

Q. 41 - 50

Q.41. Cannyエッジ検出 (Step.1) エッジ強度

Q.41 - 43 ではエッジ検出手法の一つであるCanny法の理論となる。

Canny法は、

1. ガウシアンフィルタを掛ける
2. x, y方向のSobelフィルタを掛け、それらからエッジ強度とエッジ勾配を求める
3. エッジ勾配の値から、Non-maximum suppression によりエッジの細線化を行う
4. ヒステリシスによる閾値処理を行う

以上により、画像からエッジ部分を抜き出す手法である。

ここでは、1と2の処理を実装する。

処理手順は、

1. 画像をグレースケール化する
2. ガウシアンフィルタ(5x5, s=1.4)をかける
3. x方向、y方向のsobelフィルタを掛け、画像の勾配画像fx, fyを求め、勾配強度と勾配角度を次式で求める。

```
勾配強度 edge = sqrt(fx^2 + fy^2)
勾配角度 angle = arctan(fy / fx)
```

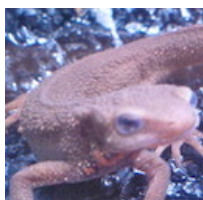
4. 勾配角度を次式に沿って、量子化する。

ただし、angleはradianから角度(degree)にして、-22.5から157.5の範囲をとるように値が修正してから、以下の計算を行う。

```
angle = {    0  (if -22.5 < angle <= 22.5)
          45  (if 22.5 < angle <= 67.5)
          90  (if 67.5 < angle <= 112.5)
          135 (if 112.5 < angle <= 157.5)
```

ただし、フィルタリングをパディングする際は、numpy.pad()を用いて、エッジの値でパディングせよ。

入力	出力(勾配強度)	出力(勾配角度)
(imori.jpg)	(answers_image/answer_41_1.jpg)	(answers_image/answer_41_2.jpg)



答え

- Python >> [answers_py/answer_41.py](#)
- C++ >> [answers_cpp/answer_41.cpp](#)

Q.42. Cannyエッジ検出 (Step.2) 細線化

ここでは3を実装する。

Q.41で求めた勾配角度から、Non-maximum suppressionを行い、エッジ線を細くする（細線化）。

Non-maximum suppression(NMS)とは非最大値以外を除去する作業の総称である。（他のタスクでもこの名前はよく出る）

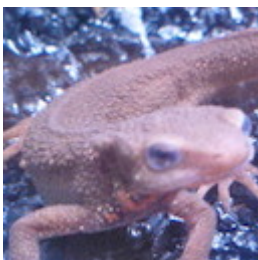
ここでは、注目している箇所の勾配角度の法線方向の隣接ピクセルの3つの勾配強度を比較して、最大値ならそのまま値をいじらずに、最大値でなければ強度を0にする、

つまり、勾配強度 $edge(x,y)$ に注目している際に、勾配角度 $angle(x,y)$ によって次式のように $edge(x,y)$ を変更する。

```
if angle(x,y) = 0
  if edge(x,y), edge(x-1,y), edge(x+1,y)で edge(x,y)が最大じゃない
    then edge(x,y) = 0
if angle(x,y) = 45
  if edge(x,y), edge(x-1,y+1), edge(x+1,y-1)で edge(x,y)が最大じゃない
    then edge(x,y) = 0
if angle(x,y) = 90
  if edge(x,y), edge(x,y-1), edge(x,y+1)で edge(x,y)が最大じゃない
    then edge(x,y) = 0
if angle(x,y) = 135
  if edge(x,y), edge(x-1,y-1), edge(x+1,y+1)で edge(x,y)が最大じゃない
    then edge(x,y) = 0
```

入力 (imori.jpg)

出力 (answers_image/answer_42.jpg)



答え

- Python >> [answers_py/answer_42.py](#)
- Python >> [answers_cpp/answer_42.cpp](#)

Q.43. Cannyエッジ検出 (Step.3) ヒステリシス閾処理

ここでは4を実装する。これがCanny法の最後である。

ここでは、閾値により勾配強度の二値化を行うがCanny法では二つの閾値(HT: high thresholdとLT: low threshold)を用いる。

はじめに、

1. 勾配強度 $\text{edge}(x,y)$ がHT以上の場合は $\text{edge}(x,y)=255$
2. LT以下の $\text{edge}(x,y)=0$
3. $\text{LT} < \text{edge}(x,y) < \text{HT}$ の時、周り8ピクセルの勾配強度でHTより大きい値が存在すれば、 $\text{edge}(x,y)=255$

ここでは、HT=50, LT=20とする。ちなみに閾値の値は結果を見ながら判断するしかない。

以上のアルゴリズムによって、Canny法が行われる。

入力 (imori.jpg) 出力 (answers_image/answer_43.jpg)



答え

- Python >> [answers_py/answer_43.py](#)
- C++ >> [answers_cpp/answer_43.cpp](#)

Q.44. Hough変換・直線検出 (Step.1) Hough変換

Q.44 - 46 ではHough変換を用いた直線検出を行う。

Hough変換とは、座標を直交座標から極座標に変換することにより数式に沿って直線や円など一定の形状を検出する手法である。ある直線状の点では極座標に変換すると一定の r , t において交わる。その点が検出すべき直線を表すパラメータであり、このパラメータを逆変換すると直線の方程式を求めることができる。

方法としては、

1. エッジ画像(ここではCannyの出力)からエッジのピクセルにおいてHough変換を行う。
2. Hough変換後の値のヒストグラムをとり、極大点を選ぶ。
3. 極大点の r , t の値をHough逆変換して検出した直線のパラメータを得る。

となる。

ここでは、1のHough変換を行いヒストグラムを作成する。

アルゴリズムは、

1. 画像の対角線の長さ r_{\max} を求める。
2. エッジ箇所 (x,y) において、 $t = 0-179$ で一度ずつ t を変えながら、次式によりHough変換を行う。

$$\rho = x * \cos(t) + y * \sin(t)$$

3. 「ボーディング（投票）」 $[2 \times r_{\max}, 180]$ のサイズの表を用意し、1で得た $\text{table}(\rho, t)$ に1を足す。2で求めた ρ は $[-r_{\max}, r_{\max}]$ の範囲を取るので、ボーディングの時には注意！

ボーディングでは、一定の箇所集中する。

今回は *torino.jpg* を用いて、ボーディングした表を図示せよ。Cannyのパラメータは, gaussian filter (5x5, $s=1.4$), $HT = 100$, $LT = 30$ で使用せよ。

入力 (thorino.jpg)

出力 (answers_image/answer_44.jpg)



答え

- Python >> [answers_py/answer_44.py](#)
- C++ >> [answers_cpp/answer_44.cpp](#)

Q.45. Hough変換・直線検出 (Step.2) NMS

ここでは2を実装する。

Q.44で得られた表では、ある一定の箇所付近に多く投票される。ここでは、その付近の極大値を抜き出す操作を行え。

今回はボーディングが多い箇所を上位20個抜き出し、図示せよ。(C++の解答は上位30個にしています。なんか20だと同じような結果にらなかったなので、)

NMSのアルゴリズムは、

1. 表において、周囲8マス(8近傍)より注目ピクセルの得票数が多ければそのまま。
2. 注目ピクセルの値が少なければ0にする。

入力 (thorino.jpg)

出力 (answers_image/answer_45.jpg)

入力 (thorino.jpg)

出力 (answers_image/answer_45.jpg)



答え

- Python >> [answers_py/answer_45.py](#)
- C++ >> [answers_cpp/answer_45.cpp](#)

Q.46. Hough変換・直線検出 (Step.3) Hough逆変換

ここではQ.45.で得られた極大値をHough逆変換をして直線を描画する。これで、Hough変換による直線検出が完了する。

アルゴリズムは、

1. 極大点(r, t)を次式で逆変換する。

$$\begin{aligned} y &= -\cos(t) / \sin(t) * x + r / \sin(t) \\ x &= -\sin(t) / \cos(t) * y + r / \cos(t) \end{aligned}$$

2. 1の逆変換を極大点ごとに $y = 0 - H-1$, $x = 0 - W-1$ で行い、入力画像に検出した直線を描画せよ。ただし、描画するのは赤線(R,G,B) = (255, 0, 0)とする。

入力 (thorino.jpg)

出力 (answers_image/answer_46.jpg)



答え

- Python >> [answers_py/answer_46.py](#)
- C++ >> [answers_cpp/answer_46.cpp](#)

Q.47. モルフォロジー処理(膨張)

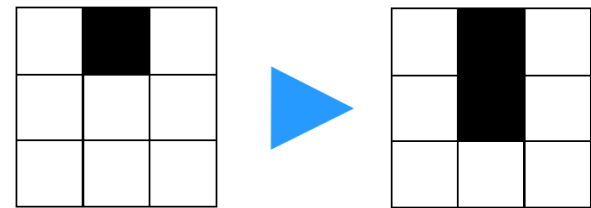
Morphology Dilate

*imori.jpg*を大津の二値化したものに、モルフォロジー処理による膨張を2回行え。

モルフォロジー処理とは二値化画像の白(255)マス部分を4近傍(上下左右1マス)に膨張、または1マスだけ収縮させる処理をいう。

この膨張と収縮を何度も繰り返すことで1マスだけに存在する白マスを消したり(Q.49. オープニング処理)、本来つながってほしい白マスを結合させたりできる(Q.50. クロージング処理)。

モルフォロジー処理の膨張(Dilation)アルゴリズムは、注目画素 $I(x, y)=0$ で、 $I(x, y-1)$, $I(x-1, y)$, $I(x+1, y)$, $I(x, y+1)$ のどれか一つが255なら、 $I(x, y) = 255$ とする。



つまり、上の処理を2回行えば2マス分膨張できることになる。

モルフォロジー処理の実装は例えば、 $[[0,1,0], [1,0,1], [0,1,0]]$ のフィルタを掛けた和が255以上なら膨張である、と考えることもできる。

入力 (imori.jpg)	大津の二値化 (answers_image/answer_4.jpg)	出力 (answers_image/answer_47.jpg)

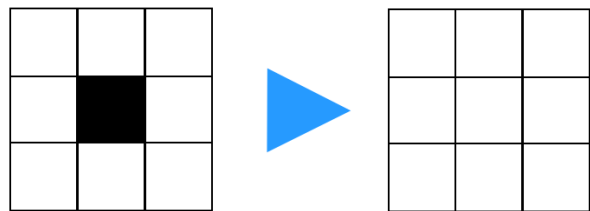
答え

- Python >> [answers_py/answer_47.py](#)
- C++ >> [answers_cpp/answer_47.cpp](#)


Q.48. モルフォロジー処理(収縮)

*imori.jpg*を大津の二値化したものに、モルフォロジー処理による収縮を2回行え。

モルフォロジー処理の収縮(Erosion)アルゴリズムは、注目画素 $I(x, y)=255$ で、 $I(x, y-1)$, $I(x-1, y)$, $I(x+1, y)$, $I(x, y+1)$ のどれか一つでも0なら、 $I(x, y) = 0$ とする。



収縮処理は例えば、 $[[0,1,0], [1,0,1], [0,1,0]]$ のフィルタを掛けた和が255*4未満なら収縮である、と考えることもできる。

入力 (imori.jpg)	大津の二値化 (answers_image/answer_4.jpg)	出力 (answers_image/answer_48.jpg)
		

答え

- Python >> [answers_py/answer_48.py](#)
- C++ >> [answers_cpp/answer_48.cpp](#)


Q.49. オープニング処理

大津の二値化後に、オープニング処理(N=1)を行え。

オープニング処理とは、モルフォロジー処理の収縮をN回行った後に膨張をN回行う処理である。



オープニング処理により、一つだけ余分に存在する画素などを削除できる。

入力 (imori.jpg)	大津の二値化 (answers_image/answer_4.jpg)	出力 (answers_image/answer_49.jpg)
		

答え

- Python >> [answers_py/answer_49.py](#)
- C++ >> [answers_cpp/answer_49.cpp](#)

Q.50. クロージング処理

Canny検出した後に、クロージング処理(N=1)を行え。

クロージング処理とは、モルフォロジー処理の膨張をN回行った後に収縮をN回行う処理である。

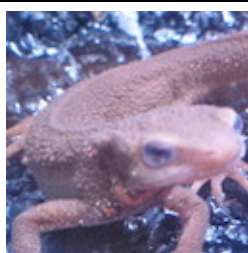


クロージング処理により、途中で途切れた画素を結合することができる。

入力 (imori.jpg)

Canny(answers_image/answer_43.jpg)

出力
(answers_image/answer_50.jpg)



答え

- Python >> [answers_py/answer_50.py](#)
- C++ >> [answers_cpp/answer_50.cpp](#)