Harris-SIFT 算法

姓　　名：陈东鑫

学　　号：1173710204

所学专业：软件工程

课程名称：数字媒体技术

提交日期：

1 harris 算法

1.1 harris 算法介绍

人眼对角点的识别通常是在一个局部的小区域或小窗口完成的。如果在各个方向上移动这个特征的小窗口，窗口内区域的灰度发生了较大的变化，那么就认为在窗口内遇到了角点。如果这个特定的窗口在图像各个方向上移动时，窗口内图像的灰度没有发生变化，那么窗口内就不存在角点；如果窗口在某一个方向移动时，窗口内图像的灰度发生了较大的变化，而在另一些方向上没有发生变化，那么，窗口内的图像可能就是一条直线的线段。
对于图像在点(x,y)处平移后的自相似性，可通过相关函数给出

$$c(x,y;\Delta x,\Delta y)=\sum_{(u,v)\in W(x,y)} w(u,v)(I(u,v)-I(u+\Delta x,v+\Delta y))^2$$

其中 W(x,y)是该点为中心的窗口， ω(u,v)为加权函数，这里使用高斯加权函数。
由泰勒展开，对该自相关函数进行简化，得

$$c(x,y;\Delta x,\Delta y)\approx\sum_w(I_x(u,v)\Delta x+I_y(u,v)\Delta y)^2=[\Delta x,\Delta y]M(x,y)\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}$$

其中

$$M(x,y)=\sum_w\begin{bmatrix}I_x(x,y)^2 & I_x(x,y)I_y(x,y)\\ I_x(x,y)I_y(x,y) & I_y(x,y)^2\end{bmatrix}=\begin{bmatrix}\sum_w I_x(x,y)^2 & \sum_w I_x(x,y)I_y(x,y)\\ \sum_w I_x(x,y)I_y(x,y) & \sum_w I_y(x,y)^2\end{bmatrix}=\begin{bmatrix}A & C\\ C & B\end{bmatrix}$$

再进一步简化为二项函数

$$c(x,y;\Delta x,\Delta y)\approx A\Delta x^2+2C\Delta x\Delta y+B\Delta y^2$$

其中

$$A=\sum_w I_x^2,B=\sum_w I_y^2,C=\sum_w I_xI_y$$

二次项函数本质上是一个椭圆函数，椭圆方程为

$$[\Delta x,\Delta y]M(x,y)\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}=1$$

计算角点响应值 R 来判断角点，R 的计算公式为

$$R=detM-\alpha(traceM)^2$$

α 取值范围为 0.04 到 0.06

1.2 harris 算法实现

计算两个方向的梯度，使用 sobel 梯度算子与图像矩阵进行卷积，计算出两个方

向的梯度 Dx 和 Dy

```
# 对整个图像求梯度
def grad(self, ksize=3):
    # 使用 Sobel 梯度算子与图像矩阵做卷积，计算两个方向梯度
    Dx = cv2.Sobel(self.grey, cv2.CV_32F, 1, 0, ksize=ksize)
    Dy = cv2.Sobel(self.grey, cv2.CV_32F, 0, 1, ksize=ksize)

    return Dx, Dy
```

进行高斯滤波。计算协方差矩阵，然后进行高斯滤波处理

```
# 高斯滤波
def gauss(self, Dx, Dy, blockSize=3):
    # 计算 cov 矩阵
    cov = np.zeros((self.x, self.y, 3), dtype=np.float32)

    for i in range(self.x):
        for j in range(self.y):
            cov[i, j, 0] = Dx[i, j] * Dx[i, j]
            cov[i, j, 1] = Dx[i, j] * Dy[i, j]
            cov[i, j, 2] = Dy[i, j] * Dy[i, j]

    # 进行高斯滤波
    cov = cv2.GaussianBlur(cov, (blockSize, blockSize), 1)

    return cov
```

计算响应值，计算公式为

$$R = detM - \alpha(traceM)^2$$

```
# 计算响应值
def getResponse(self, cov, k=0.04):
    shape0 = cov.shape[0]
    shape1 = cov.shape[1]
    res = np.zeros((shape0, shape1), dtype=np.float32)
    for i in range(shape0):
        for j in range(shape1):
            a = cov[i, j, 0]
            b = cov[i, j, 1]
            c = cov[i, j, 2]
            res[i, j] = a * c - b * b - k * (a + c) * (a + c)
```

```
12.
13.     return res
```

在一定宽度的邻域内进行非最大值抑制，局部最大值点即为图像中的角点。画出角点。

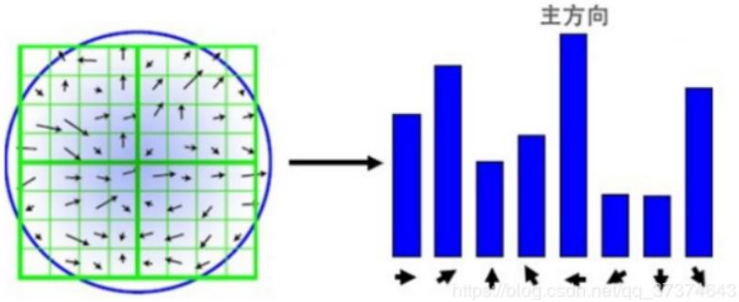
```
1.  def draw(self, response, name, maxCorners=180, qualityLevel=1e-5, minDistanc
    e=30):
2.      pos = cv2.goodFeaturesToTrack(response, maxCorners, qualityLevel, minDis
    tance)
3.      for i in range(len(pos)):
4.          cv2.circle(self.color, (pos[i][0][0], pos[i][0][1]), 5, [116, 255, 2
    38], thickness=5)
5.
6.      path = '../output/' + name
7.      cv2.imwrite(path, self.color)
8.
9.      return pos
```

1.3 harris 算法效果

2 harris-SIFT 算法

2.1 SIFT 算法的特征匹配部分

对前一步的每一个关键点，计算出该点及周围的梯度方向，每 10 度为一组，做直方图。统计梯度的主方向，为了增强匹配的鲁棒性，只保留峰值大于主方向峰值 80%的方向作为该关键点的辅方向。



关键点描述子，对每个关键点，都有位置，尺度和方向信息。为每个关键点建立一个描述符，用一组向量将这个关键点描述出来，使其不随各种变化而改变，比如光照变化、视角变化等等。这个描述子不但包括关键点，也包含关键点周围对其有贡献的像素点，并且描述符应该有较高的独特性，以便于提高特征点正确匹配的概率。描述子采用 4×4×8＝128 维向量表征，综合效果最优（不变性与独特性）.
计算主方向

```
def main_direction(self, sigma):
    # 角点集合，三维。
    # 第一维代表这是第几个角点
    # 第二维代表该角点
    # 第三维的[0]是列，[1]是行
    points = self.img['points']

    # 每个角点周围的梯度，半径为 3*sigma
    radius = int(3 * sigma)
    size = int(2 * radius + 1)

    direction = []

    for i in range(len(points)):
        # 获得该点的坐标
        row = int(points[i][0][1])
        column = int(points[i][0][0])

        # 该点半径为 radius 的所有点的范围
        row_start = int(max(row - radius, 0))
        row_end = int(min(row + radius + 1, self.img['x']))

        col_start = int(max(column - radius, 0))
        col_end = int(min(column + radius + 1, self.img['y']))

        # 如果范围超出了原图像，则记下，后续补零
        row_move = int(size - (row_end - row_start))
        col_move = int(size - (col_end - col_start))
        if row_start == 0:
            row_move = int(-1 * row_move)
        if col_start == 0:
            col_move = int(-1 * col_move)

        # 计算梯度的第一范式和角度(角度取 0 到 35 的整数)
        tempDy = self.img['Dy'][row_start:row_end, col_start:col_end].copy()

        tempDx = self.img['Dx'][row_start:row_end, col_start:col_end].copy()
```

```
        tempDx[tempDx == 0] = 1e-5
        modulus_mat = np.power(np.power(tempDx,2) + np.power(tempDy,2),0.5)

        theta_mat = np.array((np.degrees(np.arctan2(tempDy, tempDx)) + 180) / 10, dtype=np.int)
        if row_move > 0:
            modulus_mat = np.r_[modulus_mat, np.zeros((row_move, col_end - col_start))]
            theta_mat = np.r_[theta_mat, np.zeros((row_move, col_end - col_start))]
        elif row_move < 0:
            modulus_mat = np.r_[np.zeros((-1 * row_move, col_end - col_start)), modulus_mat]
            theta_mat = np.r_[np.zeros((-1 * row_move, col_end - col_start)), theta_mat]
        if col_move > 0:
            modulus_mat = np.c_[modulus_mat, np.zeros((size, col_move))]
            theta_mat = np.c_[theta_mat, np.zeros((size, col_move))]
        elif col_move < 0:
            modulus_mat = np.c_[np.zeros((size, -1 * col_move)), modulus_mat]
            theta_mat = np.c_[np.zeros((size, -1 * col_move)), theta_mat]

        # 权重系数矩阵，高斯函数
        gauss = self.getGauss(size)

        # 统计
        wgt_mod = modulus_mat * gauss
        static = np.zeros((1, 36))
        for j in range(36):
            static[0][j] = np.sum(wgt_mod[theta_mat == j])

        # 保存得到的主方向
        the_max = np.max(static)
        direction.append(10 * (np.where(static == the_max)[1][0]))
        self.img['direction'] = direction
```

计算描述子

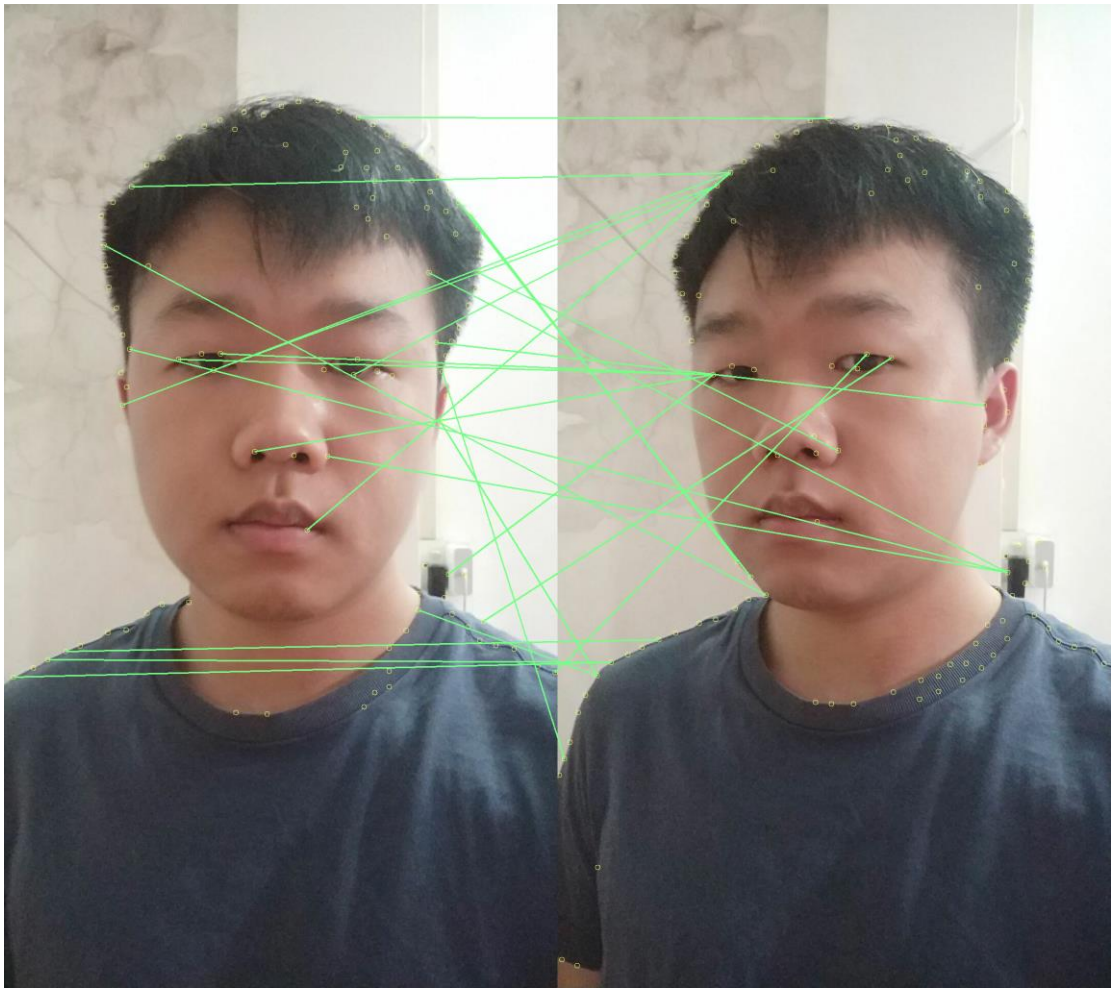
```
def descriptor(self):
    points = self.img['points']
    radius = 8
    size = int(2 * radius + 1)
    self.img['descriptor'] = np.zeros((len(points), 4, 4, 8))
    for i in range(len(points)):
```

```
7.          # 获得该点的坐标
8.          row = int(points[i][0][1])
9.          column = int(points[i][0][0])
10.
11.         # 该点半径为 radius 的所有点的范围
12.         row_start = int(max(row - radius, 0))
13.         row_end = int(min(row + radius + 1, self.img['x']))
14.
15.         col_start = int(max(column - radius, 0))
16.         col_end = int(min(column + radius + 1, self.img['y']))
17.
18.         # 如果范围超出了原图像，则记下，后续补零
19.         row_move = int(size - (row_end - row_start))
20.         col_move = int(size - (col_end - col_start))
21.         if row_start == 0:
22.             row_move = int(-1 * row_move)
23.         if col_start == 0:
24.             col_move = int(-1 * col_move)
25.
26.         # 计算梯度的第二范式和角度(角度取 0 到 7 的整数)
27.         tempDy = self.img['Dy'][row_start:row_end, col_start:col_end].copy()
28.         tempDx = self.img['Dx'][row_start:row_end, col_start:col_end].copy()
29.         tempDx[tempDx == 0] = 1e-5
30.         modulus_mat = np.power(np.power(tempDx,2) + np.power(tempDy,2),0.5)
31.
32.         theta_mat = np.array(
33.             np.degrees(np.arctan2(tempDy, tempDx)) + 180 - self.img['directi
    on'][i])
34.         theta_mat[theta_mat < 0] = theta_mat[theta_mat < 0] + 360
35.         theta_mat = np.array(theta_mat / 45, dtype=np.int)
36.
37.         if row_move > 0:
38.             modulus_mat = np.r_[modulus_mat, np.zeros((row_move, col_end - c
    ol_start))]
39.             theta_mat = np.r_[theta_mat, np.zeros((row_move, col_end - col_s
    tart))]
40.         elif row_move < 0:
41.             modulus_mat = np.r_[np.zeros((-1 * row_move, col_end - col_start
    )), modulus_mat]
42.             theta_mat = np.r_[np.zeros((-1 * row_move, col_end - col_start))
    , theta_mat]
```

```
43.         if col_move > 0:
44.             modulus_mat = np.c_[modulus_mat, np.zeros((size, col_move))]
45.             theta_mat = np.c_[theta_mat, np.zeros((size, col_move))]
46.         elif col_move < 0:
47.             modulus_mat = np.c_[np.zeros((size, -1 * col_move)), modulus_mat
    ]
48.             theta_mat = np.c_[np.zeros((size, -1 * col_move)), theta_mat]
49.
50.         # 权重系数矩阵，高斯函数
51.         gauss = self.getGauss(size)
52.
53.         wgt_mod = modulus_mat * gauss
54.
55.         # 获得每个点对应的 128 维向量，保存为一个(4, 4, 8)的 array
56.         for j in range(4):
57.             for k in range(4):
58.                 # anchor 为计算的 4*4 小区域左上角点的坐标
59.                 anchor = [4 * j, 4 * k]
60.                 if j >= 2:
61.                     anchor[0] = anchor[0] + 1
62.                 if k >= 2:
63.                     anchor[1] = anchor[1] + 1
64.
65.                 # focus_mod 为 4*4 小区域的一范式
66.                 # focus_theta 为 4*4 小区域的角度
67.                 static = np.zeros((1, 8))
68.                 focus_mod = wgt_mod[anchor[0]:anchor[0] + 4, anchor[1]:ancho
    r[1] + 4].copy()
69.                 focus_theta = theta_mat[anchor[0]:anchor[0] + 4, anchor[1]:a
    nchor[1] + 4].copy()
70.
71.                 # 分别求出八个方向的第一范式和
72.                 for theta in range(8):
73.                     static[0][theta] = np.sum(focus_mod[focus_theta == theta
    ])
74.
75.                 # 将这个 8 维向量保存起来
76.                 self.img['descriptor'][i][j][k] = static[0]
77.
78.         # 消除亮度影响
79.         # 对每个 128 维向量，分别做一次归一化(化为单位向量)
80.         vector128 = self.img['descriptor'][i].copy()
81.         self.img['descriptor'][i] = vector128 / np.linalg.norm(vector128)
82.         # 然后将大于 0.2 的值变为 0.2
```

```
83.         self.img['descriptor'][i][self.img['descriptor'][i] > 0.2] = 0.2
84.         # 再做一次归一化
85.         vector128 = self.img['descriptor'][i].copy()
86.         self.img['descriptor'][i] = vector128 / np.linalg.norm(vector128)
87.
88.     # 将(4, 4, 8)的 array 化为一个 128 维向量
89.     self.img['descriptor'] = self.img['descriptor'].reshape((len(points), -1
    ))
```

3 匹配效果



报 告 评 语

	教师签字： 日　　期：
成 绩	