Introduction to Computer Vision (ECSE 415)
Assignment 2: Image Features

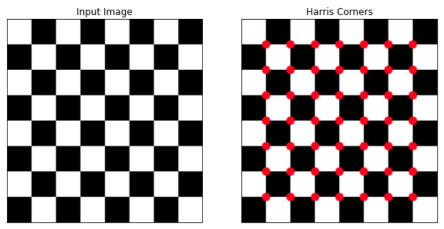
Due date: 11:59PM, February 22nd, 2021

For this assignment you will in a group of 2. Please submit your assignment solutions electronically via the myCourses assignment dropbox. More details on the format of the submission can be found below. Attempt all parts of this assignment. The assignment will be graded out of total of **90 points**. Students are expected to write their own code. (Academic integrity guidelines can be found at `https://www.mcgill.ca/students/srr/academicrights/integrity`). Assignments received up to 24 hours late will be penalized by 30%. Assignments received more than 24 hours late will not be graded.

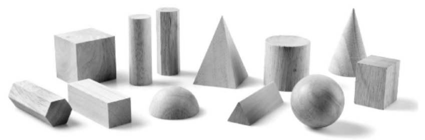
Submission Instructions

1. Submit two different jupyter notebooks (1 per person in a group) consisting of answers to 2 questions each from the assignment. All questions should be answered within these two notebooks. This will allow us to estimate the work done by each member of the group. Feel free to divide work as per your wish between members of the group. Submit one single zip file per group.
2. Comment your code appropriately.
3. Make sure that the submitted code is running without error. Add a README file if required.
4. If external libraries were used in your code, please specify their name and version in the README file.
5. Answers to reasoning questions should be comprehensive but concise.
6. Submissions that do not follow the format will be penalized 10%.

Note: For this assignment, we will work with RGB images unless specified otherwise. You are allowed to use OpenCV/Scikit-image functions for this assignment unless specified otherwise.



(a)



(b)

Figure 1: (a) checkerboard input image [source] and expected harris corner output (red dots represents detected harris corners) (b) shapes image [source] .

1 Harris Corner Detection (20 points)

Implement the Harris corner detector as described in class (lecture-5 slide-48 and tutorial-3) **primarily using NumPy** (you are prohibited from using OpenCV or Scikit-Image Harris corner detector function) **(10 points)**.

Note that you should convert all images to grayscale for this question.

Harris corner detector has the following steps:

1. Compute image derivatives using sobel operator (optionally, blur first).
2. Compute square of derivatives.
3. Apply gaussian filtering on the output of step-2.
4. Get cornerness function response on the output of step-3. (Determinant(H)-kTrace(H)2), where k=0.05. (You can vary value of k for your application).
5. Perform non-maxima suppression on the output of step-4 (the cornerness function).
6. Threshold output of Non-maxima suppression at a user defined threshold (T).

You will apply Harris Corner Detector for three different images:

1. Checkerboard image Figure:1(a) (Input image). Change the threshold value to get detected corners similar to Figure:1(a) (Harris Corner). Observe and report the effect of changing threshold values. Show the detected corners for at least three different threshold values (T). **(3 points)**
2. Shape image Figure:1(b). Try different values of thresholds and report your observations. Show detected corners for at least three different threshold values. **(3 points)**

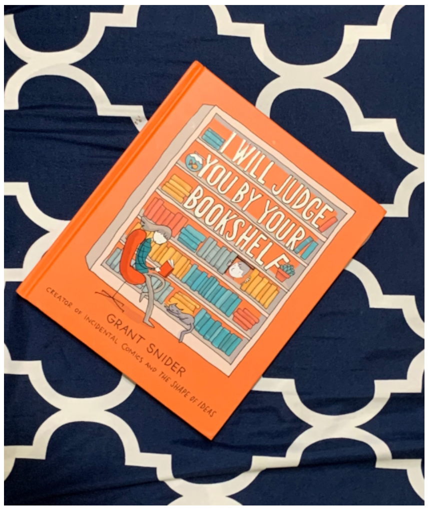

I WILL JUDGE YOU BY YOUR BOOKSHELF
GRANT SNIDER
CREATOR OF INCIDENTAL COMICS AND THE SHAPE OF IDEAS

Figure 2: Reference image of a book.

3. Take any one face image from Google Face thumbnail collection dataset. Apply the Harris corner detector on this image. Try different values of thresholds and report your observations. Show the detected corners for at least four different threshold values. **(4 points)**

2 Invariance of SIFT Features (26 Points)

You are given a reference image of a book as shown in Figure:2. Verify the invariance of SIFT features under changes in image scale and rotation. For this question, you are allowed to use in-built OpenCV/Scikit-Image functions.

2.1 Invariance Under Scale

1. Compute SIFT keypoints for the reference image. **(2 points)**
2. Scale reference image using scaling factors of (0.2, 0.8, 2, 5). This should result in a total of 4 different transformed images. Display scaled images. **(1 points)**
3. Compute SIFT keypoints for all (total 4) transformed images. **(2 points)**
4. Match all keypoints of the reference image to the transformed images using a brute-force method. **(2 points)**
5. Sort matching key points according to the matching distance. **(2 points)**

6. Display the top ten matched keypoints for each pair of the reference image and a transformed image. **(2 points)**

7. Plot the matching distance for the top 100 matched keypoints. Plot indices of keypoints on the x-axis and corresponding matching distance on the y-axis. **(1 points)**

8. Discuss the trend in the plotted results. What is the effect of increasing the scale on the matching distance? Reason the cause. **(1 points)**

2.2 Invariance Under Rotation

1. Compute SIFT keypoints for the reference image. **(2 points)**

2. Rotate reference image at the angle of (10, 90, 130, 180). Display rotated images. **(1 points)**

3. Compute SIFT keypoints for all (total 4) transformed images. **(2 points)**

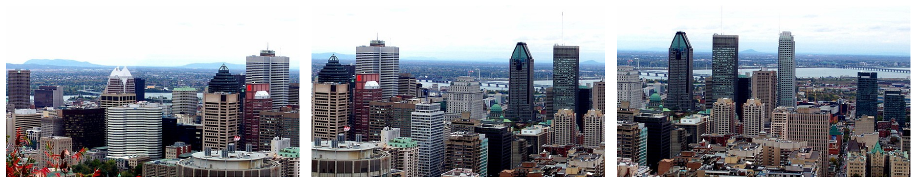
4. Match all keypoints of the reference image to the transformed images using a brute-force method. **(2 points)**

5. Sort matching keypoints according to the matching distance. **(2 points)**

6. Display top ten matched keypoints for each pair of the reference image and a transformed image. **(2 points)**

7. Plot the matching distance for the top 100 matched keypoints. Plot indices of keypoints on the x-axis and corresponding matching distance on the y-axis. **(1 points)**

8. Discuss the trend in the plotted results. What is the effect of increasing the angle of rotation on the matching distance? Reason the cause. **(1 points)**



(a) (b) (c)

Figure 3: Three different views of the same scene (a) 1Montreal (b) 2Montreal (c) 3Montreal [source]

3 Image Stitching (30 Points)

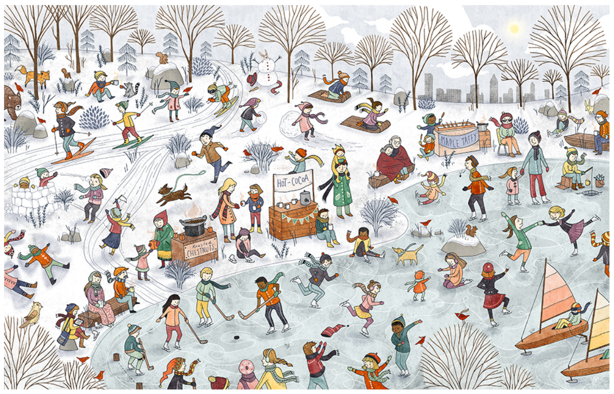
You are given three different views of the same scene in a folder 'stitching images' (Figure:3). Follow these steps in order to stitch given images:

1. Compute SIFT keypoints and corresponding descriptors for images 1Montreal and 2Montreal. **(2 points)**
2. Find matching keypoints in 1Montreal and 2Montreal images and display the 20 best pairs. **(4 points)**
3. Find homography using the RANSAC method and apply the transformation to 1Montreal. Image 2Montreal should not be transformed. **(5 points)**
4. Stitch transformed 1Montreal and the original 2Montreal together using linear image blending. Let us call this image 12Montreal. Display this image. **(4 points)**
5. Compute SIFT keypoints and corresponding descriptors for images 12Montreal and 3Montreal. **(2 points)**
6. Find matching keypoints in 12Montreal and 3Montreal images and display the 20 best pairs. **(4 points)**
7. Compute the homography using the RANSAC method. Apply the transformation to 3Montreal. Image 12Montreal should not be transformed. **(5 points)**
8. Stitch transformed 3Montreal and 12Montreal together using linear image blending. Display the resulting image. **(4 points)**

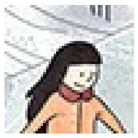
4 Template Matching (14 Points)

In this problem, we will look at how to find a template pattern in a cluttered scene. To make it more fun, we will use the Montreal Snow Festival ([source]) image – Figure:4(a), and you have to find a person, shown in Figure:4(b).

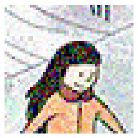
1. Given a reference image (scene of Montreal Snow Festival) and a template face image, iterate over all pixel locations of the reference image and compare the local patch with the template image using the sum of square distance (SSD) metric. Implement this in NumPy (you are prohibited from using OpenCV function for this). **(6 points)**
2. Display SSD for the whole image. Find the location ((x, y) coordinate) where the SSD is minimum and see if you can find the person there! **(4 points)**
3. Repeat the above process with the noisy template as shown in Figure:4(c). **(4 points)**



(a)



(b)



(c)

Figure 4: (a) Cluttered scene of Montreal Snow Festival [source] (b) template face (c) noisy template face.