# 计算机视觉-HW3

xTryer

2024/11/10

## 1 Fundamental matrix estimation with ground truth matches

思路：将输入的点对 matches 分离成第一幅图的 pts 和第二幅图的 pts，(normalized or unnormalized) 然后根据八点法构建 A 矩阵，然后将 A 矩阵进行 SVD 分解，取最小的奇异值对应的特征向量构建 3*3 的 F 矩阵，然后将 F 矩阵作 SVD 分解，得到的最小的奇异值设为 0，以此来使得 F 的 rank 为 2，最后如果一开始对 pts 作了标准化，那么需要对 F 进行逆归一化处理。

以 Nomrmalized 算法为例：

```
def normalize_points(pts):
    # input: pts=[[x11,y11],[x12,y12]....]
    pts_mean = np.mean(pts,axis=0)
    pts_std = np.std(pts,axis=0)

    trans_matrix=np.array([
        [1/pts_std[0],0,-pts_mean[0]/pts_std[0]],
        [0,1/pts_std[1],-pts_mean[1]/pts_std[1]],
        [0,0,1]
    ])
    tmp_array = np.ones((1,np.shape(pts)[0]))
    homo_pts = np.vstack((pts.T,tmp_array))

    normalized_pts=(np.dot(trans_matrix,homo_pts).T)[:,:2]
    return normalized_pts, trans_matrix
```

对传入的点列进行标准化处理：

```
def fit_fundamental(matches):
    pic1_pts = matches[:,:2]
    pic2_pts = matches[:,2:]

    normalized_pic1_pts,T_matrix_1 = normalize_points(pic1_pts)
    normalized_pic2_pts,T_matrix_2 = normalize_points(pic2_pts)
```

利用标准化后的点列根据八点法构建矩阵 A

```
    N_ = matches.shape[0]
    A_matrix = np.zeros((N_, 9))
    for i in range(matches.shape[0]):
        x1, y1 = normalized_pic1_pts[i]
        x2, y2 = normalized_pic2_pts[i]
        A_matrix[i] = [x2*x1, x2*y1, x2, y2*x1, y2*y1, y2, x1, y1, 1]
```

对矩阵 A 进行奇异值分解，取其最小奇异值对应的特征向量作为 F 矩阵，再对 F 矩阵进行奇异值分解，利用 rank-2 方法，将其最小的奇异值置 0，再更新 F 矩阵，最后由于进行了标准化，最后需要逆标准化，乘上原来的标准化变换矩阵。

```
    # Use SVD to decomposite the matrix
    U_,S_,V_ = np.linalg.svd(np.transpose(A_matrix).dot(A_matrix))
    smallest_index = np.argmin(S_)
    get_col = V_[smallest_index] # find smallest eigenvalue
    F_matrix_init = np.reshape(get_col, (3, 3))

    f_U, f_S, f_V = np.linalg.svd(F_matrix_init)
    f_S_mat=np.diag(f_S)
    f_S_mat[np.argmin(f_S)]=0
    final_F = np.dot(f_U, np.dot(f_S_mat, f_V))

    final_F = np.dot(T_matrix_2.T, np.dot(final_F, T_matrix_1))
```

对于 unnormalized 的情况，我们只需要把构建 A 矩阵时使用的点列换成原点列即可，且不需要进行最后的逆标准化过程。

最后得到 normalized 下的 Fundamental Matrix:

对于 Library, 得到 Fundamental_Matrix

$$\begin{bmatrix} -4.86210796e-08 & 9.98237781e-07 & -1.47341026e-04 \\ -5.80698246e-06 & -5.65060960e-08 & 1.07254893e-02 \\ 1.38164930e-03 & -9.63586245e-03 & -2.60674702e-01 \end{bmatrix}$$

误差 Loss 为:$4.857831302275882e-06$

效果图:

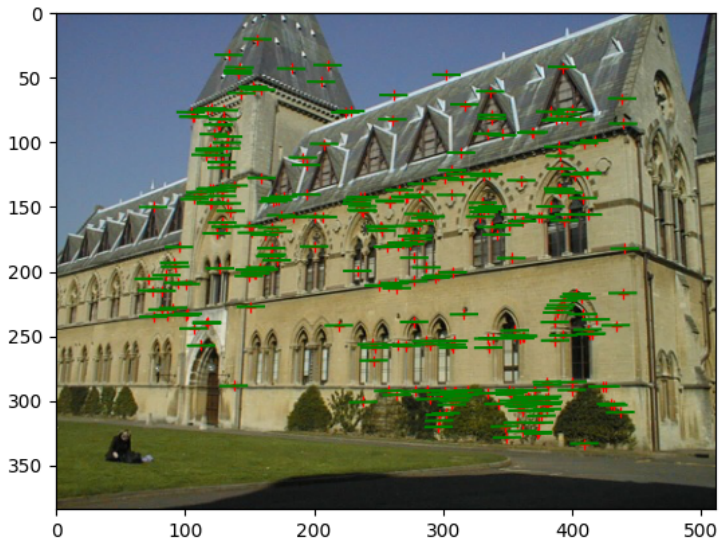


图 1: Task 1-Normalized Library

对于 Lab, 得到 Fundamental_Matrix

$$\begin{bmatrix} 2.02478523e-07 & -2.78039315e-06 & 6.94781145e-04 \\ -1.92581510e-06 & 4.74398800e-07 & -5.59813726e-03 \\ 4.16157075e-05 & 7.69192568e-03 & -1.78588215e-01 \end{bmatrix}$$

误差 Loss 为:$2.2277388816997462e-05$
效果图:

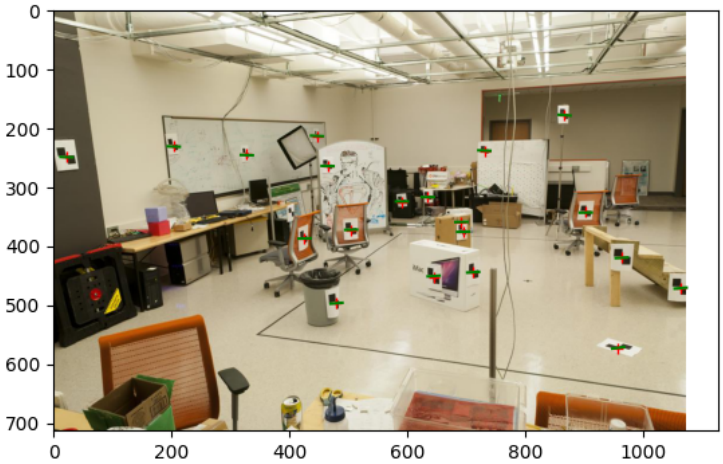


图 2: Task 1-Normalized Lab

而对于 unnormalized 下的 Fundamental Matrix:

对于 Library, 得到 Fundamental_Matrix

$$\begin{bmatrix} 1.32341616e-06 & -1.36640518e-05 & 6.82803869e-04 \\ 2.88178174e-05 & -2.66440806e-07 & -4.09069255e-02 \\ -5.62362952e-03 & 3.72771609e-02 & 9.98451273e-01 \end{bmatrix}$$

误差 Loss 为:0.0001614980653917085

效果图:

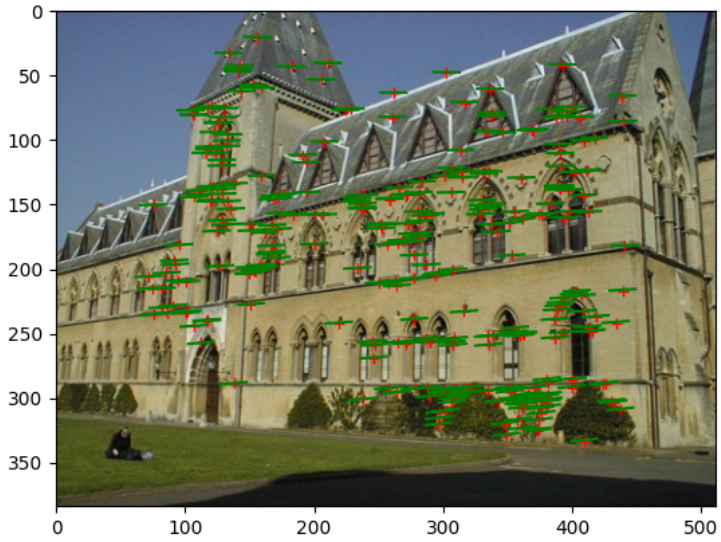


图 3: Task 1-Unnormalized Library

对于 Lab, 得到 Fundamental_Matrix

$$\begin{bmatrix} -5.36264198e-07 & 7.90364770e-06 & -1.88600204e-03 \\ 8.83539184e-06 & 1.21321685e-06 & 1.72332901e-02 \\ -9.07382265e-04 & -2.64234650e-02 & 9.99500092e-01 \end{bmatrix}$$

误差 Loss 为:0.0037634023966661954

效果图:


图 4: Task 1-Unnormalized Lab

可以看出，由 normalized 的点列计算出来的基础矩阵得到的误差要显著小于 unnormalized 的点列计算出来的基础矩阵得到的误差。

## 2 Camera calibration

思路：根据 $Points\_2d = P \cdot Points\_3d$ 来构造方程组，构造 A 矩阵，然后对 A 矩阵进行 SVD 分解，取得到的最小奇异值对应的特征向量作为投影矩阵 P

```
def calc_projection(points_2d, points_3d):
    point_pair_num = np.shape(points_2d)[0]
    A_mat = np.zeros((2*point_pair_num,12))
    for i in range(0,point_pair_num):
        x_i,y_i=points_2d[i,:2]
        X_i,Y_i,Z_i=points_3d[i,:3]
        A_mat[2*i]=[X_i,Y_i,Z_i,1,0,0,0,0,-x_i*X_i,-x_i*Y_i,-x_i*Z_i,-x_i]
        A_mat[2*i+1]=[0,0,0,0,X_i,Y_i,Z_i,1,-y_i*X_i,-y_i*Y_i,-y_i*Z_i,-y_i]

    U_,S_,V_ = np.linalg.svd(np.transpose(A_mat).dot(A_mat))
    smallest_index = np.argmin(S_)
    get_col = V_[smallest_index] # find smallest eigenvalue
    final_P_mat = np.reshape(get_col, (3, 4))
    return final_P_mat
```

对于 lab_a, 我们得到投影矩阵:

$$\begin{bmatrix} -3.09963905e-03 & -1.46205512e-04 & 4.48499082e-04 & 9.78930652e-01 \\ -3.07018335e-04 & -6.37193925e-04 & 2.77356128e-03 & 2.04144529e-01 \\ -1.67933505e-06 & -2.74767724e-06 & 6.83966359e-07 & 1.32882928e-03 \end{bmatrix}$$

对应的 $residual = 13.545763062803358, distance = 3.9652233428826427$

对于 lab_b, 我们得到投影矩阵:

$$\begin{bmatrix} -6.93154380e-03 & 4.01684151e-03 & 1.32603128e-03 & 8.26700552e-01 \\ -1.54768685e-03 & -1.02452783e-03 & 7.27440489e-03 & 5.62523258e-01 \\ -7.60945724e-06 & -3.70953992e-06 & 1.90203482e-06 & 3.38807608e-03 \end{bmatrix}$$

对应的 $residual = 15.544940188383139, distance = 3.8617794795037046$

# 3 Calculate the camera matrices

思路：由 $P = K[R|T]$, 其中 K 为上三角矩阵，R 为正交矩阵我们对 P 矩阵左侧的 3*3 矩阵进行 RQ 分解，其中分解得到的 Q 矩阵对应投影矩阵计算中的 R，分解得到的矩阵 R 对应投影矩阵计算中的 Q，这样我们就获得了矩阵 K 和 R，最后我们通过解矩阵方程得到矩阵 T，最后我们还需要对 K 矩阵进行标准化处理 (保证 $K[2,2] = 1$).

```
def rq_decomposition(P):
    R_mat,Q_mat = scipy.linalg.rq(P[:,:3])
    K=R_mat
    R=Q_mat
    T=np.linalg.solve(np.dot(K,R),P[:,3])
    K=R_mat/R_mat[2,2]

    return K, R, T
```

得到结果：对于 Lab_a

$$K\_matrix = \begin{bmatrix} 780.56707396 & 1.99870942 & 545.67047095 \\ -0. & 779.99063806 & 384.15992749 \\ -0. & -0. & 1. \end{bmatrix}$$

$$R\_matrix = \begin{bmatrix} 0.84996675 & -0.52615469 & -0.02679126 \\ -0.13167588 & -0.16292412 & -0.97781245 \\ 0.51011567 & 0.83463583 & -0.20776197 \end{bmatrix}$$

$$T\_matrix = \begin{bmatrix} -305.83276989 \\ -304.20104026 \\ -30.13699179 \end{bmatrix}$$

对于 Lab_b

$$K\_matrix = \begin{bmatrix} 767.01503632 & 8.37490683 & 536.20612054 \\ -0. & 772.02236183 & 390.71295481 \\ -0. & -0. & 1. \end{bmatrix}$$

$$R\_matrix = \begin{bmatrix} 0.43076711 & -0.90177028 & -0.03535632 \\ -0.21279855 & -0.06342297 & -0.97503554 \\ 0.87701566 & 0.42753701 & -0.21921594 \end{bmatrix}$$

$$T\_matrix = \begin{bmatrix} -303.10004025 \\ -307.18428028 \\ -30.42166839 \end{bmatrix}$$

对于 Library_a

$$K\_matrix = \begin{bmatrix} -5.79790975e+02 & 1.11782151e-06 & 2.56991552e+02 \\ 0.00000000e+00 & -5.39711147e+02 & 2.04317558e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

$$R\_matrix = \begin{bmatrix} -0.00966193 & -0.44135514 & -0.8972805 \\ -0.98017972 & -0.17338541 & 0.09583956 \\ -0.19787463 & 0.88042214 & -0.43093212 \end{bmatrix}$$

$$T\_matrix = \begin{bmatrix} -7.28863053 \\ 21.52118112 \\ -17.73503585 \end{bmatrix}$$

对于 Library_b

$$K\_matrix = \begin{bmatrix} -5.47469106e+02 & 7.30842576e-06 & 2.58430094e+02 \\ 0.00000000e+00 & -5.12933585e+02 & 2.04985542e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

$$R\_matrix = \begin{bmatrix} 0.01861384 & -0.67624076 & -0.73644549 \\ -0.981381 & -0.15319028 & 0.11586227 \\ -0.19116708 & 0.72057697 & -0.6665013 \end{bmatrix}$$

$$T\_matrix = \begin{bmatrix} -6.89405488 \\ 15.39232716 \\ -23.41498687 \end{bmatrix}$$

# 4 Triangulation

思路：利用 P1，P2，point1,point2 与 point_3d 的关系列出线性方程组，得到 A 矩阵，然后对 A 矩阵进行 SVD 分解，取其最小奇异值对应的特征向量代表最终的 3D 坐标的齐次形式，然后转该 4D 齐次坐标为 3D 笛卡尔坐标，得到对应的 point_3d

```
def triangulate_points(P1, P2, point1, point2):
    A_mat = np.zeros((4,4))
    A_mat[0]=P1[0]-point1[0]*P1[2]
    A_mat[1]=P1[1]-point1[1]*P1[2]
    A_mat[2]=P2[0]-point2[0]*P2[2]
    A_mat[3]=P2[1]-point2[1]*P2[2]

    U_,S_,V_ = np.linalg.svd(np.transpose(A_mat).dot(A_mat))
    smallest_index = np.argmin(S_)
    get_col = V_[smallest_index] # find smallest eigenvalue

    point_3d = get_col[:3]/get_col[3]

    return point_3d
```

最终检验得到

$$residual\_lab\_a = 10.899378165050477$$
$$residual\_lab\_b = 1.548507488996827$$
$$residual\_library\_a = 24.662071196867792$$
$$residual\_library\_b = 28.64953773526141$$

# 5 Fundamental matrix estimation without ground-truth matches

思路：对于传入的两张图片，先转成灰度图，然后用 opencv2 库中的 sift 算子做特征检测，再利用 BFMatcher.knnMatch，通过 knn(k=2) 暴力搜索出第一张图中的每个特征点与第二张图中最接近的两个特征点，我们在第一个匹配点对的特征距离小于 0.8* 第二个点对的特征距离时认为第一个点对是一个好的 match，最后形成 selected_matches, 然后选出对应的 point1 和 point2，通过 RANSAC 算法 (align_pair_with_RANSAC 函数中实现) 计算出“最优的”fundamental_matrix。

手动实现 RANSAC 算法, 并返回内点数量、内点点对、残差

```
def align_pair_with_RANSAC(pixels_1, pixels_2):
    pixels_pair_num = len(pixels_1)
    assert pixels_pair_num>=4,"To compute need more pixels-pair\n"

    final_fundamental_matrix=np.zeros((3,3)) # initial matrix
    final_inliers_num = -1
    final_matches =[]
    final_res_ = 0
    pixels_num =len(pixels_1)
    itration_times = 1200
    threshold_ = 0.02
```

```
    homogenenous_pixels_1 = convert_to_homogeneous_coordinates(pixels_1)
    homogenenous_pixels_2 = convert_to_homogeneous_coordinates(pixels_2)

    range_num = [i for i in range(pixels_num)]
    for it_ in range(itration_times):
        random_samples_indices = np.random.choice(range_num,int(pixels_num/50),replace=False)
        assert int(pixels_num/50)>=4
        sample_pixels_1=[]
        sample_pixels_2=[]
        for index_ in random_samples_indices:
            sample_pixels_1.append(pixels_1[index_])
            sample_pixels_2.append(pixels_2[index_])

        sample_fundamental_matrix =
            fit_fundamental(np.hstack((sample_pixels_1,sample_pixels_2)))

        sample_inlier_cnt = 0
        sample_res_sum=0
        sample_final_matches = []
        for match_index in range(pixels_num):
            check_point1 = homogenenous_pixels_1[match_index]
            check_point2 = homogenenous_pixels_2[match_index]
            res_ =
                np.abs(np.dot(check_point2.transpose(),np.dot(sample_fundamental_matrix,check_point1)

            if res_<threshold_:
                sample_inlier_cnt+=1
                sample_res_sum+=res_
                sample_final_matches.append(np.hstack((pixels_1[match_index],pixels_2[match_index])))

        if sample_inlier_cnt>final_inliers_num:
            final_fundamental_matrix=sample_fundamental_matrix
            final_inliers_num=sample_inlier_cnt
            final_res_=sample_res_sum
            final_matches=sample_final_matches

    return final_fundamental_matrix,final_inliers_num,final_res_,final_matches
```

fit_fundamental_without_gt 主函数：

```
def fit_fundamental_without_gt(image1, image2):
    gray_img1 = image1
    gray_img2 = image2
```

```
    if gray_img1.shape[2]==3:
        gray_img1 = cv2.cvtColor(gray_img1,cv2.COLOR_BGR2GRAY)
    if gray_img2.shape[2]==3:
        gray_img2 = cv2.cvtColor(gray_img2,cv2.COLOR_BGR2GRAY)

    # cv2.sift
    my_sift = cv2.SIFT_create()
    keypoints_1,description_1 = my_sift.detectAndCompute(gray_img1,None)
    keypoints_2,description_2 = my_sift.detectAndCompute(gray_img2,None)

    # cv2.BFMatcher
    my_bf = cv2.BFMatcher()
    # use knnmatch to get two match pair
    get_matches = my_bf.knnMatch(description_1,description_2,k=2)

    selected_matchs =[]
    for match_ in get_matches:
        if match_[0].distance<0.8*match_[1].distance:
            selected_matchs.append(match_[0])

    points_selected_1 = np.array([keypoints_1[pair_.queryIdx].pt for pair_ in
        selected_matchs]).astype(np.float32)
    points_selected_2 = np.array([keypoints_2[pair_.trainIdx].pt for pair_ in
        selected_matchs]).astype(np.float32)

    fundamental_matrix, inliers_num,residuals_sum,final_matches =
        align_pair_with_RANSAC(points_selected_1,points_selected_2)

    final_matches=np.array(final_matches)

    mean_residuals = residuals_sum/inliers_num

    print("get inliers_num:",inliers_num)
    print("get average residual for the inliers:",mean_residuals)


    return fundamental_matrix, final_matches
```

最终能够随机得到 library1 与 library2 的匹配结果：

inlier_num = 457

average_residual = 0.005160176026315073

效果图:

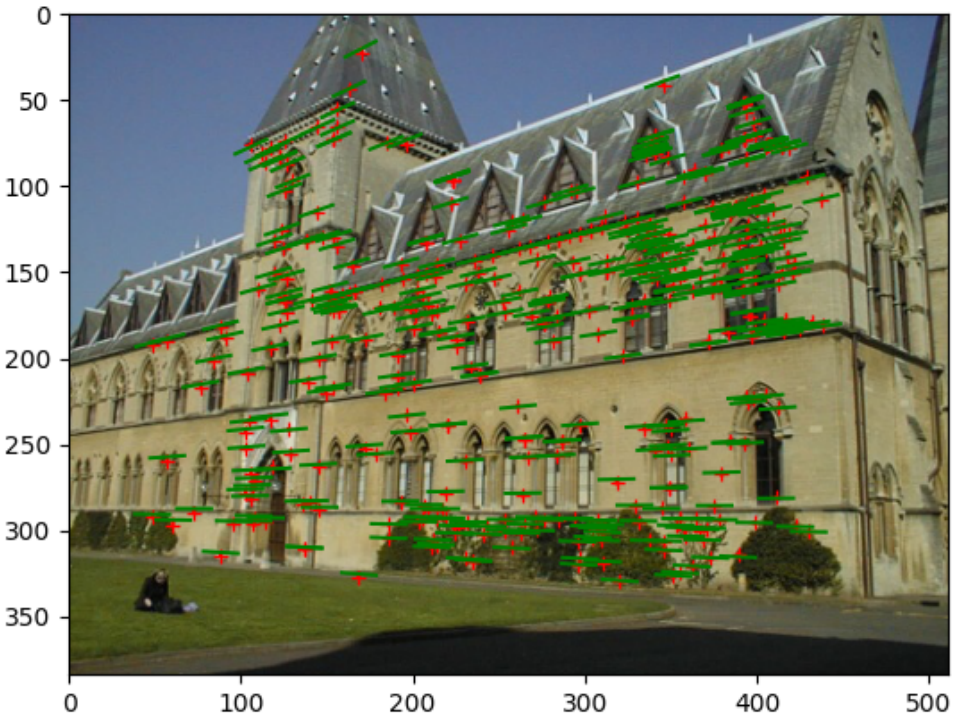


图 5: Task 5:Normalized Library