Introduction to Computer Vision

Assignment #3

Due: Dec.-19 (Fri.) (before 17:59pm) No late submission allowed.

Instruction

a. Submit your source codes, report, and descriptor in a single compressed file "ICV_A3_*StudentID*.zip" to iCampus.

b. Python 3.7 or higher / OpenCV 3.4 or higher will be used to execute your submitted codes.

c. You can submit at most 13 files.

   - Mandatory files: *A3_Fmat.py, A3_StudentID.des, A3_compute_descriptors.py, A3_report.pdf*

d. Any work that you turn in should be your own.

Part #1. Fundamental Matrix [40 pts]

The requirements of Part #1 will be evaluated by running '***A3_Fmat.py***' file.

1-1. Fundamental matrix computation

a) Load two images 'temple1.png' and 'temple2.png'. The feature correspondences between two images are also provided in 'temple_matches.txt' file. You can use `np.loadtxt()` function to load a text file:

```
M = np.loadtxt( 'temple_matches.txt' )
```

Each row of the text file gives one correspondence by 4 values (x_1, y_1, x_2, y_2) describing that a feature point (x_1, y_1) of the first image is matched to (x_2, y_2) of the second image. You can utilize those correspondences to compute the fundamental matrix.

b) Implement the Eight-point algorithm to compute the fundanmental matrix:

```
function F = compute_F_raw ( M )
```

Refer the lecture material (page #63 – #70 of '08_Two-View_Geometry.pdf').

c) Implement the Eight-point algorithm with a normalization:

```
function F = compute_F_norm ( M )
```

One simple normalization scheme indepedent from the feature locations is recommended:

- Translation to move the image center to origin $(0,0)$
- Scaling to fit the image into an unit square $[\,(-1,-1),(+1,+1)\,]$

You need to remember the normalization and un-normalization should be reflected to your fundanmental matrix.

d) Implement your own algorithm to compute the fundanmental matrix:

```
function F = compute_F_mine ( ... )
```

- It should return the result within 5 seconds.

e) Print the average reprojection errors of three functions implemented in (b), (c), and (d) to the console. In order to compute the error, you are required to use the given function in 'compute_avg_reproj_error.py' file. Refer the following example:

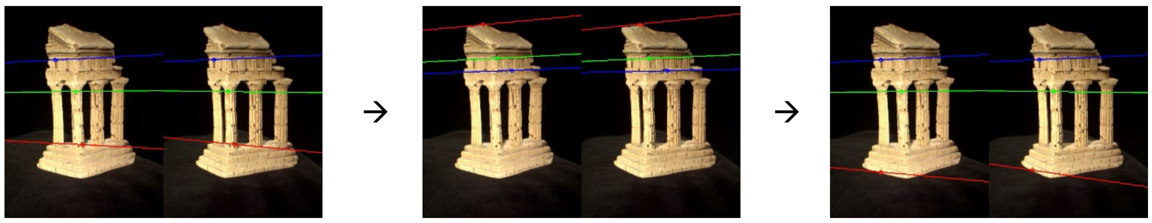
```
Average Reprojection Errors (temple1.png and temple2.png)
   Raw  = 0.8050090158803835
   Norm = 0.35460509632975395
   Mine = 0.3510098327769134
```

f) Your script should produce the results of (e) for the following three pairs of images:
('temple1.png' / 'temple2.png') , ('house1.jpg' / 'house2.jpg'), and ('library1.jpg' / 'library2.jpg')

1-2. Visualization of epipolar lines

a) Implement a script that performs the followings:

- Randomly select 3 correspondances: $(p_1 \leftrightarrow q_1)$, $(p_2 \leftrightarrow q_2)$, and $(p_3 \leftrightarrow q_3)$
- Compute 6 epipolar lines $l_1, l_2, l_3, m_1, m_2, m_3$ corresponding to $p_1, p_2, p_3, q_1, q_2, q_3$.
- Visualize p_1, q_1, l_1, m_1 as red, p_2, q_2, l_2, m_2 as green, and p_3, q_3, l_3, m_3 as blue.
- If any key except 'q' is pressed, then you should repeat the above process.
- If 'q' is pressed, then you need to close the window and finish the visualization.



b) Your script should run (a) for the following three pairs of images:
('temple1.png' / 'temple2.png') , ('house1.jpg' / 'house2.jpg'), and ('library1.jpg' / 'library2.jpg')

c) Note that, you need to use the fundamental matrix computed by the function implemented in 1-1-d).

Part #2. Image Retrieval [60 pts]

The requirements will be evaluated based on your descriptor file, ‘***A3_StudentID.des’***

1-1. Dataset: a subset of UKBench

a) In this assignment, you are provided with pre-extracted features for 2,000 images in a compressed file named *A3_P2_Features.zip*. This file includes two types of features: SIFT features (*0000.sift – 1999.sift*) and CNN features (*0000.cnn – 1999.cnn*), stored respectively in the directories ‘*./features/sift/*’ and ‘*./features/cnn/*’. Your Python scripts, *.py* files, should access these features using relative directory paths. <u>Please ensure not to include these features in your submission.</u>

b) Each (*xxxx.sift*) file adheres to the following binary file format:

$$\begin{bmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,128} \\ s_{2,1} & s_{2,2} & & s_{2,128} \\ \vdots & & \ddots & \vdots \\ s_{n,1} & s_{n,2} & \cdots & s_{n,128} \end{bmatrix}$$

The image is represented by n SIFT features with each SIFT descriptor comprising a 128-dimensional vector. Note that, the value of n may vary among different images. Each element $s_{i,j}$ is stored in the ‘**unsigned char**’ format (1 byte representing a range from 0 to 255).

c) On the other hand, each CNN feature (*xxxx.cnn*) follows this binary file format:

$$\begin{bmatrix} v_{1,1,1} & v_{1,1,2} & \cdots & v_{1,1,512} \\ v_{1,2,1} & v_{1,2,2} & & v_{1,2,512} \\ \vdots & & \ddots & \vdots \\ v_{14,14,1} & v_{14,14,2} & \cdots & v_{14,14,512} \end{bmatrix}$$

The file includes a feature map sized 14×14×512 (width×height×channel). Each element $v_{i,j,k}$ is stored in the ‘float’ format (4 bytes representing a real value).

d) For each image, there are 4 relevant items, including the image itself. Those 4 images constitute the ground truth retrieval results.

e) Additionally, 100 sample features along with their corresponding images are provided in the *A3_P2_Samples.zip* file. You can use these to verify and debug your implementations.

- *./samples/images/s_0000.jpg – s_0099.jpg*
  ./samples/features/cnn/s_0000.cnn – s_0099.cnn
  ./samples/features/sift/s_0000.sift – s_0099.sift

1-2. Task: computing image descriptors for the retrieval task

a) Your task is to compute image descriptors that reflect similarities among images using L_1 or L_2 distances. You are asked to store the descriptors in a <u>binary file format</u> as follows:

$$\boxed{\begin{array}{l} N\ D \\ d_{1,1}\ d_{1,2} \ldots d_{1,D} \\ d_{2,1}\ d_{2,2} \ldots d_{2,D} \\ \ldots \\ d_{N,1}\ d_{N,2} \ldots d_{N,D} \end{array}}$$

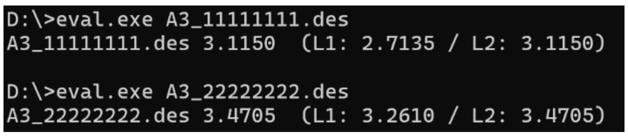
, where N is the number of images (=2000) and D is the dimensionality of each descriptor. Here, $d_{i,j}$ represents the j^{th} element of the i^{th} descriptor. The required data types of variables are summarized as follows:

Variable	Type
N	Signed 32 bits integer (***int*** in C/C++)
D	Signed 32 bits integer (***int*** in C/C++)
$d_{i,j}$	32 bits floating point (***float*** in C/C++)

Note that, the dimensionality of your descriptor cannot exceed 4,096 (i.e. $D \leq 4096$). Consequently, the size of your descriptor file '***A3_StudentID.des'*** can be up to 32,768,008 bytes ($= 4 + 4 + 2000 \times 4096 \times 4$ bytes).

b) You are required to submit a script named '***A3_compute_descriptors.py***' that produces the descriptors. This script should be executable from any directory that includes a '*./features/*' sub-directory, by utilizing relative directory paths for data access. If your algorithm relies on codeword vectors from clustering, which involves extensive computation time, you must precompute these vectors, store them, and load them within your script. In this case, you must include these precomputed files in your submission package.

c) Once you have generated the descriptor file, you can evaluate your descriptors by using provided executable '*eval.exe*'. The program assesses your descriptor based on both L_1 and L_2 distances, selecting the higher of the two for evaluation. Note that, that '*eval.exe*' is compatible only with Windows OS. When using the program, specify your descriptor file name as demonstrated in the following example:



```
D:\>eval.exe A3_11111111.des
A3_11111111.des 3.1150  (L1: 2.7135 / L2: 3.1150)

D:\>eval.exe A3_22222222.des
A3_22222222.des 3.4705  (L1: 3.2610 / L2: 3.4705)
```

Note that, the accuracy ranges from 0 to 4.

d) You are required to submit a report titled '***A3_report.pdf***' that describes your method for computing the image descriptors. The report should be longer than 1 page but less than 3 pages.

e) It is strictly prohibited to manipulate the generation of image descriptors by exploiting the characteristics of the dataset. A common example of such abuse is assigning identical values across multiple descriptors, such as setting $d_1 = d_2 = d_3 = d_4 = [1,0,0,\ldots,0]$, $d_5 = d_6 = d_7 = d_8 = [0,1,0,\ldots,0]$.