Project 2 of CSE 473/573

Due on Monday class of Nov. 05 (350PM)

Guidelines

1. Please write your programs using Python. Both Python 2 and Python 3 are OK.
2. For all tasks (except for task 3.0), you can use the Numpy, Matplotlib, OpenCV libraries (no extra ones are allowed). For task 3.0, you should only use Numpy basic routines for the k-means algorithm. OpenCV version 3.2+ is required (for non-free SIFT). Please make sure that `cv2.xfeatures2d.SIFT_create` works.
3. Images for all three tasks will be uploaded to Piazza. You should ONLY use the images uploaded to Piazza to test you programs and obtain results that you are to include in your report.
4. `cv2.imshow` and `plt.imshow` are NOT allowed, please use `cv2.imwrite, plt.imsave` or `plt.savefig` to save your results to be included in your report. Running `python <scriptname>.py` should generate all the required result images without human interaction or display window prompt.
5. Your plots in your report should be reproducible. I.e. the random seed should be fixed. Add "`UBIT = '<your UBIT name>'; import numpy as np; np.random.seed(sum([ord(c) for c in UBIT]))`" to the top of the python script. Change `<your UBIT name>` to your actual UBIT name.
6. Submit a project report of up to 15 pages (hard copy) by the due date. In the report, please provide the image results and the source code. Please upload the source code before the due date to Piazza for TA's check. Your submission requirement is the same as HW1 and PROJ1.
7. You need to work on this project independently and plagiarism will be penalized.

1 Image Features and Homography (5pt)

1. Given two images `mountain1.jpg` and `mountain2.jpg`, extract SIFT features and draw the keypoints for both images. Include the resulting two images (`task1_sift1.jpg, task1_sift2.jpg`) in the report. (1pt)

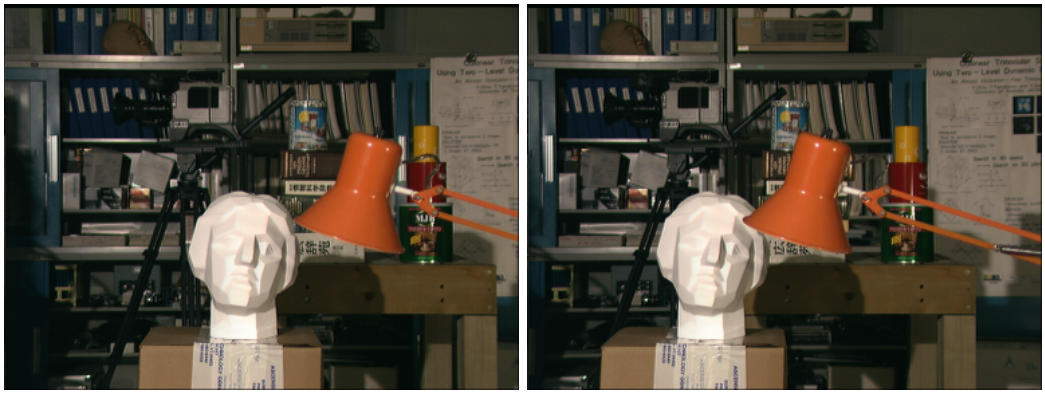
2. Match the keypoints using k-nearest neighbour (k=2), i.e., for a keypoint in the left image, finding the best 2 matches in the right image. Filter good matches satisfy `m.distance` $<$ `0.75 n.distance`, where `m` is the first match and `n` is the second match. Draw the match image using `cv2.drawMatches` for all matches (your match image should contain both inliers and outliers). Include the result image (`task1_matches_knn.jpg`) in the report. (1pt)

3. Compute the homography matrix H (with RANSAC) from the first image to the second image. Include the matrix values in the report. (1pt)

4. Draw the match image for around 10 random matches **using only inliers**. Include the result image (`task1_matches.jpg`) in the report. (1pt)

5. Warp the first image to the second image using H. The resulting image should contain all pixels in `mountain1.jpg` and `mountain2.jpg`. Include the result image (`task1_pano.jpg`) in the report. (1pt)

2 Epipolar Geometry (5 pt)



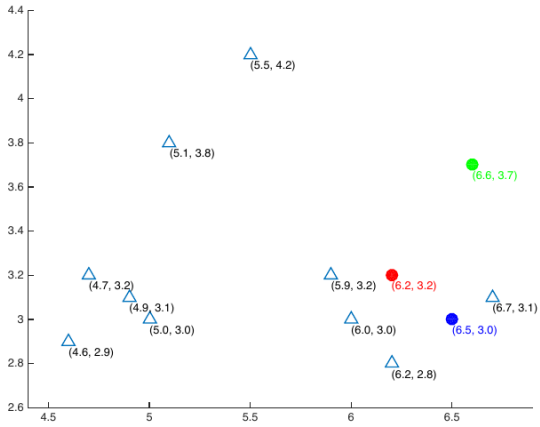
1. Given two images `tsucuba_left.png` and `tsucuba_right.png`, do the same process for Task 1.1 and 1.2. Include the three images (2 for task 1.1 and 1 for task 1.2) (`task2_sift1.jpg, task2_sift2.jpg, task2_matches_knn.jpg`) in the report. (1pt)

2. Computer the fundamental matrix F (with RANSAC). Include the matrix values in the report. (1pt)

3. Randomly select 10 **inlier** match pairs. For each keypoint in the left image, compute the epiline and draw on the right image. For each keypoint in the right image, compute the epiline and draw on the left image [Using different colors for different match pairs, but the same color for epilines on the left and right images with the same match pair.] Include two images (`task2_epi_right.jpg`, `task2_epi_left.jpg`) with epilines in the report. (2pt)

4. Compute the disparity map for `tsucuba_left.png` and `tsucuba_right.png`. Include the disparity image (`task2_disparity.jpg`) in the report. (1pt)

3 K-means Clustering (5 + 3 pt)



$$X = \begin{bmatrix} 5.9 & 3.2 \\ 4.6 & 2.9 \\ 6.2 & 2.8 \\ 4.7 & 3.2 \\ 5.5 & 4.2 \\ 5.0 & 3.0 \\ 4.9 & 3.1 \\ 6.7 & 3.1 \\ 5.1 & 3.8 \\ 6.0 & 3.0 \end{bmatrix}$$

Given the matrix X whose rows represent different data points, you are asked to perform a k-means clustering on this dataset using the Euclidean distance as the distance function. Here k is chosen as 3. All data in X were plotted in above Figure. The centers of 3 clusters were initialized as $\mu_1 = (6.2, 3.2)$ (red), $\mu_2 = (6.6, 3.7)$ (green), $\mu_3 = (6.5, 3.0)$ (blue).

Implement the k-means clustering algorithm (**you are only allowed to use the basic numpy routines to implement the algorithm**).

1. Classify $N = 10$ samples according to nearest $\mu_i (i = 1, 2, 3)$. Plot the results by coloring the empty triangles in red, blue or green. Include the classification vector and the classification plot (`task3_iter1_a.jpg`) in the report. (1pt)

   (a) [Hint:] Using `plt.scatter` with edgecolor, facecolor, marker and `plt.text` to plot the figure.

2. Recompute μ_i. Plot the updated μ_i in solid circle in red, blue, and green respectively. Include the updated μ_i values and the plot in the report (`task3_iter1_b.jpg`). (1pt)

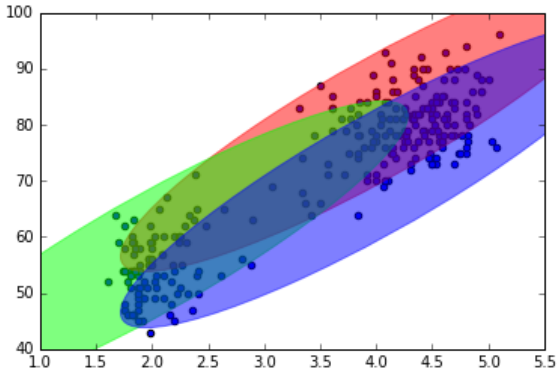
3. For a second iteration, plot the classification plot and updated μ_i plot for the second iteration. Include the classification vector and updated μ_i values and these two plots (`task3_iter2_a.jpg`, `task3_iter2_b.jpg`) in the report. (1pt)

4. [Color Quantization] Apply k-means to image color quantization. Using only k colors to represent the image `baboon.jpg`. Include the color quantized images for $k = 3, 5, 10, 20$ (`task3_baboon_3.jpg`, `task3_baboon_5.jpg`, `task3_baboon_10.jpg`, `task3_baboon_20.jpg`). (2pt)

5. [Gaussian Mixture Model] Implement the Gaussian mixture models (GMM) (**you are only allowed to use the basic numpy routines and scipy.stats.multivariate_normal to implement the algorithm**). Your GMM algorithm should run on dataset represented as a matrix X of shape (N, D), each row represent a datapoint. N is the number of datapoints, and D is the dimension of the datapoints. (3 bonus points)

(a) Run GMM on the above dataset represented as a 10×2 matrix X. Let $\mu_1 = (6.2, 3.2)$, $\mu_2 = (6.6, 3.7)$, $\mu_3 = (6.5, 3.0)$, $\Sigma_1 = \Sigma_2 = \Sigma_3 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$, $\pi_1 = \pi_2 = \pi_3 = \frac{1}{3}$. What are the μ_i after the first iteration. Include the μ_i values in the report. (1 pt)

(b) Apply GMM to the Old Faithful dataset (https://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat). The dataset matrix X should be of shape $(272, 2)$ [x: eruptions, y: waiting]. Let $k = 3$, $\mu_1 = (4.0, 81)$, $\mu_2 = (2.0, 57)$, $\mu_3 = (4.0, 71)$, $\Sigma_1 = \Sigma_2 = \Sigma_3 = \begin{bmatrix} 1.30 & 13.98 \\ 13.98 & 184.82 \end{bmatrix} (= np.conv(X.T))$, $\pi_1 = \pi_2 = \pi_3 = \frac{1}{3}$. Plot the results for the first five iterations (The following image is a sample plotted with the given parameters at iteration 0, your reporting results should be similar but with different Gaussian mixture centers and covariances). Include these five plots (`task3_gmm_iter1.jpg, ..., task3_gmm_iter5.jpg`) in the report. (2 pt)



[Hint:] You can use the `plot_cov_ellipse` in
https://github.com/joferkington/oost_paper_code/blob/master/error_ellipse.py to plot the covariance ellipse. (Setting alpha=0.5, using red, green, blue for the three clusters.)