LAB 2: EPIPOLAR GEOMETRY DUE 8TH NOVEMBER

Task 1 (25) Compute Fundamental Matrix

You are given two sets of 2D points in homogeneous coordinates that correspond to correspoinding points in images kronan1.JPG and kronan2.jpg. This points are in compEx1data.mat, you can use scipy.io. For example

x = io.loadmat("data/task3/compEx1data.mat")['x']

xIm1 = x[0,0]
xIm2 = x[1,0]

The first task for this to compute Fundamental Matrix from this corresponding points, and display the epiplolar lines for a subset of this points, let's say some 20 points.

Using the constraint of the fundamental matrix, $x'^{T}Fx = 0$, set a system of linear equations and solve it with the SVD method discussed in class.

$$\begin{pmatrix} u' & v' & 1 \end{pmatrix} \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = 0$$

The linear squares estimate of F is full rank; however, the fundamental matrix is a rank 2 matrix. As such we must reduce its rank. In order to do this we can decompose F using singular value decomposition into the matrices U S V = F. We can then estimate a rank 2 matrix by setting the smallest singular value in S to zero thus generating S2 . The fundamental matrix is then easily calculated as F = U S2 V. Again, you have more details in the lecture notes.

Task 2 (15) Draw Epipolar Lines

Write a function to plot the epipolar lines to check your fundamental matrix estimation. The epipolar line is defined by multiplying the given point by F.

IMAGE OF THE 50-80 POINTS USING THE F MATRIX RESULT FROM TASK 3.

l = Fx'. l = [a,b,c] meaning the line equation ax+by+c = 0
Evaluate the line for x = 0 and x = image.width, and draw the line on the top of the image. This is conveniently done in python using PIL, Image, and ImageDraw.

NOTE1: Draw the epipolar lines and the points for a subset of points. The result will not be very good, because we didn't normalize the points, so your epipolar lines might not cross the points, although they should be fairly close.
NOTE2: The data set contains many points and not all of them are good correspondences, so even with the normalization in task 3, you might get some error. You can tray to use a subset of the points and calculate F. At least for those points, the result should be better.

Task 3 (10) Compute Fundamental Matrix with Normalization

Normalize your 2D points before computing you matrix F. After normalization your data should have the centroid in the origin (0,0) and an average square distance of sqrt(2). This is easily achieved mean std functions. Note that you have to find this transformation as a 3x3 matrix T. Then compute your matrix F in the same way than before. As a later stage, you need to invert the effect

Objective

Given $n \geq 8$ image point correspondences $\{\mathbf{x}_i \leftrightarrow \mathbf{x}'_i\}$, determine the fundamental matrix F such that $\mathbf{x}'^{\mathsf{T}}_i \mathbf{F} \mathbf{x}_i = 0$.

Algorithm

(i) **Normalization:** Transform the image coordinates according to $\hat{\mathbf{x}}_i = \mathbf{T}\mathbf{x}_i$ and $\hat{\mathbf{x}}'_i = \mathbf{T}'\mathbf{x}'_i$, where T and T' are normalizing transformations consisting of a translation and scaling.

(ii) Find the fundamental matrix $\hat{\mathbf{F}}'$ corresponding to the matches $\hat{\mathbf{x}}_i \leftrightarrow \hat{\mathbf{x}}'_i$ by

  (a) **Linear solution:** Determine $\hat{\mathbf{F}}$ from the singular vector corresponding to the smallest singular value of $\hat{\mathbf{A}}$, where $\hat{\mathbf{A}}$ is composed from the matches $\hat{\mathbf{x}}_i \leftrightarrow \hat{\mathbf{x}}'_i$ as defined in (11.3).

  (b) **Constraint enforcement:** Replace $\hat{\mathbf{F}}$ by $\hat{\mathbf{F}}'$ such that $\det \hat{\mathbf{F}}' = 0$ using the SVD (see section 11.1.1).

(iii) **Denormalization:** Set $\mathbf{F} = \mathbf{T}'^{\mathsf{T}} \hat{\mathbf{F}}' \mathbf{T}$. Matrix F is the fundamental matrix corresponding to the original data $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$.

of the normalization. The references in this text refer to the book, multiview-geometry.

Task 4 (25) The Essential Matrix

The file compEx3data.mat contains the calibration matrix K for the two images in Tasks 1&2. Normalize the image points using the inverse of K. Set a linear system like in task 1 in this case for $x'^{\mathsf{T}} E x = 0$. This is the same as in figure one, but with points in normalized coordinates.

The essential matrix has rank 2 and the other two singular values are the same.Similarly to the case of task 1, the least square solution of the system will not fulfil this, so we need to adjust the Essential matrix, in this case we follow this solution:

Let E *be a* 3×3 *matrix with SVD given by* $E = UDV^T$, *where* $D = \text{diag}(a,b,c)$ *with* $a \geq b \geq c$. *Then the closest essential matrix to* E *is given by* $\hat{E} = U\hat{D}V^T$, *where* $\hat{D} = \text{diag}((a+b)/2, (a+b)/2, 0)$.

From the essential matrix, we can compute our new fundamental matrix by premultiplying and post multiplying by K. Note this in this case, K' and K are the same , since they were taken with the same image.

$$F = K'^{-T} E K^{-1} \qquad E = K'^{T} F K.$$

Draw the epipolar lines for the new F.

Task 5 (25): Computing P and P' and Triangulation

As we saw in class, from E we can compute P and P', i.e. a pair of possible projection matrices for both cameras, that explain the corresponding 2D points and the epipolar constraint. The following camera matrices correspond to the normalized points, but we will use these for triangulation.

P1 = [I | 0]

P2 = $[UWV^T \mid u_3]$ or $[UWV^T \mid -u_3]$ or $[UW^TV^T \mid u_3]$ or $[UW^TV^T \mid -u_3]$

where u3 is the third column of U and W is the following:

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Now we need to triangulate the points and find if they lie in front of the camera, so we can choose the correct solution for P2.

To triangulate the points, you need to create a linear system for each point and solve for X,Y,Z,W using the following equations.

$$\begin{aligned} x(\mathbf{p}^{3T}\mathbf{X}) - (\mathbf{p}^{1T}\mathbf{X}) &= 0 \\ y(\mathbf{p}^{3T}\mathbf{X}) - (\mathbf{p}^{2T}\mathbf{X}) &= 0 \end{aligned} \qquad \mathbf{A} = \begin{bmatrix} x\mathbf{p}^{3T} - \mathbf{p}^{1T} \\ y\mathbf{p}^{3T} - \mathbf{p}^{2T} \\ x'\mathbf{p}'^{3T} - \mathbf{p}'^{1T} \\ y'\mathbf{p}'^{3T} - \mathbf{p}'^{2T} \end{bmatrix}$$

where p^{iT} are the rows of P. P the corresponding P1 or P2 from the previous part of the task. X is the 3D point coordinates, including W that we want to find, and x and y , are the normalized coordinates of our 2d points. These equations are *linear* in the components of X. This is a redundant set of equations, since the solution is determined only up to scale. We will solve it in a similar fashion to the previous cases, using SVD for AX = 0.

You can check which is the correct solution for P2 by triangulating one point and make your selection, then triangulate the rest of the points. You can also check for all the points, and chose the camera with more points in front of the cameras, since the data is noisy and not all the points might be in the correct place. If you do so, you can reject the remaining points.

Once you have your solution selected, and your 3D points, you can use the attached code to create a ply file and visualize it using meshlab . http://meshlab.sourceforge.net/

```
from plyfile import PlyData, PlyElement

points = np.array(zip(XX[:,0].ravel(), XX[:,1].ravel(), XX[:,2].ravel()),dtype=[('x','f4'), ('y',
'f4'),('z', 'f4')])
el = PlyElement.describe(points, 'vertex')
PlyData([el]).write('pc.ply')
```

Where XX would be an array with your 3D points with shape (numberOfPoints, 3)

Extra: Reproject and measure error.

Convert your P1 and P2 to full projection matrix multiplying by K. Re-project the calculated 3D points in both views and measure the error as in Lab1.

Give color to the point cloud.

```
 points = np.array(zip(x.ravel(), y.ravel(), z.ravel(),r.ravel(), g.ravel(), b.ravel()),dtype=[('x',
'f4'), ('y', 'f4'),('z', 'f4'),('red', 'u1'), ('green', 'u1'),('blue', 'u1')])
```