CV HW2

111061553 謝霖泳

Problems

1. Camera Pose from Essential Matrix

- `estimate_initial_RT()`

```
def estimate_initial_RT(E):
    Z = np.zeros((3, 3))
    W = np.zeros((3, 3))
    Z[0][1], Z[1][0] = 1, -1
    W[0][1], W[1][0], W[2][2] = -1, 1, 1
    U, sigma, VT = np.linalg.svd(E)
    M = np.matmul(np.matmul(U, Z), U.T)
    Q1 = np.matmul(np.matmul(U, W), VT)
    Q2 = np.matmul(np.matmul(U, W.T), VT)
    R1 = np.linalg.det(Q1) * Q1
    R2 = np.linalg.det(Q2) * Q2
    T1 = U[:, 2]
    T2 = -U[:, 2]
    R1T1 = np.concatenate([R1, np.expand_dims(T1, axis=1)], axis=1)
    R1T2 = np.concatenate([R1, np.expand_dims(T2, axis=1)], axis=1)
    R2T1 = np.concatenate([R2, np.expand_dims(T1, axis=1)], axis=1)
    R2T2 = np.concatenate([R2, np.expand_dims(T2, axis=1)], axis=1)
    return np.array([R1T1, R1T2, R2T1, R2T2])
```

將E矩陣做SVD分解為U、Σ、V^T，便可從U得到兩種T，T_1和T_2。接著根據公式計算出相應的M以及兩種Q，即Q_1和Q_2。有了兩個Q，再根據公式$R = det(Q) \cdot Q$得到R_1和R_2，最後將R_1T_1、R_1T_2、R_2T_1、R_2T_2 包起來return回去即為所求。

2. Linear 3D Points Estimation

- `linear_estimate_3d_point()`

```
def linear_estimate_3d_point(image_points, camera_matrices):
    M = deepcopy(camera_matrices)
    n = M.shape[0]
    p = deepcopy(image_points)
    mat = np.zeros((n * 2, 4))
    for i in range(0, n):
        mat[i * 2] = p[i, 1] * M[i, 2] - M[i, 1]
        mat[i * 2 + 1] = M[i, 0] - p[i, 0] * M[i, 2]
    U, sigma, VT = np.linalg.svd(mat)
    P_temp = VT[-1]
    P_temp /= P_temp[-1]
    return P_temp[:3]
```

先利用M和p製造出下方算式左邊的矩陣，名為 `mat` ，接著將 `mat` 做SVD分解得到U、Σ、V^T，將V^T的最後一個row同除以最後一個row的最後一個數字之後，return前三個數字即為所求。

$$\begin{bmatrix} v_1 M_1^3 - M_1^2 \\ M_1^1 - u_1 M_1^3 \\ \vdots \\ v_n M_n^3 - M_n^2 \\ M_n^1 - u_n M_n^3 \end{bmatrix} \cdot P = 0.$$

3. Non-Linear 3D Points Estimation

- `reprojection error()`

```
def reprojection_error(point_3d, image_points, camera_matrices):
    M = deepcopy(camera_matrices)
    P = deepcopy(point_3d)
    P = np.append(P, 1)
    p = deepcopy(image_points)
    err = []
    for i in range(M.shape[0]):
        yi = np.dot(M[i], P)
        pi_prime = np.array([yi[0], yi[1]]) / yi[2]
        ei = pi_prime - p[i]
        err.extend(list(ei))
    return np.array(err)
```

根據公式$y = M_i P$計算出每個y_i，大小為3 * 1。再根據下方公式計算出每個p_i'，最後計算$p_i' - p_i$得到e_i。

$$p_i' = \begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{y_3} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

- `jacobian()`

```python
def jacobian(point_3d, camera_matrices):
    P = np.append(point_3d, 1)
    M = deepcopy(camera_matrices)
    Jac = np.zeros((2 * M.shape[0], 3))
    J_row = []

    for i in range(M.shape[0]):
        Mi = M[i]
        yi = np.matmul(Mi, P)
        J_row.append((Mi[0, :3] * yi[2] - Mi[2, :3] * yi[0]) / yi[2] ** 2)
        J_row.append((Mi[1, :3] * yi[2] - Mi[2, :3] * yi[1]) / yi[2] ** 2)

    for i in range(M.shape[0] * 2):
        Jac[i] = J_row[i]

    return Jac
```

$$M_i = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix}, \; P = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$y_i = M_i P = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00}X + a_{01}Y + a_{02}Z + a_{03} \\ a_{10}X + a_{11}Y + a_{12}Z + a_{13} \\ a_{20}X + a_{21}Y + a_{22}Z + a_{23} \end{bmatrix}$$

$$p_i' = \frac{1}{y_{i3}} \begin{bmatrix} y_{i1} \\ y_{i2} \end{bmatrix} = \begin{bmatrix} \frac{a_{00}X+a_{01}Y+a_{02}Z+a_{03}}{a_{20}X+a_{21}Y+a_{22}Z+a_{23}} \\ \\ \frac{a_{10}X+a_{11}Y+a_{12}Z+a_{13}}{a_{20}X+a_{21}Y+a_{22}Z+a_{23}} \end{bmatrix}$$

$$\frac{\partial e_i}{\partial X} = \begin{bmatrix} \frac{a_{00}(a_{20}X+a_{21}Y+a_{22}Z+a_{23}) - a_{20}(a_{00}X+a_{01}Y+a_{02}Z+a_{03})}{(a_{20}X+a_{21}Y+a_{22}Z+a_{23})^2} \\ \\ \frac{a_{10}(a_{20}X+a_{21}Y+a_{22}Z+a_{23}) - a_{20}(a_{00}X+a_{01}Y+a_{02}Z+a_{03})}{(a_{20}X+a_{21}Y+a_{22}Z+a_{23})^2} \end{bmatrix}$$

$$\frac{\partial e_i}{\partial Y} = \begin{bmatrix} \frac{a_{01}(a_{20}X+a_{21}Y+a_{22}Z+a_{23}) - a_{21}(a_{00}X+a_{01}Y+a_{02}Z+a_{03})}{(a_{20}X+a_{21}Y+a_{22}Z+a_{23})^2} \\ \\ \frac{a_{11}(a_{20}X+a_{21}Y+a_{22}Z+a_{23}) - a_{21}(a_{00}X+a_{01}Y+a_{02}Z+a_{03})}{(a_{20}X+a_{21}Y+a_{22}Z+a_{23})^2} \end{bmatrix}$$

$$\frac{\partial e_i}{\partial Z} = \begin{bmatrix} \frac{a_{02}(a_{20}X+a_{21}Y+a_{22}Z+a_{23})-a_{22}(a_{00}X+a_{01}Y+a_{02}Z+a_{03})}{(a_{20}X+a_{21}Y+a_{22}Z+a_{23})^2} \\ \\ \frac{a_{12}(a_{20}X+a_{21}Y+a_{22}Z+a_{23})-a_{22}(a_{00}X+a_{01}Y+a_{02}Z+a_{03})}{(a_{20}X+a_{21}Y+a_{22}Z+a_{23})^2} \end{bmatrix}$$

根據公式$e_i = p_i' - p_i$，因為p_i為常數，不影響偏微分的結果，因此考慮p_i'對每個變數偏微分的結果，計算結果如上，即可得到Jacobian的結果。

- `nonlinear_estimate_3d_point()`

```
def nonlinear_estimate_3d_point(image_points, camera_matrices):
    P = linear_estimate_3d_point(image_points, camera_matrices)
    for _ in range(10):
        J = jacobian(point_3d=P, camera_matrices=camera_matrices)
        e = reprojection_error(point_3d=P, image_points=image_points, \
            camera_matrices=camera_matrices)
        P = P - np.matmul(np.matmul(np.linalg.inv(np.matmul(J.T, J)), J.T), e)
    return P
```

根據公式$P = P - (J^TJ)^{-1}J^Te$做10次，其中J來自 `jacobian()`，e來自 `reprojection_error()`。

4. Decide the Correct RT

- `estimate_RT_from_E()`

```
def estimate_RT_from_E(E, image_points, K):
    init_RT = estimate_initial_RT(E)       # 4 * 3 * 4
    temp = np.matmul(K, np.hstack((np.eye(3), np.zeros((3,1)))))    # 3 * 4
    camera_matrices = np.array((temp, np.zeros(temp.shape)))
    cnt_list = []
    for i in range(init_RT.shape[0]):
        camera_matrices[1] = np.matmul(K, init_RT[i])
        for j in range(image_points.shape[0]):
            cnt = 0
            nonlinear_pt = nonlinear_estimate_3d_point(image_points[j], camera_matri
            nonlinear_pt = np.append(nonlinear_pt, 1)
            temp2 = np.vstack((init_RT[i], [0, 0, 0, 1]))
            Pj_prime = np.matmul(temp2, np.array((nonlinear_pt[0], nonlinear_pt[1],
            Pj_prime /= Pj_prime[3]
            Pj_prime = Pj_prime[:3]
            if nonlinear_pt[2] > 0 and Pj_prime[2] > 0:
                cnt += 1
        cnt_list.append(cnt)
    return init_RT[cnt_list.index(max(cnt_list))]
```

先呼叫 `estimate_initial_RT()`，得到四種可能的RT，對於每個可能的RT，呼叫 `nonlinear_estimate_3d_point()`，得到對應的p_j，接著利用 `RT[i]` 對p_j做矩陣相乘，轉到另一個camera的座標p_j'。

最後，檢查每一組p_j以及p'_j的z座標，看哪一組的兩個z座標均為正數，就代表該組數據對應正確的RT，最後return正確的RT即為所求。

Result

最終得到的結果如上圖，與pdf中完全相同。

我覺得這次作業比前次複雜許多，但完成之後，我對於相機的translation、rotation等運算更加的了解。