DATA.ML.300 Computer Vision Exercise 6

Uyen Phan

02 2024

1

a) The two matrices are used in determining the structure and motion of objects in a scene.

The essential matrix E encodes information about the position and orientation of the cameras within the environment, while the fundamental matrix F relates the location of the point in the two images using pixel coordinates. Therefore, the fundamental matrix is used for uncalibrated cameras, while the essential matrix is used for calibrated cameras.

b) If we know the calibration matrices of the two cameras, we can estimate the essential matrix from the fundamental matrix: $E = K'^T F K$.

c) DoF of F is 7. Although F has 9 entries, the zero value of determinant and the use of homogeneous coordinates decrease the degrees of freedom.

d) DoF of E is 5. A rotation has 3 degrees of freedom and a translation 3. Since the scale of the essential matrix does not matter, it has 5 degrees of freedom.

2

Implementation code:

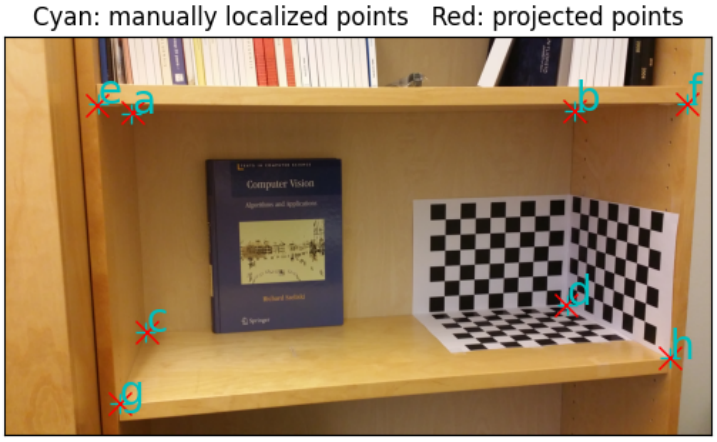
```
    ##-your-code-starts-here-##
A = []

    for i in range(len(x_world)):
        upper = [np.zeros(x_world[i].shape), x_world[i], -x_im[i, 1]*x_world[i]]
        upper = np.hstack(upper)
        lower = [x_world[i], np.zeros(x_world[i].shape), -x_im[i, 0]*x_world[i]]
        lower = np.hstack(lower)
        A.append(upper)
        A.append(lower)
```

```
A = np.array(A)
A_ = np.matmul(A.T, A)
eigenvalues, ev = np.linalg.eig(A_)
idx = np.argmin(eigenvalues)
P = ev[:,idx]

# Reshape the eigenvector into a projection matrix P
P = np.reshape(P, (3, 4))
##-your-code-starts-here-##
```

Plot for task 2:



3

Implementation code in trianglin.py:

```
##-your-code-starts-here-##
x1_cross_P1 = np.dot(np.array([[0, -1, x1[1]],
                      [1, 0, -x1[0]],
                      [-x1[1], x1[0], 0]]), P1)
x2_cross_P2 = np.dot(np.array([[0, -1, x2[1]],
                      [1, 0, -x2[0]],
                      [-x2[1], x2[0], 0]]), P2)

# Stack the cross-product matrices vertically to form matrix A
A = np.vstack([x1_cross_P1, x2_cross_P2])

A_ = np.matmul(A.T, A)
eigenvalues, ev = np.linalg.eig(A_)
idx = np.argmin(eigenvalues)
X_homogeneous = ev[:,idx]

# Convert homogeneous coordinates to cartesian coordinates
X = X_homogeneous[:-1] / X_homogeneous[-1]
##-your-code-ends-here-##
```

Implementation code in triangulation_task.py:

```
##-your-code-starts-here-##
picture_w_mm = np.linalg.norm(L-M)
picture_h_mm = np.linalg.norm(L-N)
##-your-code-ends-here-##
```

Result for task 3:

```
Picture width: 139.74 mm
Picture height: 107.02 mm
```