Computer Vision

HW2 Structure from Motion

學號:112061619

姓名:王證皓

1. Camera Pose from Essential Matrix

(1) Code

```python
def estimate_initial_RT(E):
    # TODO: Implement this method!
    Z=np.array([[0,1,0],
                [-1,0,0],
                [0,0,0]])
    W=np.array([[0,-1,0],
                [1,0,0],
                [0,0,1]])
    U,S,V_transpose = np.linalg.svd(E)
    M=U.dot(Z).dot(U.T)
    Q1=U.dot(W).dot(V_transpose)
    R1=np.linalg.det(Q1)*Q1
    T1=U[:3,2:3]
    RT1=np.around(np.hstack((R1,T1)),decimals=4)

    Q2=U.dot(W).dot(V_transpose)
    R2=np.linalg.det(Q2)*Q2
    T2=-U[:3,2:3]
    RT2=np.around(np.hstack((R2,T2)),decimals=4)

    Q3=U.dot(W.T).dot(V_transpose)
    R3=np.linalg.det(Q3)*Q3
    T3=U[:3,2:3]
    RT3=np.around(np.hstack((R3,T3)),decimals=4)

    Q4=U.dot(W.T).dot(V_transpose)
    R4=np.linalg.det(Q4)*Q4
    T4=-U[:3,2:3]
    RT4=np.around(np.hstack((R4,T4)),decimals=4)
    RT=np.stack((RT1,RT2,RT3,RT4))
    return RT
    raise Exception('Not Implemented Error')
```

(2) Results

```
--------------------------------------------------------------------------------
Part A: Check your matrices against the example R,T
--------------------------------------------------------------------------------
Example RT:
 [[ 0.9736 -0.0988 -0.2056  0.9994]
 [ 0.1019  0.9948  0.0045 -0.0089]
 [ 0.2041 -0.0254  0.9786  0.0331]]

Estimated RT:
 [[[ 0.9831 -0.1179 -0.1404  0.9994]
  [-0.1193 -0.9929 -0.0015 -0.0089]
  [-0.1392  0.0182 -0.9901  0.0331]]

 [[ 0.9831 -0.1179 -0.1404 -0.9994]
  [-0.1193 -0.9929 -0.0015  0.0089]
  [-0.1392  0.0182 -0.9901 -0.0331]]

 [[ 0.9736 -0.0988 -0.2056  0.9994]
  [ 0.1019  0.9948  0.0045 -0.0089]
  [ 0.2041 -0.0254  0.9786  0.0331]]

 [[ 0.9736 -0.0988 -0.2056 -0.9994]
  [ 0.1019  0.9948  0.0045  0.0089]
  [ 0.2041 -0.0254  0.9786 -0.0331]]]
--------------------------------------------------------------------------------
```

(3) Insights

透過將 E 做 SVD 得到 U、S、V.T，其中 R 有 det(UWV.T)UW(V.T)以及 U(W.T)(V.T)兩種，而 T 則有 u3、-u3 兩種，所以總共會輸出四個 RT 矩陣。

2. Linear 3D Points Estimation

(1) Code

```python
def linear_estimate_3d_point(image_points, camera_matrices):
    # TODO: Implement this method!
    M=np.size(camera_matrices,0)
    A=np.zeros((2*M,4))
    for i in range(M):
        A[2*i,:]=image_points[i,1]*camera_matrices[i,2,:]-camera_matrices[i,1,:]
        A[2*i+1,:]=camera_matrices[i,0,:]-image_points[i,0]*camera_matrices[i,2,:]
    U,S,V_transpose=np.linalg.svd(A)
    point_3d=V_transpose[3,:3]/V_transpose[3,3]
    return point_3d
    raise Exception('Not Implemented Error')
```

(2) Results

```
--------------------------------------------------------------------------------
Part B: Check that the difference from expected point
is near zero
--------------------------------------------------------------------------------
Difference:  0.0029243053036643873
--------------------------------------------------------------------------------
```

(3) Insights

此處將圖像座標傳入，並用投影矩陣將其結合成線性方程式，最後再使用 SVD 得到 3D 的座標，且可看到錯誤接近 0。

3. Non-Linear 3D Points Estimation

(1) Code

```python
def reprojection_error(point_3d, image_points, camera_matrices):
    # TODO: Implement this method!
    M=np.size(camera_matrices,0)
    y=np.zeros((M, 3))
    error=np.zeros(2*M)
    for i in range(M):
        y[i,:]=camera_matrices[i,:,:].dot(np.append(point_3d,[1]))
        error[2*i]=y[i,0]/y[i,2]-image_points[i,0]
        error[2*i+1]=y[i,1]/y[i,2]-image_points[i,1]
    return error
    raise Exception('Not Implemented Error')
```

```python
jacobian(point_3d, camera_matrices):
# TODO: Implement this method!
y=np.zeros((3,1))
M=np.size(camera_matrices,0)
jacobian=np.zeros((2*M,3))
for i in range(M):
    y=camera_matrices[i,:,:].dot(np.append(point_3d,[1]))
    jacobian[2*i,:]=np.array([(y[2]*camera_matrices[i,0,0]-camera_matrices[i,2,0]*y[0]),(y[2]*camera_matrices[i,0,1]-camera_matrices[i,2,1]*y[0]),(y[2]*camera_matrices[i,0,2]-camera_matrices[i,2,2]*y[0])])/y[2]**2
    jacobian[2*i+1,:]=np.array([(y[2]*camera_matrices[i,1,0]-camera_matrices[i,2,0]*y[1]),(y[2]*camera_matrices[i,1,1]-camera_matrices[i,2,1]*y[1]),(y[2]*camera_matrices[i,1,2]-camera_matrices[i,2,2]*y[1])])/y[2]**2
return jacobian
raise Exception('Not Implemented Error')
```

```
nonlinear_estimate_3d_point(image_points, camera_matrices):
# TODO: Implement this method!
point_3d=linear_estimate_3d_point(image_points, camera_matrices)
for i in range(10):
    e=reprojection_error(point_3d, image_points, camera_matrices)
    J=jacobian(point_3d, camera_matrices)
    point_3d=point_3d-np.linalg.inv((J.T.dot(J))).dot(J.T).dot(e)
return point_3d
raise Exception('Not Implemented Error')
```

(2) Results

```
--------------------------------------------------------------------------------
Part C: Check that the difference from expected error/Jacobian
is near zero
--------------------------------------------------------------------------------
Error Difference:  8.301300130674275e-07
Jacobian Difference:  1.817115702351657e-08
--------------------------------------------------------------------------------
Part D: Check that the reprojection error from nonlinear method
is lower than linear method
--------------------------------------------------------------------------------
Linear method error: 98.7354235689419
Nonlinear method error: 95.59481784846034
--------------------------------------------------------------------------------
```

(3) Insights

先將 3D 座標投影後得到 2D 的座標，再以此計算錯誤，接著以偏微分的方式來得到 Jacobian 矩陣，最後用 Gauss-Newton 來做優化，並且可看到錯誤下降。

4. Decide the Correct RT

(1) Code

```python
def estimate_RT_from_E(E, image_points, K):
    # TODO: Implement this method!
    RT0=estimate_initial_RT(E)
    N=np.size(image_points,0)
    M=K.dot(np.hstack((np.eye(3), np.zeros((3,1)))))
    M1=K.dot(RT0[0,:,:])
    M2=K.dot(RT0[1,:,:])
    M3=K.dot(RT0[2,:,:])
    M4=K.dot(RT0[3,:,:])
    x1=np.zeros((2,3,4))
    x1[0,:,:]=M
    x1[1,:,:]=M1
    x2=np.zeros((2,3,4))
    x2[0,:,:]=M
    x2[1,:,:]=M2
    x3=np.zeros((2,3,4))
    x3[0,:,:]=M
    x3[1,:,:]=M3
    x4=np.zeros((2,3,4))
    x4[0,:,:]=M
    x4[1,:,:]=M4
    cnt1=0
    cnt2=0
    cnt3=0
    cnt4=0
    RT=np.zeros((3,4))
    for i in range(N):
        points_3d1=nonlinear_estimate_3d_point(image_points[i,:,:], x1)
        points_3d2=nonlinear_estimate_3d_point(image_points[i,:,:], x2)
        points_3d3=nonlinear_estimate_3d_point(image_points[i,:,:], x3)
        points_3d4=nonlinear_estimate_3d_point(image_points[i,:,:], x4)
        if(points_3d1[2]>0):
            cnt1=cnt1+1
        if(points_3d2[2]>0):
            cnt2=cnt2+1
        if(points_3d3[2]>0):
            cnt3=cnt3+1
        if(points_3d4[2]>0):
            cnt4=cnt4+1
```

```python
    if((cnt1>=cnt2)&(cnt1>=cnt3)&(cnt1>=cnt4)):
        for j in range(3):
            for k in range(4):
                RT[j,k]=RT0[0,j,k]
    if((cnt2>=cnt1)&(cnt2>=cnt3)&(cnt2>=cnt4)):
        for j in range(3):
            for k in range(4):
                RT[j,k]=RT0[1,j,k]
    if((cnt3>=cnt1)&(cnt3>=cnt2)&(cnt3>=cnt4)):
        for j in range(3):
            for k in range(4):
                RT[j,k]=RT0[2,j,k]
    if((cnt4>=cnt1)&(cnt4>=cnt2)&(cnt4>=cnt3)):
        for j in range(3):
            for k in range(4):
                RT[j,k]=RT0[3,j,k]
    return RT
    raise Exception('Not Implemented Error')
```

(2) Result

```
--------------------------------------------------------------------------------
Part E: Check your matrix against the example R,T
--------------------------------------------------------------------------------
Example RT:
 [[ 0.9736 -0.0988 -0.2056  0.9994]
 [ 0.1019  0.9948  0.0045 -0.0089]
 [ 0.2041 -0.0254  0.9786  0.0331]]

Estimated RT:
 [[ 0.9736 -0.0988 -0.2056  0.9994]
 [ 0.1019  0.9948  0.0045 -0.0089]
 [ 0.2041 -0.0254  0.9786  0.0331]]
--------------------------------------------------------------------------------
Part F: Run the entire SFM pipeline
--------------------------------------------------------------------------------
Save results to results.npy!
```

(3) Insights

每組 RT 都會對應到兩組相機，以此來計算每組 RT 所對應的 3D 座標，並且統計得到最多 Z 值為正的 RT，最後可看到結果與預期相符。

5. Final Results