マルチメディア信号処理
第１回レポート

濱崎 直紀　　永井 智之　　中尾 文亮

令和元年６月５日

1　問題

一部重なりのある 2 枚の画像を，大きさ・位置・角度を正しく合わせてつなげるプログラムの作成.

1.1　課題内容

与えられた一部重なりのある 2 枚の画像を image1,image2 とした場合に，image1 を基準に，2 枚の画像が重なるように image2 の水平・垂直方向の移動量，回転角および拡大率を算出するプログラムを作成する．ただし，以下の条件があるものとする.

- 画像の大きさは幅 640 以下，高さ 480 以下
- 水平・垂直方向の移動量は画像の幅，高さを超えない
- 回転角は時計回り，反時計回りともに 15 度未満
- 拡大倍率は 0.5 以上 2.0 以下

1.2　補足

実際のプログラムに関しては，レポート末尾に付録として記載.

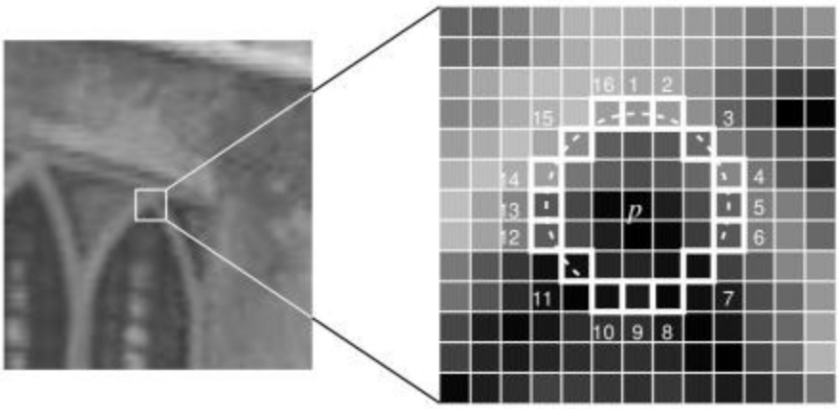
2　アルゴリズム（FAST）の概要

私たちの班は，問題解決のアルゴリズムとして FAST を用いた．FAST の概要を以下に示す.

2.1　原理

FAST による特徴量抽出の仕組みは以下のようなものである。

1. あるピクセルにおける輝度値を I_p とする.
2. 適当な閾値 t を選ぶ.
3. 周囲の 16 ピクセルに置いて，その輝度値が $I_p - t$ より小さいか，$I_p + t$ より大きいか，そのどちらでもないかを判定する．輝度値が $I_p - t$ より小さいピクセルか，$I_p + t$ より大きいピクセルがある数 n 以上の場合，その点を特徴点（corner）として抽出する.



2.2 実装

- FAST の実装

  前節で述べた FAST の仕組みを以下のように実装した．なお，$t = 50$，$n = 12$ とした．n の値は，FAST における高速化プログラムを意識した設定であるが，実験の結果，高速化のプログラムを織り交ぜた場合，ほとんどの試行に置いて，0.2s ほどの遅れが見られたため，今回は高速化のプログラムを導入していない．

  1. 画像をグレースケールに変換する．
  2. あるピクセルにおいて，その輝度値を I_p とする．また，周囲 16 ピクセルを抜き出し，その輝度値を配列に保存する．
  3. 2 で抽出した 16 ピクセルについて，$I_p - t$ より輝度が小さい，もしくは $I_p + t$ より輝度が大きいピクセルの連続数を配列に格納する．ただし，最初と最後のピクセルで輝度条件が一致する場合は，最初の連続数と最後の連続数を足し合わせ，配列の末尾に格納する．
  4. 3 で作った配列の要素の最大値を抽出し，その値が n 以上であった場合，特徴点として認め，その座標を記録する．
  5. 2〜5 を画像中の「ほぼ」全てのピクセルに置いて繰り返す．（画像の端から 10 ピクセルに置いては処理をスキップする．これは後述するマッチング処理に置いて，エラーを防ぐためである．）

- 特徴点マッチングの実装

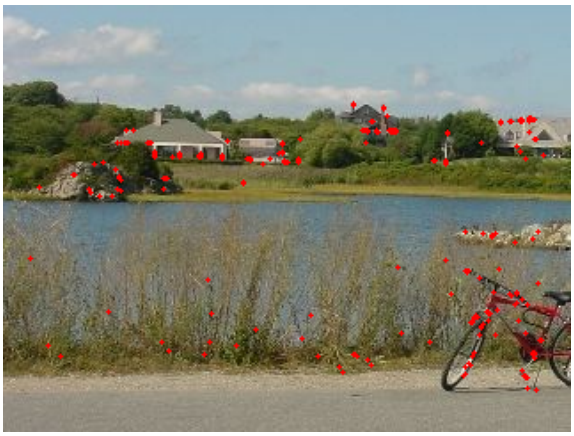
  特徴点のマッチングは以下のように行った．

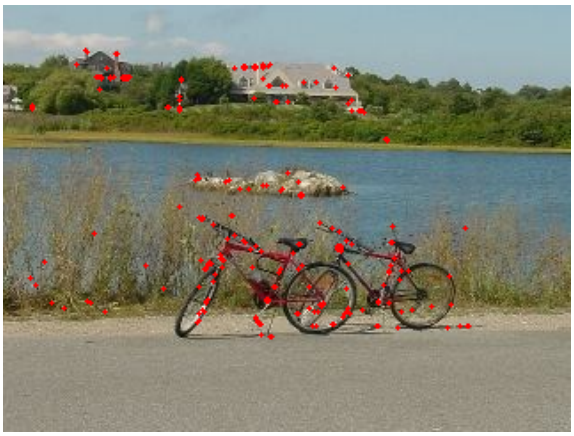
  1. 2 つの画像について，特徴点を上記の FAST の実装を用いて特徴点抽出を行う．
  2. 各画像の特徴点周辺の 10 × 10 ピクセルを抜きだし，1 次元化したのちに差分の絶対値を足し合わせる．これを「特徴点非類似度」として記録する．すなわち，この値が大きいほど特徴点は類似していないことになる．
  3. これを全ての特徴点組みに対して行い，「特徴点非類似度」がもっとも小さい 2 組をもっとも信頼のおける 2 組の特徴点対とし，変換行列を計算する．

3　結果と課題

3.1　結果

特徴量抽出を行なった 2 画像，および作成したパノラマ画像の一例を以下に示す．





また，レベル 1 のデータセット 10 セットにおけるパノラマ画像作成の成否，および処理時間は以下のようになった．

Target	Result(default)	Result(FAST)	Time(default)	Time(FAST)
1-001	○	○	35.56	12.77
1-002	×	○	292.36	53.93
1-003	×	○	3.61	3.52
1-004	○	○	215.18	47.27
1-005	×	○	58.56	26.96
1-006	×	○	20.92	16.25
1-007	×	○	295.62	55.06
1-008	×	×	3.78	3.47
1-009	○	○	183.84	46.63
1-010	×	×	60.60	24.80

処理に使用した PC は”MacBook Pro(Core i5，メモリ 8GB)”である．

3.2 課題

- 正答率の向上に関しての課題

  前節で示したように，Level1 では 8, 10 のデータセットで正しくパノラマを作成できなかった．これは，重なるべき場所に置いて特徴点が抽出されていなかったためである．これは，t および n の調節によって改善できると考えられる．

- 回転，およびスケールへの対応の課題

  本課題で作ったプログラムは，回転およびスケールには正しく対応できない．これは特徴点抽出のフェーズに置いて，隣接する 2 点を抽出してしまうために，マッチングの際にもっとも信頼のおける特徴点 2 対が隣接するピクセルになり，回転，拡大が正しく計算できない可能性があるためである．(そもそも変換行列を計算する関数は現状本プログラムには組み込まれていない．) これを解決するためには，特徴点抽出のフェーズで隣接する特徴点が存在した場合，特徴量の小さい方の特徴点を「間引く」非最大値抑制と言われる処理を行う必要がある．

4 感想

- 濱崎 直紀（馬場口研究室）
  担当：プログラミング（サンプルプログラム），レポート作成

  研究の一環などで画像から特徴などを抽出することはあったが，基本的に OpenCV などの画像処理系ライブラリを用いていたため，実際にその内部で行われていることをしっかり理解していなかった．しかし，今回の課題があったことで，数多くある手法の一つではあるが，FAST の内部で行われていることの理解を深めることができた．ライブラリの存在により実際に行われていることが曖昧になりがちだったが，そのアルゴリズムを知ることの重要さを感じた．

  また，FAST のプログラムを実際に書くことに関してはほぼ貢献できなかったという反省点があるの

で，次回の課題の際にはグループの一員としての役割をしっかりと果たせるように努めたい．

- 永井 智之（井上研究室）
  担当：プログラミング（ガウスノイズの実装）

  ガウスノイズの実装を行なったが，FAST の方での実装は行なっておらず，何か結果を示すことは出来なかった．問題点として，目標をしっかりと定めずに課題を行なっていたことが挙げられる．
  また，FAST の方は中尾くんに任せっきりだったので，次回の課題では班に貢献できるようにしたい．

- 中尾 文亮（八木研究室）
  担当：プログラミング（FAST の実装）

  今回 FAST プログラムを実装したが，高速化のプログラムを省略してしまったり，周囲のピクセルとの輝度比較のプログラムで冗長な書き方をしてしまうなど，主に「処理速度」の点で妥協ののこる実装になってしまった．次回の課題では，ただ動けばいいということではなく，速さも求めたプログラムを書きたい．

付録 A　プログラム 1

ソースコード 1　FAST

```
1   import cv2
2   import numpy as np
3
4   class FAST:
5
6     def __init__(self, t, n):
7       self.t = t
8       self.n = n
9
10    # カラー画像を引数として、FAST で抽出した特徴点座標をタプルの配列で返す
11    def get_img_feature(self, img):
12
13      img_feature= []
14
15      img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
16      height, width = img.shape
17
18      for x in range(width):
19        for y in range(height):
20
21          #上下左右 15ピクセルは処理しない(接合時のことを考慮)
22          if (
23               x in range(0,15)
24            or x in range(width-15,width)
25            or y in range(0,15)
26            or y in range(height-15,height)
27            ):
28              continue
29
30          standard_brightness = img[y][x] #基準となる明るさ
31
32          surrounding_pixels = np.array(
33            [
34              int(img[y-3][x]),
35              int(img[y-3][x+1]),
36              int(img[y-2][x+2]),
37              int(img[y-1][x+3]),
38              int(img[y][x+3]),
39              int(img[y+1][x+3]),
40              int(img[y+2][x+2]),
```

```
            int(img[y+3][x+1]),
            int(img[y+3][x]),
            int(img[y+3][x-1]),
            int(img[y+2][x-2]),
            int(img[y+1][x-3]),
            int(img[y][x-3]),
            int(img[y-1][x-3]),
            int(img[y-2][x-2]),
            int(img[y-3][x-1])
          ]
        )

        # #現状これがないほうが早く動くんだよなぁ..
        # #高速化プログラム（本来n>12の時適用)#############################################################
        # brightness_status_array = [0,0]

        # for i in [0,4,8,12]:
        # brightness_status = self.__get_brightness_status(brightness_around_array[i], brightness, self.t)
        # if brightness_status == "bright":
        # brightness_status_array[0] += 1
        # elif brightness_status == "dark":
        # brightness_status_array[1] += 1

        # #明るい点が2点の場合break
        # if not (brightness_status_array[0]>2 or brightness_status_array[1]>2):
        # continue
        # ##########################################################################################

        consecutive_num_array = [] #明るさ判断が連続して一致した数の格納用
        consecutive_num_el = 0 #明るさ判断が連続して一致した数の一時保存用
        brightness_status = 0
        brightness_status_before = 0

        #0部分のbrightness statusをここで定義
        brightness_status_before = self.__get_brightness_status(surrounding_pixels[0], standard_brightness)

        for i in range(16):

          #0部分のbrightness statusをここで定義
          brightness_status = self.__get_brightness_status(surrounding_pixels[i], standard_brightness)
```

```
81
82              #前の要素の明るさが中間値でなく、かつ前の要素と一致する場合
83              if brightness_status != 0 and brightness_status_before ==
                    brightness_status:
84
85                consecutive_num_el += 1
86
87              else: #前の要素と明るさに関する要件が一致しない
88
89                consecutive_num_array.append(consecutive_num_el)
90                consecutive_num_el = 1
91
92              #brightness_status の更新
93              brightness_status_before = brightness_status
94
95            #最終結果および、最初と最後の足し合わせを末尾に追加
96            consecutive_num_array.append(consecutive_num_el)
97            consecutive_num_array.append( consecutive_num_array[0] +
                  consecutive_num_array[-1] )
98
99            if np.amax(consecutive_num_array) > self.n:
100             img_feature.append((x,y))
101
102       print("feature detect process finished")
103       return img_feature
104
105
106   #２画像とその特徴点からもっともマッチした組み合わせを返す
107   #(この関数は回転対応のためにちょっと書き換える必要あり)
108   def get_best_match_feature(self, img1, img1_features, img2, img2_features):
109
110     img1_best_feature = (0,0)
111     img2_best_feature = (0,0)
112     min_diff = 1000000000
113
114     for feature1 in img1_features:
115
116       x1,y1 = feature1
117
118       for feature2 in img2_features:
119
120         x2,y2 = feature2
121
122         img1_fraction = img1[y1-10:y1+10, x1-10:x1+10].astype(np.int8).flatten()
123         img2_fraction = img2[y2-10:y2+10, x2-10:x2+10].astype(np.int8).flatten()
124
```

```
125             diff = np.sum(np.abs(img1_fraction-img2_fraction))
126
127             if min_diff > diff:
128                 min_diff = diff
129                 img1_best_feature = (x1,y1)
130                 img2_best_feature = (x2,y2)
131
132       return (img1_best_feature, img2_best_feature)
133
134     #brightness_status 取得部分
135     def __get_brightness_status (self, target, brightness):
136       if target > brightness+self.t:
137         return 1
138       elif target < brightness-self.t:
139         return -1
140       else:
141         return 0
```

付録 B　プログラム 2

ソースコード 2　FAST のテスト用プログラム

```
1  import cv2
2  import numpy as np
3  import fast
4  import time
5  import process_img
6
7  #測定開始
8  start = time.time()
9
10 #画像読み込み
11 img1 = cv2.imread('img/1-001-1.jpg')
12 img2 = cv2.imread('img/1-001-2.jpg')
13
14 #各種変数値
15 t = 50 #明るさの閾値
16 n = 12 #周囲の点 16点のうち明るさ条件が連続する点の数
17
18 #各画像の特徴量の格納用array
19 img1_feature = []
20 img2_feature = []
21
22 fast = fast.FAST(t,n)
23 img1_feature = fast.get_img_feature(img1)
24 img2_feature = fast.get_img_feature(img2)
25
26 best_feature1, best_feature2 = fast.get_best_match_feature(img1, img1_feature, img2,
       img2_feature)
27
28 joined_img = process_img.ProcessImg.join_img(img1, img2, best_feature1, best_feature2)
29
30 cv2.imshow("joined", joined_img)
31
32 elapsed_time = time.time() - start
33 print ("elapsed_time:{0}".format(elapsed_time) + "[sec]")
34
35
36 k = cv2.waitKey(0)
37 if k == 27:
38     cv2.destroyAllWindows()
39 elif k == ord('s'):
```

```
40      cv2.imwrite('joined.png',joined_img)
41      cv2.destroyAllWindows()
```

付録 C　プログラム 3

ソースコード 3　サブプログラム

```
1  import cv2
2  import numpy as np
3  import fast
4
5  class ProcessImg:
6
7    @classmethod
8    def join_img(self, img1, img2, img1_feature, img2_feature):
9
10     height1, width1, color1 = img1.shape
11     height2, width2, color2 = img2.shape
12
13     x1, y1 = img1_feature
14     x2, y2 = img2_feature
15
16     base_img = np.zeros((height1+height2, width1+width2, 3)).astype(np.uint8)
17     height,width,color = base_img.shape
18     half_width = int(width/2)
19     half_height = int(height/2)
20     base_img[half_height-y1:half_height+(height1-y1),half_width-x1:half_width+(width1-x1
           )] = img1
21     base_img[half_height-y2:half_height+(height1-y2),half_width-x2:half_width+(width1-x2
           )] = img2
22
23     return base_img
24
25   @classmethod
26   def rotate_img(self, img, angle):
27     rotate_matrix = np.array([[np.cos(angle), -np.sin(angle), 0],
28                     [np.sin(angle), np.cos(angle), 0],
29                     [ 0, 0, 1]])
30
31     H, W, color = img.shape
32     WID = int(np.max(img.shape) * 2**0.5)
33     e_img = np.zeros((WID, WID, 3)).astype(np.uint8)
34     e_img[int((WID-H)/2):int((WID+H)/2),
35           int((WID-W)/2):int((WID+W)/2)] = img
36     x = np.tile(np.linspace(-1, 1, WID).reshape(1, -1), (WID, 1))
37     y = np.tile(np.linspace(-1, 1, WID).reshape(-1, 1), (1, WID))
38     p = np.array([[x, y, np.ones(x.shape)]])
```

```
    dx, dy, _ = np.sum(p * rotate_matrix.reshape(3, 3, 1, 1), axis=1)
    u = np.clip((dx + 1) * WID / 2, 0, WID-1).astype('i')
    v = np.clip((dy + 1) * WID / 2, 0, WID-1).astype('i')
    return e_img[v, u]


  @classmethod
  def plot_img_feature(self, img, features):

    for feature in features:
      cv2.circle(img, feature, 1, (0, 0, 255), thickness=-1)

    return img
```

付録 D　プログラム 4

ソースコード 4　サンプルプログラム（Python）

```
import cv2
import math
import numpy as np
import os
import time
os.chdir(os.path.dirname(os.path.abspath(__file__)))


def mosaicGetMin(image3, image4, xmin, xmax, ymin, ymax, tmin, tmax, tstep, width1, height1, width2, height2):
    min = 1.7976931348623157e+308

    t = tmin
    while t < tmax:
        s1 = math.sin(t * math.pi / 180.0)
        c1 = math.cos(t * math.pi / 180.0)
        y = ymin
        while y <= ymax:
            x = xmin
            while x <= xmax:
                print('x:{} y:{} t:{}'.format(x, y, t))
                s = 0
                count = 0
                i = 0
                while i < height2:
                    j = 0
                    while j < width2:
                        # 画像を回転したときのピクセルの位置
                        u = int(math.floor((s1 * i + c1 * j) + x + 0.5))
                        v = int(math.floor((c1 * i - s1 * j) + y + 0.5))
                        # 画像が重ならない部分は誤差を計算しない
                        if u < 0 or u >= width1 or v < 0 or v >= height1:
                            j += 1
                            continue
                        pixelValue1 = image3[v, u]
                        pixelValue2 = image4[i, j]
                        # 二乗誤差の計算
                        tmp = int(pixelValue1[0]) - int(pixelValue2[0])
                        s += tmp * tmp
                        tmp = int(pixelValue1[1]) - int(pixelValue2[1])
```

```
                        s += tmp * tmp
                        tmp = int(pixelValue1[2]) - int(pixelValue2[2])
                        s += tmp * tmp
                        count += 1

                        j += 1
                    i += 1

                # 平均二乗誤差の計算
                ave = s / count
                # 平均二乗誤差が最小値より小さい場合はパラメータの更新
                if min > ave:
                    min = ave
                    i_min = y
                    j_min = x
                    t_min = t

                x += 1
            y += 1
        t += tstep

    return i_min, j_min, t_min


def mosaicResizeImage(image, re_w, re_h):
    h, w, c = image.shape

    re_image = np.arange(re_w * re_h * 3).reshape(re_h, re_w, 3)

    # 元の画像と新しい画像の大きさの比を計算
    stepx = int(w / re_w)
    stepy = int(h / re_h)

    i = 0
    while i < re_h:
        j = 0
        while j < re_w:
            b = g = r = 0
            count = 0
            k = stepy * i
            while k <= stepy * (i + 1) - 1:
                l = stepx * j
                while l <= stepx * (j + 1) - 1:
                    if k >= h or l >= w:
                        l += 1
                        continue
```

```
                    pixelValue = image[k, l]
                    b += pixelValue[0]
                    g += pixelValue[1]
                    r += pixelValue[2]
                    count += 1
                    l += 1
                k += 1
            re_image[i, j][0] = b / count
            re_image[i, j][1] = g / count
            re_image[i, j][2] = r / count
            j += 1
        i += 1

    return re_image


def mosaic(image1, image2):
    # 入力画像の幅と高さを抽出
    h1, w1, c1 = image1.shape
    h2, w2, c2 = image2.shape

    # 1/5に縮小した画像を作成
    image3 = mosaicResizeImage(image1, int(w1/5), int(h1/5))
    image4 = mosaicResizeImage(image2, int(w2/5), int(h2/5))

    # 縮小画像の幅と高さを抽出
    h3, w3, c3 = image3.shape
    h4, w4, c4 = image4.shape

    # 縮小した画像で、画像が最も重なるときのパラメータを概算
    tempy, tempx, tempt = mosaicGetMin(image3, image4, 2, w3-3, 2, h3-3, -15, 15,
        2.0, w3, h3, w4, h4)
    # 元の画像でパラメータを計算
    tempy, tempx, tempt = mosaicGetMin(image1, image2, (tempx-1)*5, (tempx+1)*5-1, (
        tempy-1)*5, (tempy+1)*5-1, tempt-1.0, tempt+1.0, 1.0, w1, h1, w2, h2)

    return tempx, tempy, tempt


if __name__ == '__main__':
    filename1 = 'Level1/1-001-1.jpg'
    filename2 = 'Level1/1-001-2.jpg'

    image1 = cv2.imread(filename1)
    image2 = cv2.imread(filename2)

```

```
130     start = time.time()
131
132     dx, dy, dt = mosaic(image1, image2)
133
134     elapsed_time = time.time() - start
135
136     print('dx={}'.format(dx))
137     print('dy={}'.format(dy))
138     print('dt={}'.format(dt))
139
140     print('処理時間: {}'.format(elapsed_time))
```