第十周报告

1. 实验概览

本实验旨在理解并实现 SIFT 算法，理解 SIFT 特征子具有不变性、独特性等诸多良好特征的原因，学习 SIFT 算法的主要步骤和具体原理。在实验中自行实现 SIFT 算法并和 opencv 提供的算法进行比较。

2. 实验环境

Docker: sjtumic/ee208

3. 解决思路

a) 图像预处理

- 灰度图读取：利用 opencv 自带 cv2.IMREAD_GRAYSCALE

```
tg = cv2.imread("./target.jpg", cv2.IMREAD_GRAYSCALE)
tg0 = cv2.imread("./target.jpg")
img_lst = [
    cv2.imread("./dataset/" + str(i + 1) + ".jpg", cv2.IMREAD_GRAYSCALE) for i in range(5)
]
img_lst0 = [cv2.imread("./dataset/" + str(i + 1) + ".jpg") for i in range(5)]
```

- Resize 目标图像：基于给定的参数对目标图像进行缩放，缩放参数将进行实验获得

```
resize_cof = 1
shape = np.shape(tg)
tg = cv2.resize(tg, (int(shape[0] * resize_cof), int(shape[1] * resize_cof)))
```

- 基于 Harris 角点提取获得特征点

```
self.corners = [
    [int(i[0][0]), int(i[0][1])]
    for i in cv2.goodFeaturesToTrack(img, 300, 0.01, 10)
]
```

- 高斯滤波

```
img = cv2.GaussianBlur(img, (3, 3), 1, 1)
img = np.array(img, dtype="float")
```

b) 获取梯度和特征点方向

- 基于索贝尔核对图像进行卷积梯度提取

```
kernel = (
    np.array([[[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]],
              [[-1, -2, -1], [0, 0, 0], [1, 2, 1]]], dtype="float")
)

gx = cv2.filter2D(img, -1, np.array(kernel[1]))
gy = cv2.filter2D(img, -1, np.array(kernel[0]))
```

- 获得各点的具体梯度值并计算得到该点的梯度方向

```
for i in range(w):
    for j in range(h):
        gradient[i][j] = ((gx[i][j]) ** 2 + (gy[i][j]) ** 2) ** 0.5
        angle[i][j] = atan2(gy[i][j], gx[i][j])
return gradient, angle
```

c) 投票得到关键点主方向

对图像内角点周围各个像素点进行遍历，计算所属 bin 值，然后 gradient 为权重为其所在的 bin 投票；将票数最高的 bin 来作为该关键点的主方向（代码中总共分为 36 个方向）

```
def vote_for_direction(self):
    direction = []
    for corner in self.corners:
        y, x = corner
        to_vote = [0 for _ in range(36)]
        for i in range(max(x - self.bins, 0), min(x + self.bins + 1, self.height)):
            for j in range(max(y - self.bins, 0), min(y + self.bins + 1, self.height)):
                k = int((self.angle[i][j] + pi) / pi * 180 / 36)
                if k >= 36:
                    k = 35
                to_vote[k] += self.gradient[i][j]

        p = to_vote.index(max(to_vote[2:]))
        direction.append((p / 18) * pi)

    return direction
```

d) 计算获得特征值

- 分块计算特征

  物体坐标系 16*16 的邻域分成 4*4 个块，每个块 4*4 个像素。

  在每个块内按照求主方向的方式把 360 度分成 8 个 bins，统计梯度方向直方图，最终每个块可生成 8 维的直方图向量，每个关键点可生成 4*4*8=128 维的 SIFT 描述子。

  - 双线性插值

```
def bilinear_interpolation(x, y): # 双线性插值
    def angle(x, y):
        if (x < 0 or x >= self.height) or (y < 0 or y >= self.height):
            return 0
        dif = self.angle[x][y] - theta
        return dif if dif > 0 else dif + 2 * pi

    xx, yy = int(x), int(y)
    dy1, dy2 = y - yy, yy + 1 - y
    dx1, dx2 = x - xx, xx + 1 - x
    res = (
        angle(xx, yy) * dx2 * dy2
        + angle(xx + 1, yy) * dx1 * dy2
        + angle(xx, yy + 1) * dx2 * dy1
        + angle(xx + 1, yy + 1) * dx1 * dy1
    )
    return res
```

  - 对每个 bin 投票方式

```
def count(x1, x2, y1, y2, xsign, ysign, H, V):
    voting = [0 for _ in range(9)]
    for x in range(x1, x2):
        for y in range(y1, y2):
            dp = [x * xsign, y * ysign]
            p = H * dp[0] + V * dp[1]
            bin = int(
                (bilinear_interpolation(p[0] + x0, p[1] + y0)) // (pi / 4) + 1)
            if bin > 8:
                bin = 8
            voting[bin] += 1
    return voting[1:]
```

- 组合获得特征

```
bins = (self.height + self.height) // 150
for xsign in [-1, 1]:
    for ysign in [-1, 1]:
        val += vote(0, bins, 0, bins, xsign, ysign, H, V)
        val += vote(bins, bins * 2, 0, bins, xsign, ysign, H, V)
        val += vote(bins, bins * 2, bins, bins * 2, xsign, ysign, H, V)
        val += vote(0, bins, bins, bins * 2, xsign, ysign, H, V)

return val
```

e) 图像匹配

对两张图片的每个关键点的特征值进行匹配，若匹配值大于 threshold 则记录，若匹配的关键点大于 10 个则认为匹配成功

```
for id in range(len(sift_img)):
    x = []
    cnt = 0
    for i in range(length_tg):
        tmp = []
        for j in range(lengths[id]):
            sc = np.dot(np.array(feature_tg[i]), np.array(features[id][j]))
            tmp.append(sc)
        x.append([tmp.index(max(tmp)), max(tmp)])
    for a in range(len(x)):
        b, s = x[a]
        if s < threshold:
            continue
        cnt += 1
        color = (
            (random.randint(0, 255)),
            (random.randint(0, 255)),
            (random.randint(0, 255)),
        )
        cv2.line(
            imgs[id],
            tuple(corner_tg[a]),
            tuple([corners[id][b][0] + w, corners[id][b][1]]),
            color,
            1,
        )
    if cnt > 10:
        print("match" + str(id))
        img = np.array(imgs[id], dtype="uint8")
        cv2.imwrite("./match.jpg", img)
        cv2.imshow("result", img)
        cv2.waitKey(0)
```

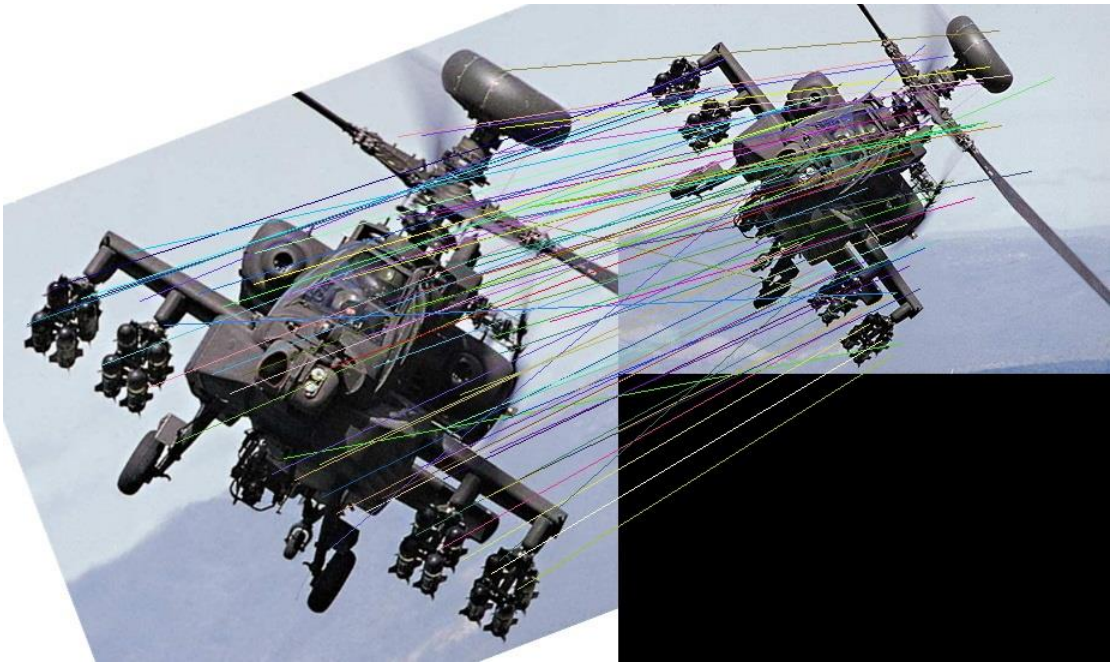
f) 对比原版库

运行原版库代码

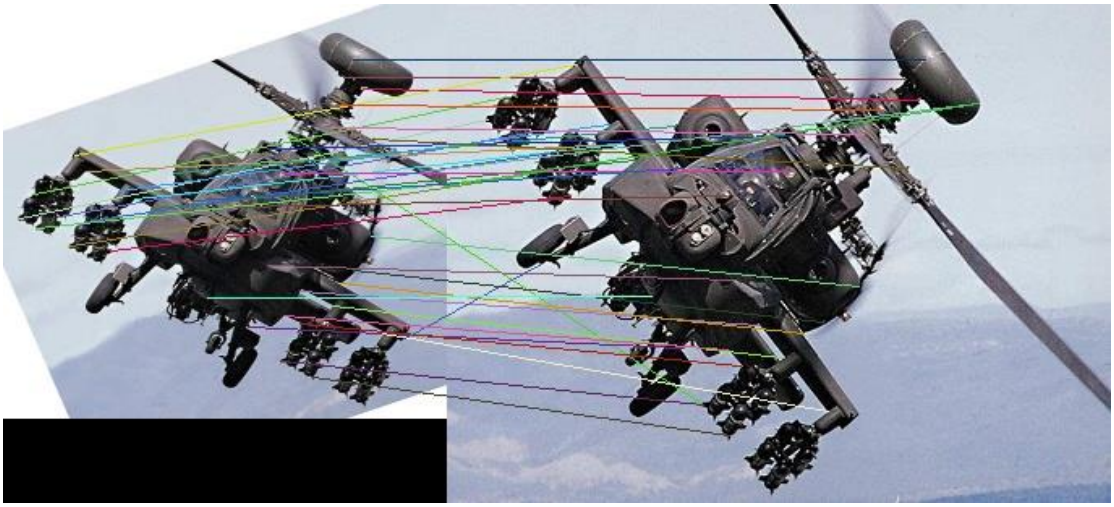
```python
import cv2

img_orign = cv2.imread('./dataset/3.jpg')
img = cv2.imread('./target.jpg')
rows, cols = img.shape[:2]
gray_orign = cv2.cvtColor(img_orign, cv2.COLOR_BGR2GRAY)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
sift = cv2.xfeatures2d.SIFT_create()

kp1, des1 = sift.detectAndCompute(gray_orign, None)
kp2, des2 = sift.detectAndCompute(gray, None)

bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)

match = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        match.append([m])
img3 = cv2.drawMatchesKnn(img_orign, kp1, img, kp2, match[:20], None, flags=2)

cv2.imshow('img', img3)
cv2.waitKey(0)
```

4. 代码运行结果

resize_cof=1，当 threshold 设置为 0.8 时可以得到正确匹配结果：

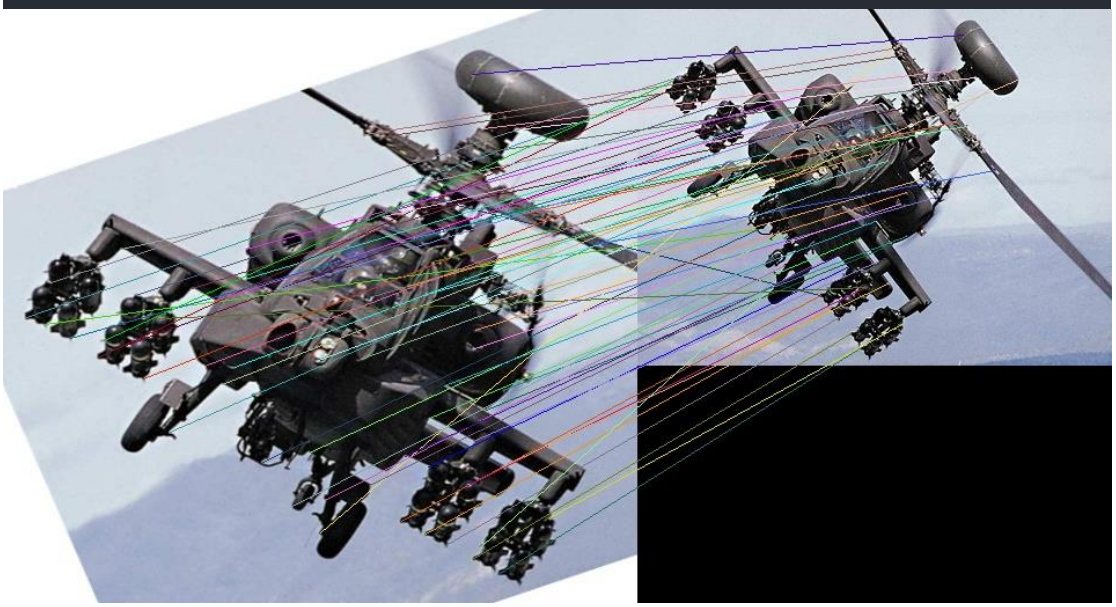


resize_cof=0.5，当 threshold 设置为 0.75 时可以得到正确匹配结果：

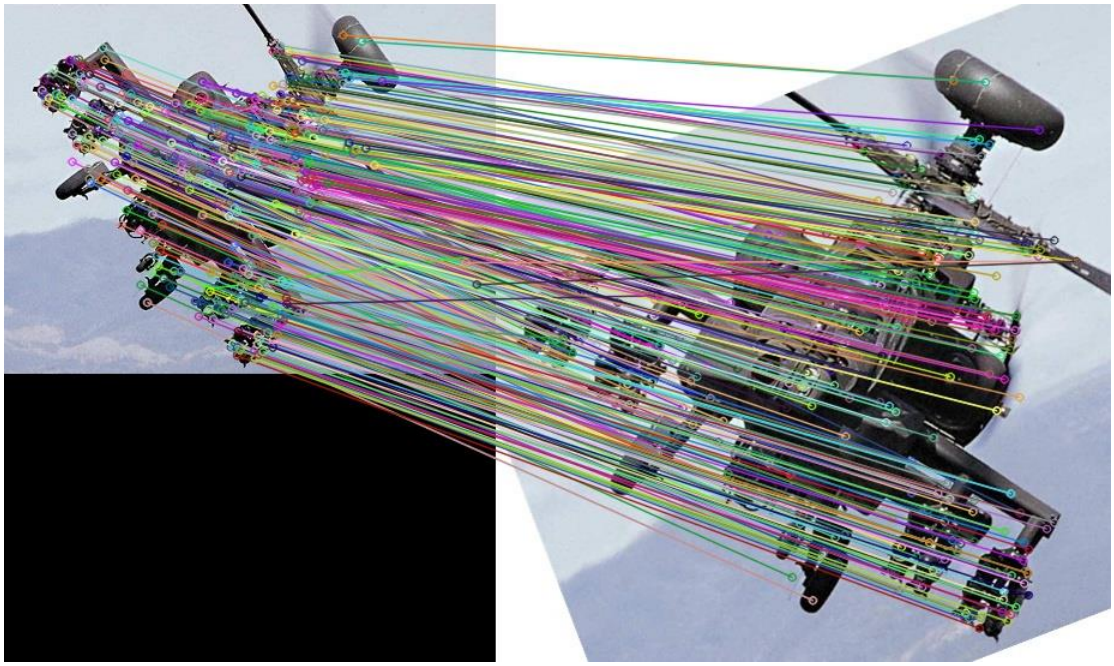


resize_cof=1，使用其他卷积核结果：

```
kernel = (
    np.array([[[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]],
              [[-1, -1, -1], [0, 0, 0], [1, 1, 1]]], dtype="float")
)
```



原版匹配结果：



5. 分析与思考

a) **Resize_cof:** 图片缩小时，匹配成功的点会相应增加，但对于关键点的匹配容易增加误匹配的可能性

b) **算子比较：**Sobel 算子和 Prewitt 算子匹配效果均可，整体而言 Sobel 算子的匹配点较多，两者误判水平近似

6. 代码结构解释

sift.py：自行搭建 Sift 算法
cv_sift.py：调用 opencv 的 Sift 算法
match.jpg：调用 sift.py 得到的匹配结果
cv_match.jpg：调用 cv_sift.py 得到的匹配结果