Computer Vision

Homework 2: Structure from Motion (SfM)

110065502 李杰穎

3.1 Camera Pose from Essential Matrix

In the estimate_initial_RT, we can compute 4 initial guesses of the relative RT between the two cameras. E is the Essential Matrix between the two cameras. RT is a 4x3x4 tensor in which the 3x4 matrix RT is one of the four possible transformations.

```python
def estimate_initial_RT(E):
    # TODO: Implement this method!
    W = np.array([[0, -1, 0], [1, 0, 0], [0, 0, 1]])
    U, S, V = np.linalg.svd(E)
    Q1 = np.dot(U, np.dot(W, V))
    Q2 = np.dot(U, np.dot(W.T, V))
    R1 = np.linalg.det(Q1) * Q1
    R2 = np.linalg.det(Q2) * Q2
    t1 = U[:, 2]
    t2 = -U[:, 2]
    RT = np.zeros((4, 3, 4))
    RT[0, :, :] = np.hstack((R1, t1.reshape(3, 1)))
    RT[1, :, :] = np.hstack((R1, t2.reshape(3, 1)))
    RT[2, :, :] = np.hstack((R2, t1.reshape(3, 1)))
    RT[3, :, :] = np.hstack((R2, t2.reshape(3, 1)))

    return RT
```

Output:

The third matrix matches the Example RT.

```
--------------------------------------------------------------------------------
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:
 [[[ 0.98305251 -0.11787055 -0.14040758  0.99941228]
  [-0.11925737 -0.99286228 -0.00147453 -0.00886961]
  [-0.13923158  0.01819418 -0.99009269  0.03311219]]

 [[ 0.98305251 -0.11787055 -0.14040758 -0.99941228]
  [-0.11925737 -0.99286228 -0.00147453  0.00886961]
  [-0.13923158  0.01819418 -0.99009269 -0.03311219]]

 [[ 0.97364135 -0.09878708 -0.20558119  0.99941228]
  [ 0.10189204  0.99478508  0.00454512 -0.00886961]
  [ 0.2040601  -0.02537241  0.97862951  0.03311219]]

 [[ 0.97364135 -0.09878708 -0.20558119 -0.99941228]
  [ 0.10189204  0.99478508  0.00454512  0.00886961]
  [ 0.2040601  -0.02537241  0.97862951 -0.03311219]]]
```

3.2 Linear 3D Points Estimation

In the linear_estimate_3d_point given corresponding points in different images, computing the 3D point is the best linear estimate. Image_points is the measured points in each of the M images. Camera_matrices is the camera projective matrices. Point_3d is the 3D point

```
def linear_estimate_3d_point(image_points, camera_matrices):
    # TODO: Implement this method!
    M = image_points.shape[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 = np.linalg.svd(A)
    point_3d = V[-1, :3] / V[-1, 3] # (x, y, z, w) -> (x/w, y/w, z/w)
    return point_3d
```

Output:

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

3.3 Non-Linear 3D Points Estimation

Since the linear estimation from SVD is not robust to noise, we need to further apply non-linear optimization to our estimated points. In this question, you need to implement the Gauss-Newton approach.

In the reprojection_error, given a 3D point and its corresponding points in the image planes, computing the reprojection error vector and associated Jacobian.

```python
def reprojection_error(point_3d, image_points, camera_matrices):
    # TODO: Implement this method!
    point_3d = np.hstack((point_3d, 1)) # (x, y, z) -> (x, y, z, 1)
    error = []
    for i in range( image_points.shape[0]):
        pi = image_points[i, :]
        P = camera_matrices[i, :, :]
        y = np.dot(P, point_3d)
        pi_prime = 1.0/y[2] * np.array([y[0], y[1]])
        error_item = pi_prime - pi
        error.append(error_item[0])
        error.append(error_item[1])
    error = np.array(error)
    return error
```

```python
def jacobian(point_3d, camera_matrices):
    # TODO: Implement this method!
    M = camera_matrices.shape[0]
    P1, P2, P3 = point_3d[0], point_3d[1], point_3d[2]
    jacobian = np.zeros((2*M, 3))

    for i in range(M):
        P = camera_matrices[i, :, :]
        y1 = P[0, 0] * P1 + P[0, 1] * P2 + P[0, 2] * P3 + P[0, 3]
        y2 = P[1, 0] * P1 + P[1, 1] * P2 + P[1, 2] * P3 + P[1, 3]
        y3 = P[2, 0] * P1 + P[2, 1] * P2 + P[2, 2] * P3 + P[2, 3]
        jacobian[2*i, 0] = (y3 * P[0, 0] - y1 * P[2, 0]) / y3**2
        jacobian[2*i, 1] = (y3 * P[0, 1] - y1 * P[2, 1]) / y3**2
        jacobian[2*i, 2] = (y3 * P[0, 2] - y1 * P[2, 2]) / y3**2
        jacobian[2*i+1, 0] = (y3 * P[1, 0] - y2 * P[2, 0]) / y3**2
        jacobian[2*i+1, 1] = (y3 * P[1, 1] - y2 * P[2, 1]) / y3**2
        jacobian[2*i+1, 2] = (y3 * P[1, 2] - y2 * P[2, 2]) / y3**2

    return jacobian
```

Output:

```
--------------------------------------------------------------------------------
Part C: Check that the difference from expected error/Jacobian
is near zero
--------------------------------------------------------------------------------
Error Difference:  8.301299988565727e-07
Jacobian Difference:  1.817115702351657e-08
--------------------------------------------------------------------------------
```

In the nonlinear_estimate_3d_point, given corresponding points in different images, computing the 3D point that iteratively updates the points.

```python
def nonlinear_estimate_3d_point( image_points, camera_matrices):
    # TODO: Implement this method!
    point_3d = linear_estimate_3d_point( image_points, camera_matrices)
    error = reprojection_error(point_3d, image_points, camera_matrices)
    for i in range(10):
        J = jacobian(point_3d, camera_matrices)
        point_3d = point_3d - np.dot(np.linalg.inv(np.dot(J.T, J)), np.dot(J.T, error))
        error = reprojection_error(point_3d, image_points, camera_matrices)
    return point_3d
```

Output:

```
--------------------------------------------------------------------------------
Part D: Check that the reprojection error from nonlinear method
is lower than linear method
--------------------------------------------------------------------------------
Linear method error: 98.7354235689419
Nonlinear method error: 95.59481784846031
--------------------------------------------------------------------------------
```

3.4 Decide the Correct RT

In the estimate_RT_from_E, we can compute the relative RT between the two cameras from the Essential Matrix. K is the intrinsic camera matrix. E is the Essential Matrix between the two cameras. image_points are measured points in each of the M images.

```python
def estimate_RT_from_E(E, image_points, K):
    # TODO: Implement this method!
    RT = estimate_initial_RT(E)
    max_points = 0
    for i in range(4):
        positive_count = 0
        RT_i = RT[i, :, :]
        I = np.array([
            [1.0, 0.0, 0.0, 0.0],
            [0.0, 1.0, 0.0, 0.0],
            [0.0, 0.0, 1.0, 0.0]
        ])
        c1 = np.dot(K, I)
        c2 = np.dot(K, RT_i) # (3, 4)
        camera_matrices = np.zeros((2, 3, 4))
        camera_matrices[0, :, :] = c1
        camera_matrices[1, :, :] = c2
        for j in range( image_points.shape[0]):
            point_3d = nonlinear_estimate_3d_point( image_points[j], camera_matrices)
            point_3d_prime = camera1_to_camera2(point_3d, RT_i)
            if point_3d[2] > 0 and point_3d_prime[2] > 0:
                positive_count += 1
        if positive_count > max_points:
            max_points = positive_count
            correct_RT = RT_i

    return correct_RT

def camera1_to_camera2(P, RT):
    temp = np.ones((4, 1))
    temp[0:3, :] = P.reshape((3, 1))
    temp1 = RT[:, :3].T
    temp2 = -temp1.dot(RT[:, 3:])
    A = np.concatenate((temp1, temp2), axis=1)
    point_3d_prime = A.dot(temp)
    return point_3d_prime
```

Output:

```
--------------------------------------------------------------------------------
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:
 [[ 0.97364135 -0.09878708 -0.20558119  0.99941228]
 [ 0.10189204  0.99478508  0.00454512 -0.00886961]
 [ 0.2040601  -0.02537241  0.97862951  0.03311219]]
--------------------------------------------------------------------------------
```

3.5 Result