Assignment- 3 Documentation

Group- 7
Soumava Paul (16EE10056)
Swagatam Haldar (16EE10063)

- In this assignment, two images of the same scene captured with different views were given and we were to calculate different parameters (*matching key-points*, *epipolar lines*, *epipoles, camera matrices, scene depth*) pertaining to Two view geometry.
- The auxiliary helper functions are written in the file **utils.py** and the main tasks are implemented in the **modules.py** file.
- The user is required to interact with the **main.py** file only where we sequentially display the tasks and their results. The details of the functions follow.

In `utils.py` file-

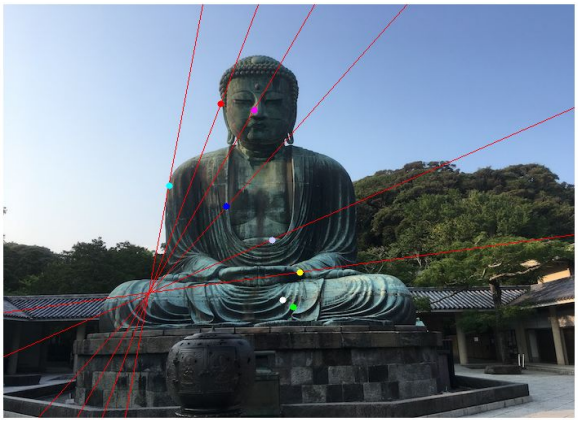
1. **`def brute_force_matcher(img1, img2, kp1, kp2, des1, des2)`**- Computes 8 pairs of keypoints among the 2 images having the top 8 least distances. This is done by a simple brute force search (hence the name) using the keypoints and feature descriptors of each image found by SIFT or SURF.
2. **`def draw_epipolar_lines(X1, X2, F, img1, img2, save_flag)`:-** Takes the 8 matching keypoint sets from the two images and the fundamental matrix F. Using F, for each key-point in one image it computes the epipolar line which will contain the same image in the other line using the point to line correlations **Fx** (and $\mathbf{F^Tx'}$ respectively). Then it draws onto the image the lines in line-intercept form using OpenCV line function. It also computes the epipoles from the intersection of any two of the epipolar lines
3. **`def give_scene_points(X1, X2, P1, P2)`**- Takes the 8 matching keypoints and the two camera matrices. For each matching keypoint pair computes the matrix A and solves the equation **AX = 0 (**by **min (||AX||)** where X = 3D coordinate (x,y,z,w) of the scene constrained to **||X|| = 1**. To solve the equation, we use SVD decomposition of A and find the solution in the last column of V. The final coordinate is obtained by dividing it by the fourth coordinate.

In `modules.py` file-

1. **`def compute_SIFT_ckp(img1, img2, save_flag)`**- Takes the 2 input images and computes the best 8 pairs of keypoints using the SIFT and a brute force matching algorithm. This is one of the 2 choices available for finding the 8 keypoints and the fundamental, camera matrices, etc. hence.
2. **`def compute_SURF_ckp(img1, img2, save_flag)`:-** Takes the 2 input images and computes the best 8 pairs of keypoints using the SURF and a brute force matching

algorithm. This is one of the 2 choices available for finding the 8 keypoints and the fundamental, camera matrices, etc. hence.

3. **`def solve_8pt_corr(X1, X2):`** – Computes the fundamental matrix using 8 point correspondences. It uses the identity $\mathbf{x'^{T}Fx = 0}$ which gives one equation in the elements of F as $x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0$ which is solved for 8 variables by assuming $f_{33}= 1$. Thus forming the equation $\mathbf{Af = [-1]_{8x1}}$. Since 8 points are different A is a rank = 8 matrix and can be solved by inverting A. However to ensure the singularity constraint i.e **det F = 0,** we use a result which minimizes the Frobenius norm between **F** (obtained using Af = [-1]) and **F'** (closest to F with **det F' = 0**). Using SVD, $\mathbf{F = UDV^{T}}$ and if D = diag(r,s,t) with r>=s>=t then $\mathbf{F' = Udiag(r,s,0)V^{T}}$. (Ref- Hartley, Zissermann Chapter 11.1.1)
4. **`def calc_epipoles_from_F(F):-`** The left and right epipoles are computed by solving the homogeneous rank deficient equations **Fe = 0** and $\mathbf{F^{T}e' = 0}$. They are first solved for (e1, e2) and then appended with 1 to represent in homogeneous coordinates.
5. **`def estimate_proj_matrices(F, e_):-`** Takes the fundamental matrix F and Right epipole e_ and computes the two matrices by assuming **P = [I | 0]** and **P' = [SF | e_]**. Here $\mathbf{S = [e']_{x}f}$. We also do a compatibility check to ensure that the camera matrices thus obtained satisfy $\mathbf{P'^{T}FP}$ = a skew-symmetric matrix.
6. **`def estimate_scene_depth(img1, img2, X1, X2, P1, P2, save_flag):-`** Computes the 3D coordinates of the 8 matched points using the function **`give_scene_points(X1, X2, P1, P2)`** and marks each point with a unique color to display their depths as shown in the figure below.



```
Estimating scene point depths . . .

Depth of Red feature point = 0.14598639093089538
Depth of Yellow feature point = 0.06260831660215471
Depth of Green feature point = 0.057077161154661056
Depth of Indigo feature point = 0.09912299807731124
Depth of Blue feature point = 0.08419628113891668
Depth of Pink feature point = 0.13190778674531436
Depth of Light Pink feature point = 0.07112965920861355
Depth of White feature point = 0.058742696874321376
(adipEnv3) swagatam@swagatam-hp:~/cpp_test/wwsIP/practice/Adv
Compatibility test for P, P' and F
```

Observations

As it can be readily inferred from the above result or user's own terminal interface, the red feature point (near the right ear of the Buddha) has the maximum depth and the green point on his clothing has the minimum depth. All other points have intermediate depths and their relative depths also concur with our intuitions.

Running Instructions

From the terminal, run the `main.py` file as **`python3 main.py -save 0/1 -dt 0/1`**

The **save** flag is left to the user to decide. (1 - all images get saved to disk, 0 - none get saved. Defaults to 1.) All intermediate images will always be shown as the code executes and the user can just press Esc when he's done seeing one. (Note: Sometimes a pair can also be displayed instead of just a single one.)

The **dt** flag is for the feature descriptor algorithm (SIFT(0) or SURF(1)). Defaults to 0.

Packages Required

Python3, NumPy, cv2 (We have used version 3.4.2, other versions might also work, however, SIFT and SURF are usually not available for free use under some of the recent versions of opencv-contrib).