电工导 C 第十一次实验报告

姓名: 宋士祥　　学号:521030910013　　班级:F2103001

2022 年 12 月 8 日

1　实验概览

本次实验简单学习了 SIFT 算法的相关知识，SIFT 算法是 David Lowe 于 1999 年提出的局部特征描述子，并于 2004 年进行了更深入的发展和完善。Sift 特征匹配算法可以处理两幅图像之间发生平移、旋转、仿射变换情况下的匹配问题，具有很强的匹配能力。总体来说，SIFT 算子具有以下特性：

1. SIFT 特征是图像的局部特征，对平移、旋转、尺度缩放、亮度变化、遮挡和噪声等具有良好的不变性，对视觉变化、仿射变换也保持一定程度的稳定性。
2. 独特性好，信息量丰富，适用于在海量特征数据库中进行快速、准确的匹配。
3. 多量性，即使少数的几个物体也可以产生大量 SIFT 特征向量。
4. 速度相对较快，经优化的 Sift 匹配算法甚至可以达到实时的要求。
5. 可扩展性强，可以很方便的与其他形式的特征向量进行联合。

在`openCV` 库中，我们有一个成熟的`SIFT` 相关函数可以直接提取特征并进行比较，然而本次实验中我们会尝试自己写一个与之相仿的`SIFT` 类。

对于 SIFT 函数，本次实验分为以下步骤：

1. 进行关键点检测并定位。
2. 对图像预处理进行高斯模糊，并求梯度幅值和梯度方向
3. 对每个关键点计算主方向。
4. 计算每一个关键点的 SIFT
5. 进行描述子匹配，并再图像中绘制出图像。

2　实验环境

本次实验采用所需的实验环境如下：

- Docker 中的`sjtumic/ee208` 镜像
- Python3（使用 VSCode 编译）
- `numpy` 扩展。
- `OpenCV` 环境。(均在 SJTUEE208 镜像中给出)

3 进行关键点检测并定位

根据 ppt 中已经给出的内容，我们直接采用 Harris 角点检测的方法，并直接利用 Harris 角点的方法对特征点进行检测，并直接将特征点标记在图片上。利用 numpy 的性质，可以对图片直接进行点乘，对应的代码如下

```
def feature(self):
    gray = self.gray.copy()
    # gray = cv2.cvtColor(i, cv2.COLOR_BGR2GRAY)
    gray = np.float32(gray)
    dst = cv2.cornerHarris(gray, 2, 3, 0.1)
    dst = cv2.dilate(dst, None)
    self.img[dst > 0.01 * dst.max()] = [0, 0, 255]
    feature = list()
    feature += [dst > 0.01 * dst.max()]
    return np.array(feature[0])
```

4 对图像进行高斯模糊，并对每个点计算梯度幅值和梯度大小

我们在类的初始状态已经进行高斯模糊了，此处不再赘述。

对于图像的梯度大小和方向，我们有公式

$$m(x,y)=\sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y-1)-L(x,y+1))^2}$$

$$\theta(x,y)=\arctan\frac{L(x+1,y)-L(x-1,y)}{L(x,y-1)-L(x,y+1)}$$

需要注意的是，在 math 库中的 atan2 函数提供了分数三角函数的计算，不会出现越界的情形。同时由于图像的数值限制在 [0,255] 之间，我们需要先将其转化为 int 形式再进行处理。

我们还需要将负数化为 $(\pi,2\pi)$ 区间范围，再将其化为度数。为了简化计算，记为整数。

相关代码如下

```
# 计算梯度幅值和梯度方向
def calculte_gradient(self):
    M = np.zeros(self.gray.shape)
    T = np.zeros(self.gray.shape)
    theta_lst = list()
    for i in range(1, self.gray.shape[0] - 1):
        for j in range(1, self.gray.shape[1] - 1):
            theta = math.atan2(int(self.gray[i][j + 1]) - int(self.gray[i][j - 1]),
                               int(self.gray[i + 1][j]) - int(self.gray[i - 1][j]))
            if theta < 0:
                theta = 2 * math.pi + theta
            theta = int(theta * 360 / (2 * math.pi))
            theta_lst.append(theta)
            m = math.sqrt((int(self.gray[i][j + 1]) - int(self.gray[i][j - 1]))**2 +
                          (int(self.gray[i + 1][j]) - int(self.gray[i - 1][j]))**2)
            T[i][j] = theta
            M[i][j] = m
    return (M, T)
```

5 计算每个关键点的主方向

对于关键点的主方向，我们再以关键点为中心，半径为 $3 \times 1.5\sigma$ 的半径范围内取点。同时，考虑到越靠近中心点对主方向的贡献越大，我们采用了对其加权的方式。考虑中心点要为 1，我们的加权方式为 $\frac{1}{1+d}$。

我们将方向等分为 36 个方向，每个方向分别取一个直方图，主方向为其中的某一个方向，对应的代码如下：

```
# 计算每一个关键点的梯度方向
def calculate_direction(self):
    M, T = self.M, self.T
    feature = self.Feature
    directions = np.zeros(feature.shape)
    # print(feature.shape)
    for i in range(self.gray.shape[0]):
        for j in range(self.gray.shape[1]):
            if feature[i][j]:
                weight_lst = [0] * 36
                r = 3 * 1.5 * 1.6  #按照论文建议 σ 取1.6 r=7.2
                # 以下是求极值点
                minx = i - 7 if i - 7 > 0 else 0
                maxx = i + 7 if i + 7 < self.gray.shape[
                    0] else self.gray.shape[0]
                miny = j - 7 if j - 7 > 0 else 0
                maxy = j + 7 if j + 7 < self.gray.shape[
                    1] else self.gray.shape[1]
                for p in range(minx, maxx):
                    for q in range(miny, maxy):
                        distance = math.sqrt((p - i)**2 + (q - j)**2)
                        if (distance < r):
                            weight_lst[int(T[p, q]) // 10] += M[p, q] /(1+distance)
                dirpoint = max(weight_lst)
                direct = 0
                # subdir = list() 辅助方向#
                for k in range(36):
                    if k == dirpoint:
                        direct = k * 10
                    # if k > 0.8 * dirpoint:
                    #     subdir += [k * 10]
                directions[i, j] = direct
                # if subdir:
                #     subdirs[(i, j)] = subdir
    return directions
```

6 计算 SIFT 描述子

对于 SIFT 描述子的计算，我们选取周边的 16*16 的点阵。如果在边界我们直接置零。然后，我们将这一个区块分成 16 个 4×4 的小区间。利用 numpy 生成一个 128 维的向量。每一个区间将角度从 0-360 分为八个区间，每个区间的 m 值直接累加到该区间中。对应的代码如下：

```
def sift_feature(self):
    features = dict()
```

```
    feature = self.Feature
    M, T = self.M, self.T
    directions = self.calculate_direction()
    for i in range(self.gray.shape[0]):
        for j in range(self.gray.shape[1]):
            theta_block = np.zeros((16, 16))
            m_block = np.zeros((16, 16))
            if feature[i, j]:
                for p in range(0, 16):
                    for q in range(0, 16):
                        x = int(i + (p - 8) * math.cos(directions[i, j] /
                                                       360 * 2 * math.pi))
                        y = int(j + (q - 8) * math.cos(directions[i, j] /
                                                       360 * 2 * math.pi))
                        try:
                            m_block[p, q] = M[x, y]
                            theta_block[p, q] = T[x, y] - directions[i, j]
                            if theta_block[p, q] < 0:
                                theta_block[p, q] += 360
                        except:
                            continue
                # 生成特征向量
                ans = [[[0]*8]*4]*4
                for p in range(0,16):
                    for q in range(0,16):
                        ans[p//4][q//4][int(theta_block[p][q])//45] += m_block[p][q]
                features[(i,j)] = np.array(ans)
    return features
```

7 比较两个 SIFT 描述子

考虑 numpy 有强大的相减，我们对于两个描述子采取直接相减的方法，再直接计算其二维范数即可。对应代码

```
for index1, feature1 in features1:
    for index2, feature2 in features2:
        point = feature2 - feature1
        ans = 0
        ans = np.linalg.norm(point)
```

同时，我们考虑如何将两个图合并到一个图中。我们采用的方法为扩展图片，并将两个图片强心塞进去的方法。对应的代码如下：

```
def show(self, another):
    image = self.img
    transformed_image = another.img
  #  print(image.shape)
    #print(transformed_image.shape)
    h0,w0=image.shape[0],image.shape[1]  #cv2 读取出来的是h,w,c
    h1,w1=transformed_image.shape[0],transformed_image.shape[1]
    h=max(h0,h1)
    w=max(w0,w1)
```

```
    org_image=np.ones((h,w,3),dtype=np.uint8)*255
    trans_image=np.ones((h,w,3),dtype=np.uint8)*255

    org_image[:h0,:w0,:]=image[:,:,:]
    trans_image[:h1,:w1,:]=transformed_image[:,:,:]
    all_image = np.hstack((org_image[:,:w0,:], trans_image[:,:w1,:]))
    return all_image
```

8 选取合适的 match

对于最后匹配的选取，我们采取直接暴力遍历的方法。（因为我们将半径设置的很大所以其实特征点只有几百个），取最小值，然后采用 KNN 的思想，也即“近朱者赤近墨者黑”。比较周围的距离，如果周围的距离和自己相差很小，说明冲突，舍去。在这里我们选取的比例是 0.75.

同时，考虑到有一些点会重复的出现，我们采取匹配到一点，该点就从队列中删去的思想，可以有效减小实验带来的误差。

对于匹配比较好的点，我们会在图片上标红，便于后续的处理。

```
def comparison(self, another):
    shown = self.show(another)
    #print(shown.shape)
    features1 = self.sift_feature()
    features2 = another.sift_feature()
    img1 = self.img
    img2 = another.img
    #print(len(features1))
    feat1 = list(features1.items())
    feat2 = list(features2.items())
    dises = dict()
    for index1, feature1 in feat1:
        try:
            points = list()
            indeices = list()
            for index2, feature2 in feat2:
                point = feature2 - feature1
                distance = np.linalg.norm(point)
                points.append(distance)
                indeices.append(index2)
            distance = min(points)
            index2 = indeices[points.index(distance)]
            for i in feat2:
                if i[0] == index2:
                    feat2.remove(i)
                    break
            shown[index1] = (255,0,0)
            shown[index2[0], index2[1] + self.img.shape[1]] = (255, 0, 0)
            dises[(index1,index2)] = distance
        except:
            break
            # print(index1,index2)
    #print(match)
    match = list(dises.items())
    good_match = list()
```

```
    for i in range(len(match) - 1):
        if match[i][1] < 0.75*match[i+1][1]:
            good_match.append(match[i][0])
    print("There are {} matches!".format(len(good_match)))
    return (good_match, list(features1.keys()), list(features2.keys()))
```

9 绘制最后的图像

numpy 自带一个 line 的功能。但需要注意的是，numpy 绘制的 Line 坐标与图像的坐标是相反的。因此，我们需要注意将两个点进行反向，最终达到我们想要的效果。

对应代码

```
for i in range(1,6):
    print("This is image {}!".format(i))
    box = cv2.imread("./dataset/{}.jpg".format(i))
    test = MySIFT(box)
    test2 = MySIFT(box_in_sence)
    test.feature()
    lst = test.comparison(test2)
    good_points.append(lst)
    length.append(len(lst[0]))

index = length.index(max(length))
box = cv2.imread("./dataset/{}.jpg".format(index + 1))
test = MySIFT(box)
img = np.array(test.show(test2))
cv_kpts1 = good_points[index][1]
#cv_kpts1 = [cv2.KeyPoint(int(cv_kpts1[i][0]), int(cv_kpts1[i][1]), 1)
  # for i in range(len(cv_kpts1))]
cv_kpts2 = good_points[index][2]
#cv_kpts2 = [cv2.KeyPoint(int(cv_kpts2[i][0]), int(cv_kpts2[i][1]), 1)
   #for i in range(len(cv_kpts2))]
for i in good_points[:10][index][0]:
  img[i[0][0]][i[0][1]] = (0,0,255)
  img[i[1][0]][box.shape[1]+i[1][1]] = (0,0,255)
  cv2.line(img,(i[0][1],i[0][0]),(box.shape[1]+i[1][1],i[1][0]),(0,0,255))
img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
plt.imshow(img)
plt.savefig('result.png')
```

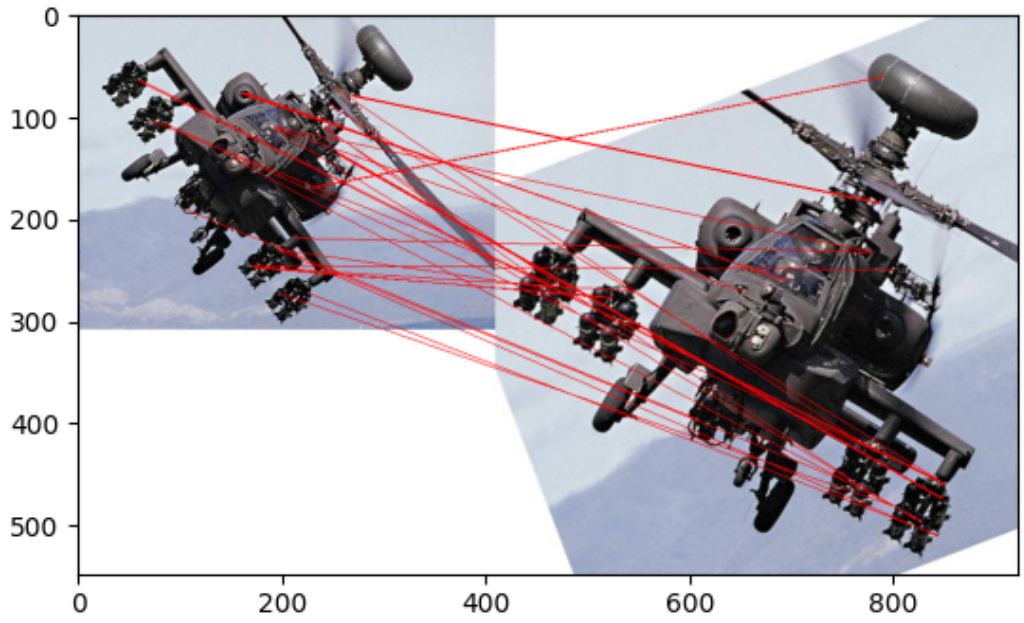
10 结果

我们采取的参数为 $\sigma = 1.6$，Harris 半径为 0.245。对应的各图像匹配结果：

```
This is image 1!
There are 3 matches!
This is image 2!
There are 5 matches!
This is image 3!
There are 33 matches!
```

```
This is image 4!
There are 4 matches!
This is image 5!
There are 8 matches!
```

匹配到图像 3 为最佳，图像如下：



匹配效果较好。

11 问题思考

11.1 为何要删除匹配到的点

笔者发现，如果不删除，很有可能造成某一点同时匹配到了多个点，同时由于我们的遍历不是按照区域遍历导致不同列上的点都保留了下来。

11.2 为何不强制暴力匹配

笔者和助教交流时得知，包括在 openCV 自带的 canny 中，我们都会默认每个特征点都会有一个匹配，这就导致会产生较大的误差，不利于最后的结果输出。

11.3 为何要进行高斯模糊

我们采取高斯模糊的主要原因是防止出现某条边的边界过于明显导致结果出现明显失真。

11.4 对于 Harris 角点检测的问题

和直接的 SIFT 算法进行比较，我们发现，Harris 角点检测产生的特征点较多，实际上也产生了很多无用的角点。采用高斯金字塔也许可以规避这一问题。

11.5 numpy 的一些问题

在完成本次任务时，我们发现，numpy 的一些数据结构和 python 的不兼容。例如图像的 int 限制在 [0,255] 之间，又例如 float32 和 python 的 float 不兼容等。

12 致谢

这次实验对我来说，难度有点大。大约花费了 4 个下午才完成了这个程序。在此感谢为我提供思路和问题解答的助教、班级同学、和室友。看来自己造轮子时真的难.....