Assignment 2

Due on 5/5/2016 23:55

Introduction

In the past few years, content based image retrieval (CBIR) was in the spotlight of academic research and industrial projects. The term CBIR used to describe an application in which images are retrieved based on their content, as opposed to old methods in which images were retrieved using labels and tags. In this assignment you will implement a simple CBIR program using global and local image features. The purpose of this assignment is to get familiar with the following:

1- The OpenCV library for image processing
2- The accuracy performance in image retrieval using global features vs. local features
3- Basic C++ concepts

OpenCV

OpenCV is an open source library which includes many algorithms, data structures and other features which are used for image processing. Before proceeding with the assignment instructions, make sure you have installed OpenCV and made your first C++ project using OpenCV. The installation instructions can be found on the moodle website.

Image Features

Image features (also called image descriptors) are pieces of information, which are relevant for solving many tasks related with certain application in computer vision. You saw in class two types of features, global and local features. Global features are information that describe the picture as a whole, that is, information gathered by looking on the picture globally. Local features on the other hand are obtained by focusing on local regions in the picture.

CBIR using OpenCV

In this section, you will implement a simple CBIR system for image retrieval. The system is command line based program. In **Part A** we will implement a utility functions that will be later used for the purpose of CBIR. In **Part B** we will implement the rest of the program.

Part A - Utility functions in Image Processing

All the functions you will implement below must be located in the source file **"sp_image_proc_util.cpp"**. You can find the declaration of the functions in the header file **"sp_image_proc_util.h**".

Note: We will ignore the run-time complexity of the functions in OpenCV. Thus we will explicitly state the number of operations per function in openCV (if not stated then we will be referring to all OpenCV operations/functions/methods).

```
spGetRGBHist(char* str, int nBins);
```

Write a function in C/C++ which receives a string (char* str) and an integer (int nBins). The function returns a histogram of colors (RGB Histogram).
The parameters of the function are the following:

1- char* str - A C string representing an image path relative to the current directory. For example:
"./images/img1.jpg"
This string represents the path of the image **img1.jpg** which is located in the directory **"images".**
2- Int nBins - The number of bins of the histogram

The function return type is **int**** (two-dimensional array). The function returns a dynamically allocated two dimensional array representing the RGB histogram. The dimensions of the array are $3 \times nBins$. The first row represents the Histogram of the **red** channel, the second row is the histogram of the **green** channel while the third row is the histogram of the **blue** channel.

More information on histograms can be found in the tutorial on moodle and the following link.

Complexity: $O(3 \times nBins) - operations.$
Note: You may use any OpenCV function/method.

```
spRGBHistL2Distance(int** histA, int** histB, int nBins);
```

Write a function in C/C++, which receives two RGB Histograms (int** histA, int** histB) and an integer (int nBins). Each histogram is $3 \times nBins$ matrix, in which the first row is the **red** channel histogram, the second row is the **green** channel histogram and the third row is the **blue** channel. The function returns the average of the L_2^2 (let's call it L2-squared) distance between the histograms of the corresponding channels.

Recall:

The L_2^2 distance (squared Euclidean distance) between two vectors is given by:

$$d^2(p,q) = (p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2.$$

For example:

$$INPUT:$$
$$nBins = 3$$
$$histA = \begin{bmatrix} r_{a_1} & r_{a_2} & r_{a_3} \\ g_{a_1} & g_{a_2} & g_{a_3} \\ b_{a_1} & b_{a_2} & b_{a_3} \end{bmatrix}, histB = \begin{bmatrix} r_{b_1} & r_{b_2} & r_{b_3} \\ g_{b_1} & g_{b_2} & g_{b_3} \\ b_{b_1} & b_{b_2} & b_{b_3} \end{bmatrix}$$
$$Output:$$
$$0.33\left(\sum_{i=1}^{nBins}\left(r_{a_i} - r_{b_i}\right)^2\right) + 0.33\left(\sum_{i=1}^{nBins}\left(g_{a_i} - g_{b_i}\right)^2\right) + 0.33\left(\sum_{i=1}^{nBins}\left(b_{a_i} - b_{b_i}\right)^2\right)$$

Complexity: $O(3 \times nBins)$

Note: You are expected to write this function by yourself. You **are not allowed to use** any OpenCV functions.

```
spGetSiftDescriptors(char* str, int maxNFeautres, int
*nFeatures);
```

The function returns a two dimensional array of dimensions $nFeatures \times 128$. Each row in the matrix represents a SIFT descriptor; use OpenCV `SiftDescriptorExtractor` class to achieve the requirements of the functions.

The inputs of the function are:

- (char* str) which is a string representing the image path
- (Int maxNFeatures) which is an integer representing the maximum number of features (descriptors) to be extracted.
- (int nFeatures) a pointer to which the total number of features that were actually extracted will be stored.

Complexity: $O(nFeatures \times 128) - operations.$
Note: You may use any OpenCV function/method.

```
spL2SquaredDistance(double* featureA, double* featureB);
```

Calculates the L2 square between the two SIFT features. The two arrays representing SIFT features. The L2-squared distance is defined as in previous section. Before defining the distance between two featurs, we will denote:

- $d_2^2(featuresA, featuresB)$ - The L2-squared distance between the featureA and featureB.

For example:

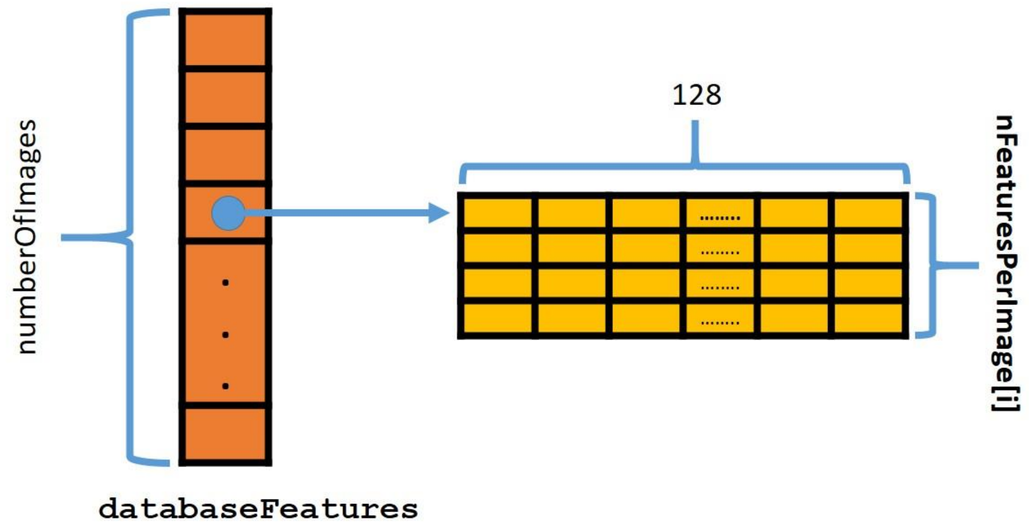
If $featureA = [a_1 \quad a_2 \quad \dots \quad a_{128}]$, $featureB = [b_1 \quad b_2 \quad \dots \quad b_{128}]$ then the function will return:

$$d_2^2(featureA, featureB) = (a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_{128} - b_{128})^2$$

Complexity: 128X128 = O(1)

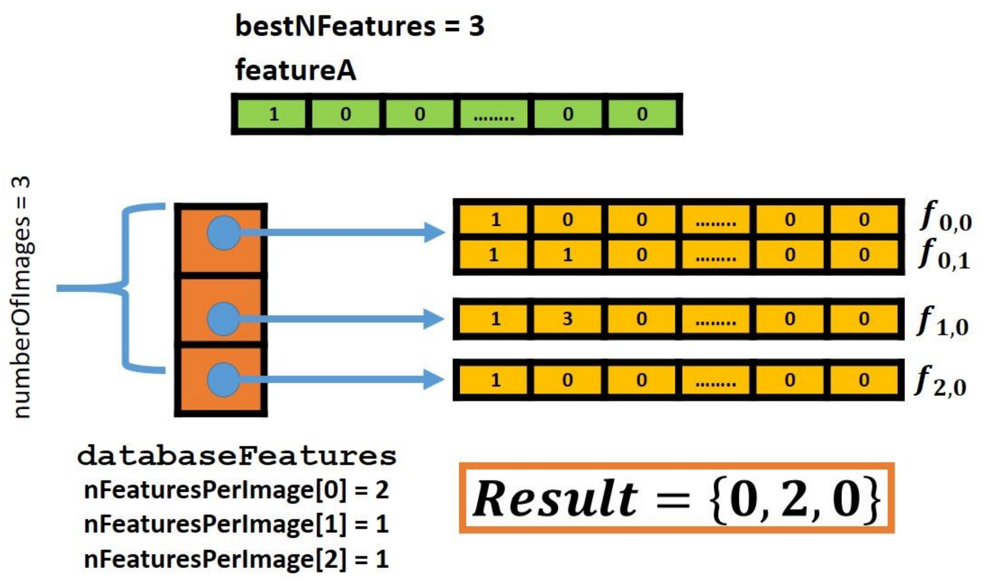
```
int* spBestSIFTL2SquaredDistance(int bestNFeatures, double*
featureA, double*** databaseFeatures, int numberOfImages,
int* nFeaturesPerImage);
```

The function receives an array `databaseFeatures`, each entry in the array corresponds to the features of some image in the database. The number of features of an image is given by the array `nFeaturesPerImage`. See below:



The function calculates the L2-Squared distance between `featureA` and all the other features in each one of the matrices given by `databaseFeatures`. The function then returns the indices of the images to which the best (closest) `bestNFeatures` belongs to. The result must be sorted in ascending order (According to the distances of the features). In

case there are two features that have the same distance with featureA, then the one with the smallest index will be the smaller one.
For example:
Given the parameters below the result is {0,2,0}, since
$d_2^2(featureA, f_{0,0}) = d_2^2(featureA, f_{2,0}) = 0$,
$d_2^2(featureA, f_{0,1}) = 1$ and $d_2^2(featureA, f_{1,0}) = 9$,
thus the closets 3 features are $f_{0,0}$,$f_{2,0}$ $f_{0,1}$. (Note that the $d_2^2(featureA, f_{0,0}) = d_2^2(featureA, f_{2,0}) = 0$ but the index of $f_{0,0}$ is 0 and $f_{2,0}$ is 2 hence the first index returned is 0 and the second index is 2). See the header file for more information.



Complexity: $O(bestNFeatures \times nFeatures \times numberOfImages \times 128) - operations.$

Part B - Main function

In this section, all helper functions (auxiliary functions) must be implemented in the file “main_aux.cpp” and all definition must be placed in the header file “main_aux.h”.

Write a main function (“main.cpp”) that receives a directory of images and a query image, and retrieves similar images to the query image.

The behavior of the main function (program) should be the following:

1. Ask the user to enter a string representing the directory path of the images (More information will be given shortly):
   `"Enter images directory path:\n"`
   The directory contains the images that we will be searching from. All image names must

be the same differencing only in their index. For examples, the directory: **"./images/"** may contain the following images: **"img0.jpg", "img1.jpg", "img2.jpg"…**

2. Ask the user to enter a string representing the prefix of the images in the images directory.
   `"Enter images prefix:\n"`
   The images name start with the prefix that the user enters. In the example given in Section 1 the images prefix is **"img".**

3. Ask the users to enter a string representing the total number of images in the directory given by Section1.
   `"Enter number of images:\n"`
   The total number of images in the directory will be the number represented by the input of the user. The images will be numbered starting from 0 to n-1. Thus if the user entered 4, then the number of images in the directory is 4 and the images are: **"img0.jpg", "img1.jpg", "img2.jpg" and "img3.jpg".**
   If the user entered an integer that is less than 1 then an error message will be printed:
   `"An error occurred - invalid number of images\n"`
   After the error is printed the program will terminate. (All allocations must be free'd.)

4. Ask the user to enter the suffix of the images in the images directory:
   `"Enter images suffix:\n"`
   The suffix will be the same for all images, in our example the prefix is **".jpg"**.

5. Ask the user to enter the number of bins which will be the total number of bins calculated for the RGB histogram:
   `"Enter number of bins:\n"`
   If the user entered a number that is less than 1, then an error will be printed:
   `"An error occurred - invalid number of bins\n"`
   After the error is printed the program will terminate. (All allocations must be free'd).

6. Ask the user to enter the maximum number of SIFT features which will be extracted for each image in the database:
   `"Enter number of features:\n"`
   If the user entered a number that is less than 1, then an error will be printed:
   `"An error occurred - invalid number of features\n"`
   After the error is printed the program will terminate. (All allocations must be free'd).

7. Now the program will calculate the RGB histogram for each image in the images directory, and the SIFT descriptors (You may store the results in a multi-dimensional arrays). Use the functions you did in PART - A, the parameters maxNFeatures, nBins and numberOfImages were given by the user in previous sections.

8. After the preprocessing is done (section 7) the program is expecting a query image path or a termination command. The termination command is the special character **"#".** The program will print the following line:
```
"Enter a query image or # to terminate:\n"
```

9. If a termination command is entered, i.e. **"#"** then the program will terminate. Prior to exiting the program will print the following message:
```
"Exiting...\n"
```
After the exiting message is printed the program will terminate. (All allocations must be free'd).

10. If the termination command wasn't entered, the program will do the following:
    - It will calculate the RGB histogram and Sift descriptors of the query image.
    - **Search using Global Features:**
      For each RGB histogram of the images in the database, the program computes the L2-Squared distances between the query image and the histograms. It will show the best (closest) 5 image indexes based on the L2-squared distance with the histograms (i.e. the best 5 images for which the closest histograms belongs to).
      The program will present the best 5 scores (lowest 5 distances based on the global features) with respect to the RGB Histograms. It will present the results by printing the following message:
      ```
      "Nearest images using global descriptors:\n"
      "i1, i2, i3, i4, i5\n"
      ```
    - **Search using Local Features**:
      FOR EACH SIFT feature of the query image, the program will search the 5 closest features in the database using the function **spBestSIFTL2SquaredDistance**. The function as stated in part A returns the indexes of the images to which those features belong. We will track the number of hits per image in the database and present the best 5 images (Those with the highest number of hits). For example, if the query image has 3 features, and the function in Part A returned {3,4,1,2,3} ,{0,3,3,3,4} and {3,1,1,2,0} for those features. Then image0 has 2 hits, image 1 has 3 hits, image 2 has 2 hits, image 3 has 6 hits, image 4 has 2 hits and the rest has 0 hits. Then the best 5 images are {image 3, image 1, image 0, image 2, image 4}.
      The program will present the best 5 scores (highest 5 images) with respect to the number of hits in descending order (The image with the highest hit first, the image with second highest second etc..). It will present the results by printing the following message:
      ```
      "Nearest images using local descriptors:\n"
      "j1, j2, j3, j4, j5\n"
      ```
    - **After the results are shown, all descriptors that belongs to the query image must be free'd**

NOTE: In case two images have same score with respect to the query image, use the indices as a tie-breaker (the lower index image should be printed first).

For example:
Let us assume that the query image is “query.jpg” (that is the user entered the line “query.jpg”) and that:

- Images: “img0.jpg”, “img6.jpg”, “img3.jpg”, “img7.jpg” and “img2.jpg” were the nearest images to “query.jpg” with respect to their RGB histograms. (The images appears in the order of their distances - in ascending order)
- Images: “img1.jpg”, “img8.jpg”, “img5.jpg”, “img7.jpg” and “img3.jpg” were the nearest images to “query.jpg” with respect to their SIFT Descriptors. (The images appears in the order of their distances)

Then the program will print:

```
Nearest images using global descriptors:
0, 6, 3, 7, 2
Nearest images using local descriptors:
1, 8, 5, 7, 3
```

11. Repeat section **(8-10)**

Notes:

1- In case of memory allocation failure at any stage in the program, print:
`"An error occurred - allocation failure\n"`
After the error is printed the program will terminate. (All allocations must be free’d).
2- Memory leaks (dynamically allocated memory) will cause points reductions.
3- For sorting purposes, you **MUST** use qsort (or implement one yourself) any other naïve sorting methods will not be accepted and points reduction will be given.
4- The source file main.cpp will contain only the main function. Other function must be defined in main_aux.h.
For example, if for some reason you implemented a max function that returns the maximum between to numbers. The max function must be defined and implemented in main_aux.h/main_aux.cpp.
5- Constant strings/Magic numbers must be defined using macros as taught in class. Don’t write 3.14 in your code to represent the number pi, use:

```
#define PI 3.14
#define EXIT_MSG "Exiting...\n"
```

Assumptions:

1- You may assume the directory and images given by the user exist.
2- You may assume the user enters valid integers, Thus “-1” and “10” are possible while “a12” isn’t.
3- You may assume each line is of maximum length 1024

4- You may assume any image path length will not exceed 1024 characters.
5- You may assume the number of images is at least 5.

Hints:

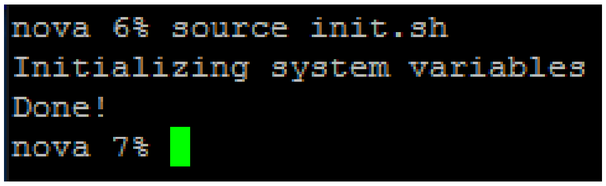
1- In the first assignment you saw that the standard input (stdin) didn't change when the user entered the input (Thus the output only appeared last after the termination of the program).
Add the following line after each I/O function:
```
fflush(NULL);
```
For example, if you want to print hello word:
```
printf("Hello world\n");
fflush(NULL);
```
if You want to receive a character use:
```
getchar();
fflush(NULL);
```
This is a bug as a result of eclipse on windows. And doing so will make your life a lot easier when working on windows.

2- Use `atoi(number);` to convert a string to an integer.

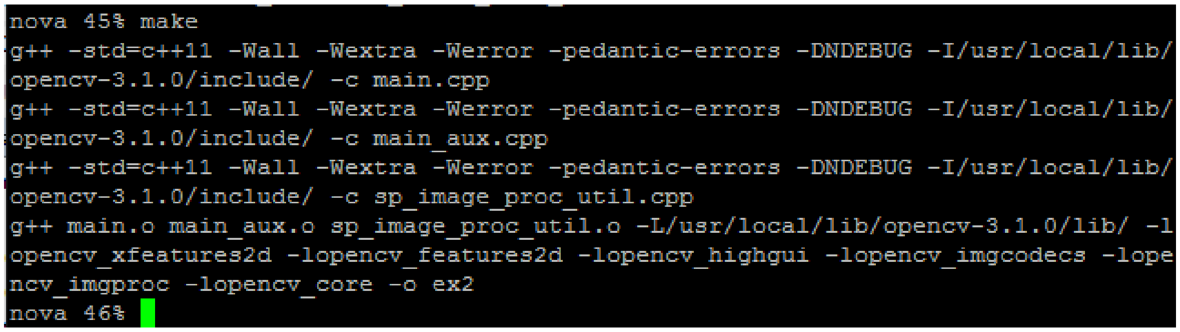
3- You may use any library functions, we suggest the following functions:
```
sprintf (char*, const char*, ...);
fgets (char*, int, FILE*);
strcspn (const char*, const char*);
qsort(void*, size_t, size_t, int (*)(const void*, const void*));
```

Testing

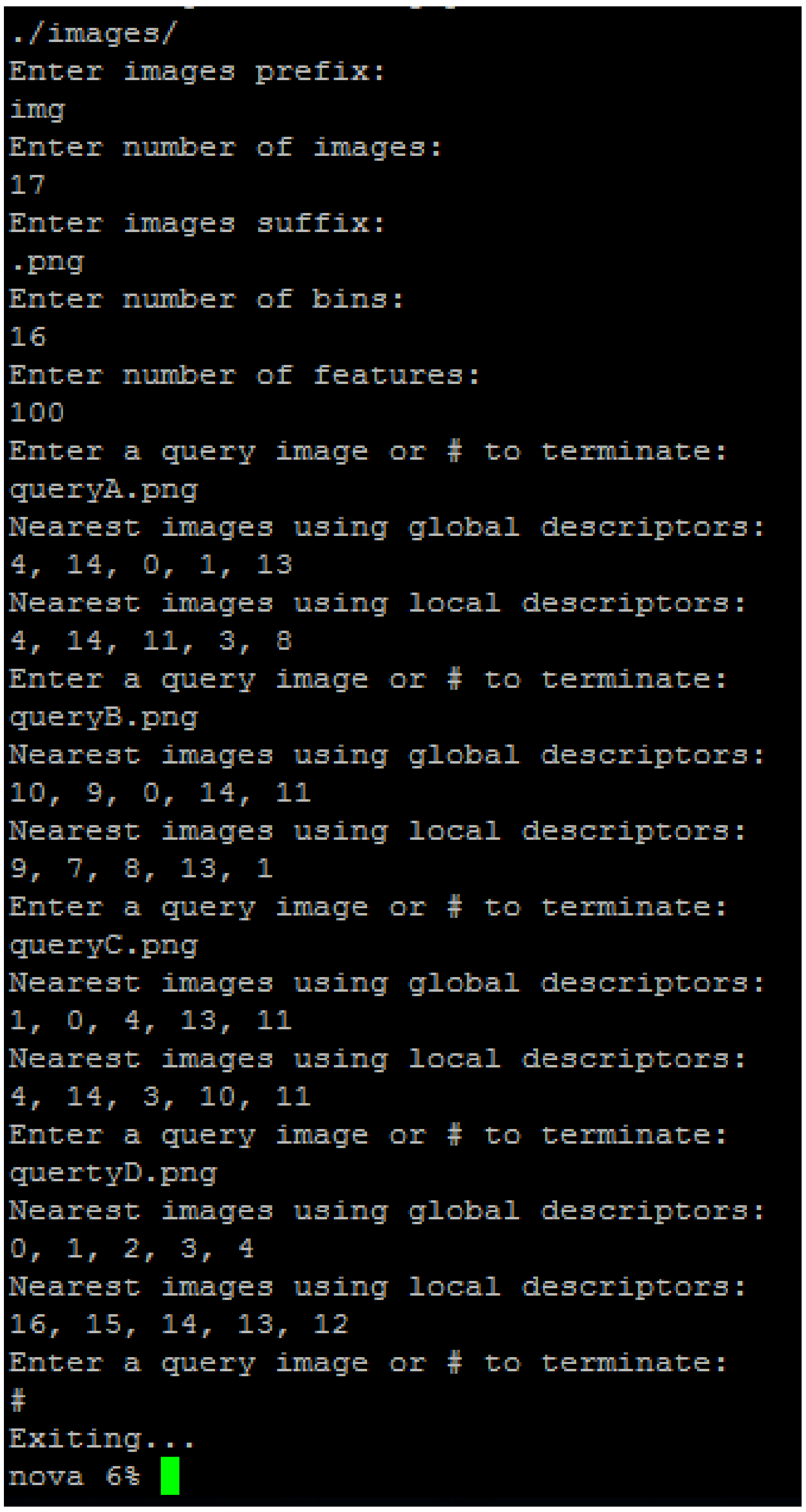
When you log-in to nova, please run the script init.sh (Given in the assignment file), you must run the script every time you connect to nova (Invoke **>> source init.sh**):


```
nova 6% source init.sh
Initializing system variables
Done!
nova 7%
```

Use the make file in the assignments directory in order to build your project on nova.


```
nova 45% make
g++ -std=c++11 -Wall -Wextra -Werror -pedantic-errors -DNDEBUG -I/usr/local/lib/
opencv-3.1.0/include/ -c main.cpp
g++ -std=c++11 -Wall -Wextra -Werror -pedantic-errors -DNDEBUG -I/usr/local/lib/
opencv-3.1.0/include/ -c main_aux.cpp
g++ -std=c++11 -Wall -Wextra -Werror -pedantic-errors -DNDEBUG -I/usr/local/lib/
opencv-3.1.0/include/ -c sp_image_proc_util.cpp
g++ main.o main_aux.o sp_image_proc_util.o -L/usr/local/lib/opencv-3.1.0/lib/ -l
opencv_xfeatures2d -lopencv_features2d -lopencv_highgui -lopencv_imgcodecs -lope
ncv_imgproc -lopencv_core -o ex2
nova 46%
```

Now you need to simply run the program **ex2** that was created after compilation is over.



```
./images/
Enter images prefix:
img
Enter number of images:
17
Enter images suffix:
.png
Enter number of bins:
16
Enter number of features:
100
Enter a query image or # to terminate:
queryA.png
Nearest images using global descriptors:
4, 14, 0, 1, 13
Nearest images using local descriptors:
4, 14, 11, 3, 8
Enter a query image or # to terminate:
queryB.png
Nearest images using global descriptors:
10, 9, 0, 14, 11
Nearest images using local descriptors:
9, 7, 8, 13, 1
Enter a query image or # to terminate:
queryC.png
Nearest images using global descriptors:
1, 0, 4, 13, 11
Nearest images using local descriptors:
4, 14, 3, 10, 11
Enter a query image or # to terminate:
quertyD.png
Nearest images using global descriptors:
0, 1, 2, 3, 4
Nearest images using local descriptors:
16, 15, 14, 13, 12
Enter a query image or # to terminate:
#
Exiting...
nova 6%
```

Submission

Please submit a zip file named **id1_id2_assignment2.zip** where id1 and id2 are the ids of the partners. The zipped file must contain the following files:

- main.cpp – Your source code for the program.
- main_aux.h – All auxiliary functions you used in the main function (Other than those implemented in Part A) will be defined in this file.
- main_aux.cpp – Your implementation for the header main_aux.h
- sp_image_proc_util.h - The header given to you in the assignments files **(Don't change it)**
- sp_image_proc_util.cpp -Your implementations of the functions defined in sp_image_proc_util.h
- partners.txt – This file must contain the full name, id and moodle username for both partners. Please follow the pattern in the assignment files. **(do not change the pattern)**
- makefile – the makefile provided in the assignment files. **(Don't change it!)**

Note:

- **When submitting your code, make sure you press "submit for grading". Your submission will not be automatically submitted when on "draft" state.**

Remarks

- For any question regarding the assignment, please don't hesitate to contact Moab Arar by mail: moabarar@mail.tau.ac.il.
- Borrowing from others' work is not acceptable and may bear severe consequences.

Good Luck