius, maxRadius)
```

第 1 引数では対象の画像を入力する。

第 2 引数では Hough 変換の手法を選択する。OpenCV 独自のコードで指定するが、この引数は内部的な話になってワケワカメなのでとりあえず円を検出するための勾配法である関数の cv2.HOUGH_GRADIENT としよう。

第 3 引数では解像度の比率を示すものとなっていて、この値が大きくほど検出基準がガバガバになっていって、小さくなるほど検出基準がキツキツになっていく。0.8~1.2 くらいで調整していくと良い。0 とするとエラーが発生する。

第 4 引数では検出される円同士が最低限これだけ離れていないと嫌なんだと思っている距離を入力する。Hough 検出では同じ円に対して複数の検出結果を返してしまうためこの引数はわりと大切な引数である。

第 5 引数では Canny 法における Hysteresis 処理の閾値を表していて、上限値をここで設定している。よく分からなければ大体 100 くらいに設定していれば良い。

第 6 引数では円の中心を検出する際の閾値を表していて、高くしすぎると未検出が多くなり、低くしすぎると円の誤検出が多くなる。第 5 引数よりも重要なとっっっっっても大切な引数なので上手く調整するとよい。

第 7 引数では検出される円の半径の最低値を設定する。

第 8 引数では検出される円の半径の最大値を設定する。

画像から円を検出して確認するまでの過程はこれまでと少し異なり、段階を踏まなくてはならない。

1. 画像の読み込み
2. 画像のグレースケール化
3. 画像から円を検出する
4. 検出した情報をキャストして丸める
5. 検出した円を描画する
6. 画像を保存する or ディスプレイに表示する

2 で何故円を検出する前にグレースケール化を行うのか疑問に思ってほしい。そもそもグレースケール化とはカラーの画像を単一の明るさの値のみを持つモノクロ画像に変換することであり、これにより情報量が削減されて単純化されるためノイズに対して強くなるというメリットがある。したがって円をなるべく精度良く検出するため（円の検出に限らずどの検出においても）には必要不可欠であると言えるのだ。

cv2.HoughCircles()関数の戻り値は np.float という NumPy における独自クラス型となっているため、5 において描画するためには⑤で用いたいくつかの引数が int 型でなくてはならない cv2.circle()関数に合わせて np.float 型から int 型に丸めた（ここで用いる関数が np.around()関数）後に 16 ビット情報にキャストする（ここで用いる関数が np.uint16()関数）必要がある。以上の処理をするとやっと検出できた円を描画することができる。

ここまで大変だったが、円の検出とその描画を行うコードを作成してみよう。以下の内容を記述すると pic4 というコピーされた画像に対して円の検出＆描画がされて更新された画像が出力される。circles[0, :]は、検出された全ての円の情報を含む配列であり、それぞれの円は中心座標と半径の情報を持っている。したがって、for circle in circles[0, :]により配列 circles[0, :]の各要素に対して順番に繰り返し処理を行うことができる。（各円を circle として参照している）円周については円の中心座標は(circle[0], circle[1])であり、circle[2]により円の半径を表している。円の中心については円の半径を 2 とすることで円の中心を小さい円として疑似的に表している。しかし、cv2.HoughCircles()関数の各パラメータは適当に置いているので、各自調整してどうなるのか確認しておくとよい。

```
################################################
################################################
################################################
#モジュールのインポートを行う
################################################
################################################
################################################

import numpy as np # Numpy モジュールを np というあだ名を付けてインポートする
import cv2 # OpenCV モジュールをインポートする

################################################
################################################
################################################
#バージョン確認を行う
################################################
################################################
################################################
```

```
print("NumPy のバージョンは", np.__version__) # Numpy のバージョン確認
print("OpenCV のバーージョンは", cv2.__version__) # OpenCV のバージョン確認

################################################
################################################
################################################
#画像ファイルを読み込んだりコピーしたりする
################################################
################################################
################################################

pic = cv2.imread("dataset/tsukihime.jpg") # tsukihime.jpg という画像ファイルを
読み込んで pic に格納
pic2 = pic.copy() # pic2 に pic をコピーする。後でこれに円を描画するため
pic3 = pic.copy() # pic3 に pic をコピーする。後でこれに四角形を描画するため
pic4 = pic.copy() # pic4 に pic をコピーする。後でこれの円を検出するため

################################################
################################################
################################################
#色変換や描画を行う
################################################
################################################
################################################

gray = cv2.cvtColor(pic, cv2.COLOR_BGR2GRAY) # pic をグレースケール化

circle0 = cv2.circle(pic2, (500, 300), 150, (0, 255, 0), 10, cv2.LINE_AA,
0) # 円を描画

rect = cv2.rectangle(pic3, (1300, 40), (1550, 330), (0, 0, 255), 10,
cv2.LINE_AA, 0) # 四角形を描画

circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, dp=1, minDist=20,
param1=100, param2=60, minRadius=0, maxRadius=0) # 円の検出
circles = np.uint16(np.around(circles)) # 円の検出結果を整数値に変換
```

```
for circle in circles[0, :]: # 検出されたすべての円に対してループ処理
    cv2.circle(pic4, (circle[0], circle[1]), circle[2], (0, 165, 255), 5)
# 円周の描画
    cv2.circle(pic4, (circle[0], circle[1]), 2, (0, 165, 255), 3) # 中心点
の描画

################################################
################################################
################################################
#画像保存を行う
################################################
################################################
################################################

cv2.imwrite("dataset/gray_of_tsukihime.jpg", gray) # gray を保存

cv2.imwrite("dataset/tsukihime_with_circle.jpg", circle0) # circle を保存

cv2.imwrite("dataset/tsukihime_with_square.jpg", rect) # rect を保存

cv2.imwrite("dataset/tsukihime_with_circles_by_censor.jpg", pic4) #
circles を保存

cv2.imwrite("dataset/tsukihime2.jpg", pic) # pic を保存

################################################
################################################
################################################
#画面出力を行う
################################################
################################################
################################################

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki", pic) # pic を Arcueid & Shiki という名前で画
面に出力
```

```
    if cv2.waitKey(1) == 13: # Enter キー(ここでは 13 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("gray of Arcueid & Shiki", gray) # pic を gray of Arcueid &
Shiki という名前で画面に出力

    if cv2.waitKey(1) == 32: # space キーを(ここでは 32 が対応)押したら
        cv2.destroyWindow("gray of Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with a circle", circle0) # circle を
Arcueid & Shiki with a circle という名前で画面に出力

    if cv2.waitKey(1) == 97: # a キー(ここでは 97 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with a circle") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with a square", rect) # rect を Arcueid &
Shiki with a square という名前で画面に出力

    if cv2.waitKey(1) == 115: # s キー(ここでは 115 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with a square") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with circles by sensor", pic4) # pic4 を
Arcueid & Shiki with という名前で画面に出力

    if cv2.waitKey(1) == 109: # m キー(ここでは 109 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with circles by sensor") # ウィ
ンドウを破棄
```

```
        break
```

⑧ 画像から輪郭を検出するには cv2.findContours()関数を使う。

```
cv2.findContours(2 値化された画像, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

第 1 引数では対象の画像を入力する。但し、その画像は 2 値化されている必要がある。したがって、この関数を記述する前に 2 値化する関数を記述しなければならないが、そこで使われるのが cv2.threshold()関数である。

```
cv2.threshold(グレースケール化された画像, 閾値, 閾値を超えた場合に割り当てられる値, cv2.THRESH_BINARY)
```

詳しい内容は省略する。

第 2 引数では輪郭検出の方法を指定する。cv2.RETR_TREE 関数により輪郭の階層構造を保持する。

第 3 引数では輪郭の近似方法を指定する。cv2.CHAIN_APPROX_SIMPLE 関数により輪郭の輪郭点を格納するが、これにより輪郭の形状を近似する。

画像から輪郭を検出した後はそれを描画しなければどうなっているのか確認できないので今度は cv2.drawContours()関数を用いて描画していく。

```
cv2.drawContours(対象の画像, 輪郭のリスト, -1, 輪郭の色(B, G, R), 輪郭の太さ)
```

第 2 引数で輪郭のリストを指定して第 3 引数で-1 とすることで全ての輪郭が第 4,5 引数で指定した輪郭の色と太さで描画される。

これが最後のコードである。今までの内容をまとめてみよう。

```
################################################
################################################
################################################
#モジュールのインポートを行う
################################################
################################################
################################################

import numpy as np # Numpy モジュールを np というあだ名を付けてインポートする
import cv2 # OpenCV モジュールをインポートする

################################################
################################################
```

```
################################################
#バージョン確認を行う
################################################
################################################
################################################

print("NumPy のバージョンは", np.__version__) # Numpy のバージョン確認
print("OpenCV のバーージョンは", cv2.__version__) # OpenCV のバージョン確認

################################################
################################################
################################################
#画像ファイルを読み込んだりコピーしたりする
################################################
################################################
################################################

pic = cv2.imread("dataset/tsukihime.jpg") # tsukihime.jpg という画像ファイルを
読み込んで pic に格納
pic2 = pic.copy() # pic2 に pic をコピーする。後でこれに円を描画するため
pic3 = pic.copy() # pic3 に pic をコピーする。後でこれに四角形を描画するため
pic4 = pic.copy() # pic4 に pic をコピーする。後でこれの円を検出するため
pic5 = pic.copy() # pic5 に pic をコピーする。後でこれの輪郭を検出するため

################################################
################################################
################################################
#色変換や描画を行う
################################################
################################################
################################################

gray = cv2.cvtColor(pic, cv2.COLOR_BGR2GRAY) # pic をグレースケール化

circle0 = cv2.circle(pic2, (500, 300), 150, (0, 255, 0), 10, cv2.LINE_AA,
0) # 円を描画
```

```
rect = cv2.rectangle(pic3, (1300, 40), (1550, 330), (0, 0, 255), 10,
cv2.LINE_AA, 0) # 四角形を描画

circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, dp=1, minDist=20,
param1=100, param2=60, minRadius=0, maxRadius=0) # 円の検出
circles = np.uint16(np.around(circles)) # 円の検出結果を整数値に変換
for circle in circles[0, :]: # 検出されたすべての円に対してループ処理
    cv2.circle(pic4, (circle[0], circle[1]), circle[2], (0, 165, 255), 5)
# 円周の描画
    cv2.circle(pic4, (circle[0], circle[1]), 2, (0, 165, 255), 3) # 中心点
の描画

_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY) # グレースケー
ル画像を 2 値化
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE) # 輪郭を検出
cv2.drawContours(pic5, contours, -1, (0, 0, 255), 3) # 輪郭を描画

################################################
################################################
################################################
#画像保存を行う
################################################
################################################
################################################

cv2.imwrite("dataset/gray_of_tsukihime.jpg", gray) # gray を保存

cv2.imwrite("dataset/tsukihime_with_circle.jpg", circle0) # circle を保存

cv2.imwrite("dataset/tsukihime_with_square.jpg", rect) # rect を保存

cv2.imwrite("dataset/tsukihime_with_circles_by_censor.jpg", pic4) # pic4
を保存
```

```
cv2.imwrite("dataset/tsukihime_with_contours.jpg", pic5) # pic5を保存

cv2.imwrite("dataset/tsukihime2.jpg", pic) # picを保存

################################################
################################################
################################################
#画面出力を行う
################################################
################################################
################################################

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki", pic) # picをArcueid & Shikiという名前で画面に出力

    if cv2.waitKey(1) == 13: # Enterキー(ここでは13が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("gray of Arcueid & Shiki", gray) # picをgray of Arcueid & Shikiという名前で画面に出力

    if cv2.waitKey(1) == 32: # spaceキーを(ここでは32が対応)押したら
        cv2.destroyWindow("gray of Arcueid & Shiki") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with a circle", circle0) # circleをArcueid & Shiki with a circleという名前で画面に出力

    if cv2.waitKey(1) == 97: # aキー(ここでは97が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with a circle") # ウィンドウを破棄
        break
```

```
while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with a square", rect) # rect を Arcueid &
Shiki with a square という名前で画面に出力

    if cv2.waitKey(1) == 115: # s キー(ここでは 115 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with a square") # ウィンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with circles by sensor", pic4) # pic4 を
Arcueid & Shiki with circles by sensor という名前で画面に出力

    if cv2.waitKey(1) == 109: # m キー(ここでは 109 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with circles by sensor") # ウィ
ンドウを破棄
        break

while True: # 無限ループ
    cv2.imshow("Arcueid & Shiki with contours", pic5) # circle を Arcueid &
Shiki with contours という名前で画面に出力

    if cv2.waitKey(1) == 113: # q キー(ここでは 113 が対応)を押したら
        cv2.destroyWindow("Arcueid & Shiki with contours") # ウィンドウを破棄
        break
```

ここまでいけた人は画像に対して色空間の変換ができたり、描画ができたり、簡単な特徴を抽出できたりする能力が身に付けられたということになる。ここからは動体を検出するために必要となる関数について紹介する。余力がある人向けの内容となっており、内容の説明は大きく省く。

⑨ 現在フレームと移動平均との差を計算するには cv2.accumulateWeighted()関数を使う。

```
cv2.accumulateWeighted(入力画像, 加重平均が格納される出力配列, 入力画像の重み係数)
```

第 3 引数では入力画像の重み係数を入力するが、これを設定することで動きの検出や背景差分処理の調整を行うことができる。

以下にロボコンで実装しようとしたが断念した内容のコードを記す。赤と青の動体を検知して、描画された長方形の座標を出力するものとなっている。

```
import cv2 #opencv モジュールを導入 ok
import time #time モジュールを導入 ok
import numpy as np #numpy モジュールを導入 ok
#numpy を np と省略して書く。 ok

class Color:
    BLACK     = '¥033[30m'
    RED       = '¥033[31m'
    GREEN     = '¥033[32m'
    YELLOW    = '¥033[33m'
    BLUE      = '¥033[34m'
    PURPLE    = '¥033[35m'
    CYAN      = '¥033[36m'
    WHITE     = '¥033[37m'
    END       = '¥033[0m'
    BOLD      = '¥038[1m'
    UNDERLINE = '¥033[4m'
    INVISIBLE = '¥033[08m'
    REVERCE   = '¥033[07m'
#----------------------
R_G_n = 15
B_G_n = 15
#----------------------

red_count = 0
blue_count = 0

red_num = 0
blue_num = 0

R_x = 0
R_y = 0
R_h = 0
```

```
R_w = 0

B_x = 0
B_y = 0
B_h = 0
B_w = 0

R_Gx = 0
R_Gy = 0
B_Gx = 0
B_Gy = 0

R_G = np.zeros((2,R_G_n))
B_G = np.zeros((2,B_G_n))

R_BGR = [40, 50, 170] #赤の代表値 ok
R_threshoud = 37 #赤を許容できる閾値 ok

RminBGR = np.array([R_BGR[0]-R_threshoud, R_BGR[1]-R_threshoud, R_BGR[2]-
R_threshoud]) # 赤判定の最小 ok
RmaxBGR = np.array([R_BGR[0]+R_threshoud, R_BGR[1]+R_threshoud,
R_BGR[2]+R_threshoud]) # 赤判定の最大 ok


B_BGR = [100, 50, 0] #青の代表値 ok
B_threshoud = 37 #青を許容できる閾値 ok

BminBGR = np.array([B_BGR[0]-B_threshoud, B_BGR[1]-B_threshoud, B_BGR[2]-
B_threshoud]) # 青判定の最小 ok
BmaxBGR = np.array([B_BGR[0]+B_threshoud, B_BGR[1]+B_threshoud,
B_BGR[2]+B_threshoud]) # 青判定の最大 ok



Real = cv2.VideoCapture(0) # リアルタイムのやつを Real に入れる ok
#cv2.VideoCapture()関数は動画を読み込んでくれるやつ。 ok
#第 1 引数に動画のパスを入れるとその動画を指定する。 ok
```

```
#第 1 引数に 0 を入れるとウェブカメラを指定する。1 を入れると USB で指定したリアルセンスとかになる。 ok
#(第 2 引数はビデオ処理メカニズムの指定でデフォルトは cv2.CAP_ANY) ok


Real.set(cv2.CAP_PROP_FPS, 90) #FPS 設定 ok
#VideoCapture オブジェクトの set()関数はプロパティを指定して値を変更できる。 ok
#全てのプロパティが変更できるわけでなく、カメラの場合は変更できる。 ok
#変更できるときは True を返すが、対応してないものを設定すると True を返すが、値は変更不可。 ok

red = (0, 0, 255) #枠線の色(BGR)-赤 ok
blue = (255, 0, 0) #枠線の色(BGR)-青 ok

R_before = None #前回の画像を保存する変数-Null みたいなやつ ok
B_before = None

fps = int(Real.get(cv2.CAP_PROP_FPS)) #動画(リアルタイム)の FPS(1 秒間に取得/表示する画像の枚数)を取得 ok
print('リアルタイム動画の fps は:',fps) #出力 ok

#------------------------------------------------------------------------


#------------------------------------------------------------------------

while True: # ∞に繰り返す ok
    ret, frame = Real.read() #(Real.read()に画像情報が格納されてたら ret は True となる。)画像を取得 ok
    if ret == False: break #(もし格納されてなかったら ret は False となる。)再生が終了したらループを抜ける ok


    #このグレースケール化する前の作業はフィルタかけるためのやつ
    R_maskBGR = cv2.inRange(frame, RminBGR, RmaxBGR) #画像から特定の色の範囲を指定する。ok
    B_maskBGR = cv2.inRange(frame, BminBGR, BmaxBGR)
```

```
    #cv2.inRange()関数は指定した色の範囲の画素を 255、それ以外の画素を 0 としてい 2 値化する関数。 ok
    #第 1 引数は元の画像の多次元配列。 ok
    #第 2 引数は抽出する画素の下限。 ok
    #第 3 引数は抽出する画素の上限。 ok
    #戻り値には、元の画像を 2 値化した多次元配列を返す。 ok4
    R_resultBGR = cv2.bitwise_and(frame, frame, mask = R_maskBGR) #元画像をマスク(覆い隠す)する。 ok
    B_resultBGR = cv2.bitwise_and(frame, frame, mask = B_maskBGR)
    #cv2.bitwise_and()関数はビット単位での AND 処理を行う関数。 ok
    #第 1、2 引数に入力画像(多次元配列)を入れ、それらの各画素ごとの値の AND が出力画像の画素値となる。 ok
    #1:2 枚の画像を用意。 ok
    #2:元画像，マスク(覆い隠し)画像内の、同じ位置のある画素に注目。 ok
    #3:元画像のある画素の値(40)，マスク(覆い隠し)画像のある画素の値(112)をそれぞれ 2 進数に変換。 ok
    #4:元画像のある画素の値(00101000)，マスク(覆い隠し)画像のある画素の値(01110000)の論理積を行う。 ok
    #5:2~4 を繰り返す。 ok

    R_gray = cv2.cvtColor(R_resultBGR, cv2.COLOR_BGR2GRAY) # グレースケール化 ok
    B_gray = cv2.cvtColor(B_resultBGR, cv2.COLOR_BGR2GRAY)
    #cvtColor()関数は取得した画像の色空間を変換してくれるやつ。 ok
    #第 2 引数の cv2.COLOR_BGR2GRAY はグレースケールにさせるやつ。 ok
    #(第 2 引数は他にも cv2.COLOR_BGR2HSV や cv2.COLOR_BGR2YCrCb や cv2.COLOR_BGR2XYZ などがある。) ok




    if R_before is None: # before が None だったら ok
        R_before = R_gray.astype("float") # 1 フレームを取って変数に格納。常に現在のフレームと以前フレームと比較するため取っておく必要がある。現在のフレームと以前フレームは両方グレースケール画像。 ok
```

```
        continue # while ループ戻る ok

    if B_before is None:
        B_before = B_gray.astype("float")
        continue

    cv2.accumulateWeighted(R_gray, R_before, 0.92)  # 現在のフレームと移動平均との差を計算。重みは蓄積し続ける。 ok
    cv2.accumulateWeighted(B_gray, B_before, 0.92)
    # 第 1 引数は最新の値。ここではグレースケール化したやつ。 ok
    # 第 3 引数は「どれくらいの早さで以前の画像を忘れるか」。大きいほど最新のデータの影響大 ok

    R_frameDelta = cv2.absdiff(R_gray, cv2.convertScaleAbs(R_before))
    B_frameDelta = cv2.absdiff(B_gray, cv2.convertScaleAbs(B_before))
    #absdiff()関数は第 1 引数の画像と第 2 引数の画像の差分を抽出する関数。 ok
    #1 つ目はグレースケール化したやつ
    #2 つ目は before を 8 ビット符号なし整数変換したやつ。なにこれ。てか、before って何なん。 ok

    R_thresh = cv2.threshold(R_frameDelta, 3, 255, cv2.THRESH_BINARY)[1]
    B_thresh = cv2.threshold(B_frameDelta, 3, 255, cv2.THRESH_BINARY)[1]
    #frameDelta の画像を2値化 ok
    #2 値化とは、画像をある閾値を基準に、2 つの諧調で表現される 2 値画像(バイナリイメージ)に変換する処理のこと。要は、各画素の値をハイかローかで表現するってこと。 ok
    #threshold()関数は 1 つの閾値により 2 値化を行う関数。 ok
    #第 1 引数は入力画像 ok
    #第 2 引数は閾値 ok
    #第 3 引数は maxValue。cv2.THRESH_BINARY か cv2.THRESH_BINARY_INV の場合に使用する 255 は白ってこと？ ok
    #第 4 引数は 2 値化の方法。THRESH_BINARY タイプを指定しているが、これは閾値以下は 0、閾値より大きいと maxValue にする。 ok
    #(2 値化の方法は他にもある。THRESH_BINARY_INV とか THRESH_TRUNC とか THRESH_TOZERO とか THRESH_TOZERO_INV とか THRESH_OTSU とか THRESH_TRIANGLE とか。ok)
    #参考:https://pystyle.info/opencv-image-binarization/ ok
```

```
    R_contours = cv2.findContours(R_thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[0]
    B_contours = cv2.findContours(B_thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[0]
    #findContours()関数は 2 値化された画像中の白の部分の外側の輪郭のデータを取得する関数。 ok
    #第 1 引数は輪郭抽出する画像データ ok
    #第 2 引数は輪郭構造の取得方法。RETR_EXTERNAL タイプを指定しているが、これは一番外側の白の輪郭だけを取得する。 ok
    #第 3 引数は輪郭座標の取得方法。CHAIN_APPROX_SIMPLE タイプを指定しているが、これは縦、横、斜め 45°方向に完全に直線の部分の輪郭の点を省略する。 ok
    #(輪郭座標の取得方法は他にもある。CHAIN_APPROX_NONE とか CHAIN_APPROX_TC89_L1 とか CHAIN_APPROX_TC89_KCOS とか。 ok)
    #参考:https://imagingsolution.net/program/python/opencv-python/opencv-python-findcontours/ ok

    for target in R_contours: # 差分があった点を画面に描く ok
        R_x, R_y, R_w, R_h = cv2.boundingRect(target) # target は輪郭を表す座標…? ok
        #boundingRect()関数は点群から外接矩形の座標を計算できる関数。 ok
        if R_w < 30: continue # 小さな変更点は無視。…ここ変えれば差分の具合が変えられる…? ok
        cv2.rectangle(frame, (R_x, R_y), (R_x+R_w, R_y+R_h), red, 2)
        #rectangle()関数は画像に長方形を描画する関数。 ok
        #第 1 引数は対象の画像データ ok
        #第 2 引数は描画する長方形の左上頂点の座標( , ) ok
        #第 3 引数は描画する長方形の右下頂点の座標( , ) ok
        #第 4 引数は色 ok
        #第 5 引数は描画する線の太さ。ここでは 2 ピクセルだねぇ(負の値にすると中を塗りつぶしできる) ok
        #w は外接矩形の幅で h は高さ ok
        #他にも第 6、7 引数があるけどここでは割愛 ok


        #print(Color.RED+"赤は今",red_count+1,"回描画されたよ¥n"+Color.END)
```

```
        R_Gx = R_x+R_w/2
        R_Gy = R_y+R_h/2

        R_G[0][red_count] = R_Gx
        R_G[1][red_count] = R_Gy

        if red_count==R_G_n-1:
            print(R_G)
            Rx_mean = np.mean(R_G[0],axis=0)
            Ry_mean = np.mean(R_G[1],axis=0)
            red_count = -1
            print("赤の重心位置のｘ座標は",Rx_mean,"だよぉ")
            print("赤の重心位置のｙ座標は",Ry_mean,"だよぉ")
            break
        red_count  += 1

    for target in B_contours:
        B_x, B_y, B_w, B_h = cv2.boundingRect(target)
        if B_w < 30: continue
        cv2.rectangle(frame, (B_x, B_y), (B_x+B_w, B_y+B_h), blue, 2)
        #print(Color.BLUE+"青は今",blue_count+1,"回描画されたよ
¥n"+Color.END)

        B_Gx = B_x+B_w/2
        B_Gy = B_y+B_h/2

        B_G[0][blue_count] = B_Gx
        B_G[1][blue_count] = B_Gy

        if blue_count==B_G_n-1:
            print(B_G)
            Bx_mean = np.mean(R_G[0],axis=0)
            By_mean = np.mean(R_G[1],axis=0)
            blue_count = -1
            print("青の重心位置のｘ座標は",Bx_mean,"だよぉ")
            print("青の重心位置のｙ座標は",By_mean,"だよぉ")
```

```
            break
        blue_count  += 1



    time.sleep(1/fps) # (引数に設定した時間(秒)の間、プログラムが停止する。)ウィンドウでの再生速度を元動画と合わせる。 ok
    cv2.imshow("Result R_BGR", R_resultBGR) #描画出力
    cv2.imshow("Result R_mask", R_maskBGR) #描画出力
    cv2.imshow("Result B_BGR", B_resultBGR) #描画出力
    cv2.imshow("Result B_mask", B_maskBGR) #描画出力
    cv2.imshow('NOW!!!!!', frame) # ウィンドウで表示 ok
    if cv2.waitKey(1) == 13: break # Enter キー(ここでは 13 が対応)が押されたらループを抜ける ok

cv2.destroyAllWindows() # ウィンドウを破棄 ok
cv2.waitKey(1) # 1ms だけ遅延 ok

#○○.□□()は、○○っていうクラス(モジュール)の中の□□メソッドっていう意味 ok


#描画する個数の制限したかったな

#HCL 設定もありだったな

#今回は赤(244,54,76)、青(48,127,226)、紫(161,90,149)
#https://aburobocon2024.vtv.gov.vn/gamerules
```

５．　最後に

今回は忙しい中、画像認識についての講習を受けてくださりありがとうございました。拙い説明で、参考にした文献を整理したものが大部分を占めてしまいましたが、分かりやすく書き直してまとめたので皆さんの理解の一助になってればと思います。一番重要な動体の検知の部分を端折ってしまい申し訳ありません。時間に余裕が出たときにまた内容を更新していきたいと思います。それでは