Assignment 3

Due on 29/1/2017 23:55

Introduction

In the past few years, content based image retrieval (CBIR) was in the 6spotlight 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

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 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(const char* str, int imageIndex, int nBins);
```

Write a function in C++ which receives a string (const char* str) and an integer (int nBins). The function returns a histogram of colors (RGB Histogram) of some image with the given index. The function returns an array of points of size 3. The first entry is the R-channel histogram, the second entry is the G-channel histogram and the third entry is the B-channel histogram.
The parameters of the function are the following:

1- const char* str - A string representing an image path.
 For example:
 "./images/img1.jpg"
 This string represents the path of the image **img1.jpg** which is located in the directory **"images".**
2- Int imageIndex - The index of the given image, in the above example it is '1'.
3- Int nBins - The number of bins of the histogram

The function return type is **SPPont**** (array of SPPoint*). The function returns 3 points representing the RGB histogram. The cardinality of each point is $nBins$, and it has the index given by imageIndex. The first point represents the histogram of the **red** channel, the second point is the histogram of the **green** channel while the third row is the histogram of the **blue** channel. For more information refer to the header file sp_image_proc_util.h.

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

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

```
spRGBHistL2Distance(SPPoint** histA, SPPoint ** histB);
```

Write a function in C++, which receives two RGB Histograms (SPPoint** histA, SPPint** histB). Each histogram is an array of size 3, 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:

$$INPUT:$$

$$histA = \begin{bmatrix} r_{a_1} & r_{a_2} & r_{a_3} \\ g_{a_1} & g_{a_2} & g_{a_3} \\ b_{a_1} & b_{a_2} & b_{a_3} \end{bmatrix}, histB = \begin{bmatrix} r_{b_1} & r_{b_2} & r_{b_3} \\ g_{b_1} & g_{b_2} & g_{b_3} \\ b_{b_1} & b_{b_2} & b_{b_3} \end{bmatrix}$$

$$Output:$$

$$0.33\left(\sum_{i=1}^{nBins}\left(r_{a_i} - r_{b_i}\right)^2\right) + 0.33\left(\sum_{i=1}^{nBins}\left(g_{a_i} - g_{b_i}\right)^2\right) + 0.33\left(\sum_{i=1}^{nBins}\left(b_{a_i} - b_{b_i}\right)^2\right)$$

Complexity: $O(3 \times nBins)$

Notes:

- You are expected to write this function by yourself. You **are not allowed to use** any OpenCV functions.
- When averaging over double, use the constant **0.33** and don't use fractions (i.e 1/3). This is safer since working with integers may result in wrong answers.

```
spGetSiftDescriptors(char* str, int imageIndex, int
nFeaturesToExtract, int *nFeatures);
```

The function returns an array of points each has an index given by imageIndex. Each entry in the array represents a SIFT descriptor; use OpenCV `SiftDescriptorExtractor` class to achieve the requirements of the functions.

The inputs of the function are:

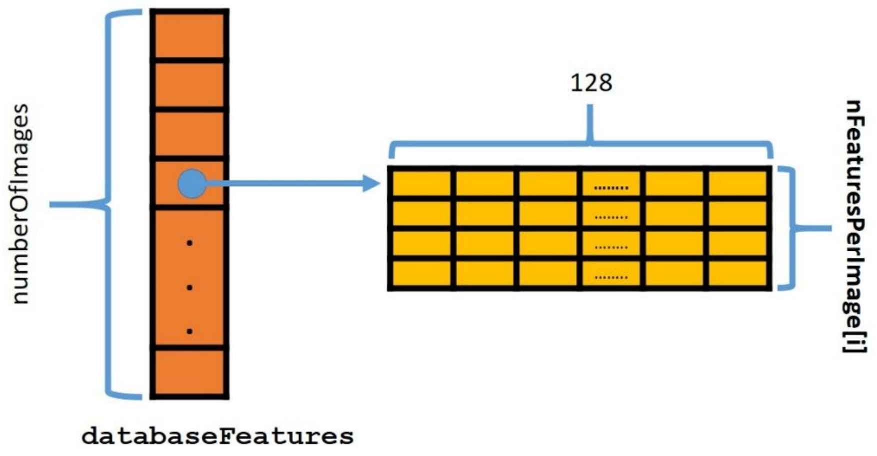
- (const char* str) a string representing the image path
- (int imageIndex) an integer representing the image index
- (Int nFeaturesToExtract) which is an integer representing the number of features (descriptors) to be extracted. Notice that OpenCV may extract less/more than this parameter.
- (int nFeatures) a pointer to which the total number of features that were actually extracted will be stored.

The function will return an array of size k, each entry in the array is of type SPPoint* where such point represents a SIFT descriptor (dimension 128). Each point in the array is given the index imageIndex. The size of the array will be stored at the address given by nFeatures.

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

```
int* spBestSIFTL2SquaredDistance(int kCloses, SPPoint*
queryFeature, SPPoint*** databaseFeatures, int
numberOfImages,
int* nFeaturesPerImage);
```

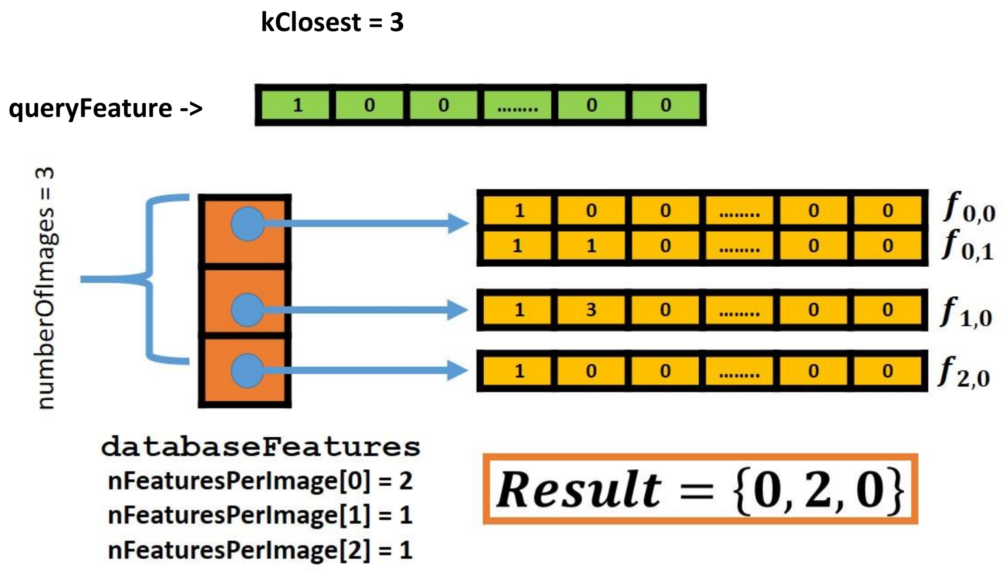
The function receives an array `databaseFeatures`, each entry in the array corresponds to the features of some image in the database. The number of features of an image is given by the array `nFeaturesPerImage.` That is the array databaseFeatures has numOfImages entries, each entry $databaseFeatures[i]$ is also an array of size $nFeaturesPerImage[i]$. The array $databaseFeatures[i]$ stores SIFT features that correspond to the image with the index i (notice that databaseFeatures is a 2 dimensional array, thus databaseFeatures[i][j] refers to the j^{th} feature of the i^{th} image) . See below figure for further clarification:



The function calculates the L2-Squared distance between `queryFeature` and all the other features stored in `databaseFeatures`. The function then returns the indices of the images to which the best (closest) `kClosest` 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 queryFeature, then the feature with the smallest index will be the closest to queryFeature (i.e use the indexes as tie-breaker).

For example:

Let us denote $f_{i,j}$ to be the j^{th} SIFT feature of the i^{th} image, thus in this case the index of the feature $f_{i,j}$ is i. Given the parameters below:



Then we expect the function to return the following array {0,2,0}, since we have that:

$$d^2(queryFeature, f_{0,0}) = d^2(queryFeature, f_{2,0}) = 0$$

$$d^2(queryFeature, f_{0,1}) = 1$$

And

$$d_2^2(queryFeature, f_{1,0}) = 9$$

Thus the closets 3 features are $f_{0,0}$,$f_{2,0}$ $f_{0,1}$ (sorted by their distance from queryFeature in ascending order).

Note that $d^2(queryFeature, f_{0,0}) = d^2(featureA, f_{2,0}) = 0$, but the index of $f_{0,0}$ is 0 while the index of $f_{2,0}$ is 2, hence we have that $f_{0,0}$ is closer to queryFeature that $f_{2,0}$.

See the header file for more information.

Complexity: $O(kClosest \times totalNumFeaturesInDatabase \times SIFT\_DIM) - operations.$

Special Notes:

- If a certain image cannot be loaded, then you need to print the following message: `"Image cannot be loaded - 'str':\n"`
  Where str is a string representing the image name. For instance if the image "img.jpg" wasn't loaded, then the following message is printed:
  "Image cannot be loaded - img.jpg\n".
  After the message is printed, you need to terminate the program properly (freeing all memory resources that were allocated).
- Although we have stated that SIFT descriptor is of dimensionality 128, please use the attribute (cols) in MAT in order to obtain this constant as we did in class. Thus you are expected to avoid the usage of the constant 128 for the SIFT descriptor.
- It is recommended that you use the priority queue that you implemented in assignment 2. Especially when you implement the function **`spBestSIFTL2SquaredDistance`**
- Don't forget when working with SIFT Descriptors, you need to load your image in gray scale mode (use the CV_LOAD_IMAGE_GRAYSCALE flag)

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.c") that receives a directory of images 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 path of the images directory (more information will be given shortly):
   `"Enter images directory path:\n"`
   The directory contains the images that we will be searching from. All images names must be the same except for their indexes. 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 or greater than 255, 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 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 and the SIFT descriptors for each image in the images directory. Use the functions you did in PART A, the parameters numOfBins, numberOfImages, nFeaturesToExtract 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, then it expects a string that represents the query image relative path. 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 print the best 5 scores (lowest 5 distances based on the global features) with respect to the RGB Histograms. It will print the results by printing the following message:
      ```
      "Nearest images using global descriptors:\n"
      "i1, i2, i3, i4, i5\n"
      ```
      Where i1, i2,..,i5 are the indexes of the images which are closest to the query image based on the RGB histogram.

    - **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 print the best 5 scores (**highest** 5 images) with respect to the number of hits in descending order (The image with the highest hits 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"
```

Where j1, j2,..,j5 are the indexes of the images which are closest to the query image based on their SIFT features.

- **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 **may** use qsort.
4- Avoid implementing long functions - usually in C any function that exceeds 60 lines of code (not including comments) is considered long. If you exceed this limit, then your function must be divided into several smaller functions.

5- Avoid code reusing - If you repeat some operation several times then it must be in a separate function
6- The source file main.cpp must only contain the main function, other functions must be defined in main_aux.h.
For example, if for some reason you implemented a max function that returns the maximum between two numbers, then the max function must be defined and implemented in main_aux.h and main_aux.cpp respectively.
7- 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
```
And don't use constant strings in your code (when printing for example), replace these strings with the macro:
```
#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- Some of you may have seen that the standard input (stdin) didn't change when the user entered the input (i.e the output only appeared after the termination of the program). To resolve this issue, you may 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*));
```

Search online for more information (these functions are very recommended for the purpose of this assignment)

Mixing C with C++

Although this is a C project, but in order to use the C++ source code sp_image_proc_util.cpp, your main function should be in a c++ source file (i.e main.cpp). Thus your main function should only be in the source file main.cpp doing otherwise will cause an unexpected outcome. Carefully review and follow the guideline:

C++ programming language is far more advanced than C and it is relatively easy to work with. **FOR THIS REASON YOU SHOULD NOT (FOR ANY REASON) USE ANY OF THE C++ LIBRARIES. ANY SOLUTION MIXING C and C++ WILL NOT BE ACCEPTED.**

Important notes when you come to implement you main.cpp file:

- If you want to use C libraries in main.cpp just include the library name without the .h extension and with c at the beginning. For example if you want to use <stdlib.h> then you need to #include <cstdlib> (see example below).
- If you want to use C source files that you implemented, then you need to use the special macro **extern** (see example below).
- If you want to use sp_image_proc_util, then your don't need to put your include inside the **extern** block (see example below).

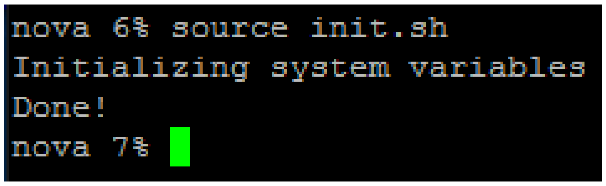
```
#include <cstdlib> //include c library
extern "C"{
//include your own C source files
#include "SPPoint.h"
}
//include C++ source files
#include "sp_image_proc_util.h"

int main(){
     //Your implementation of the main function
     return 0;
}
```

- The source file main.cpp must contain only one function (i.e the main function). Don't define any auxiliary functions in main.cpp. If you want to implement helper functions then use main_aux **C++ source file** and include it in your main function as has been described in previous section. Note that in
- The only C++ source files your code must have are **main.cpp**, **main_aux.cpp** and **sp_image_proc_util.cpp**. If you use any other C++ files **your assignment will not be accepted!**

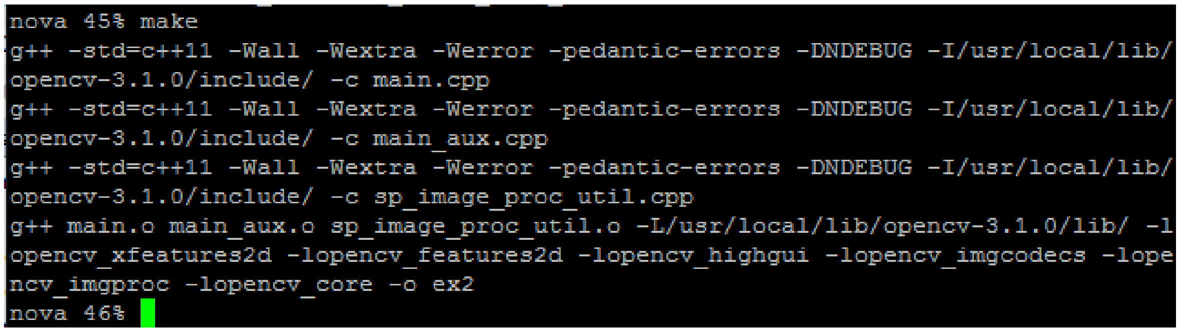
Testing

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**):



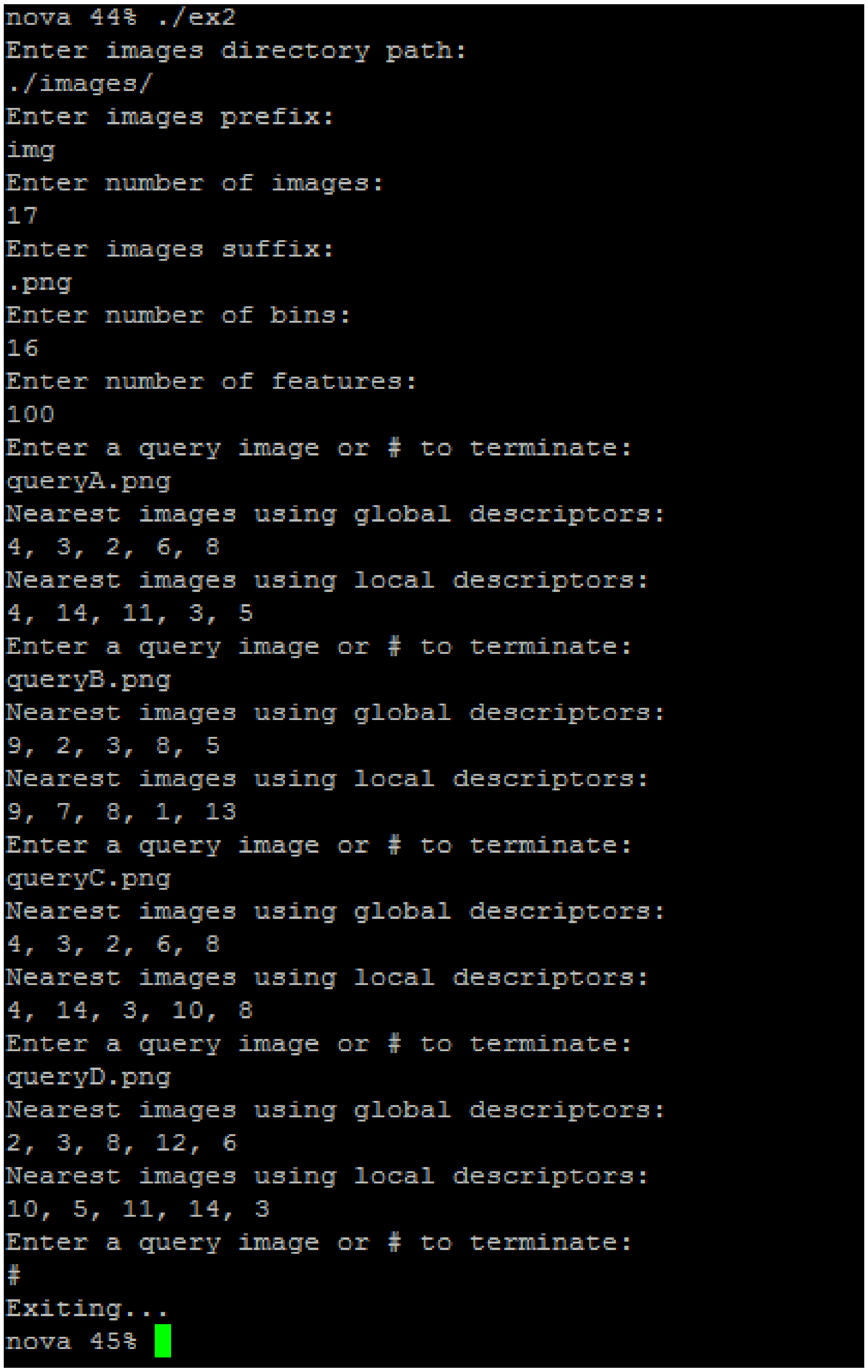
```
nova 6% source init.sh
Initializing system variables
Done!
nova 7%
```

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



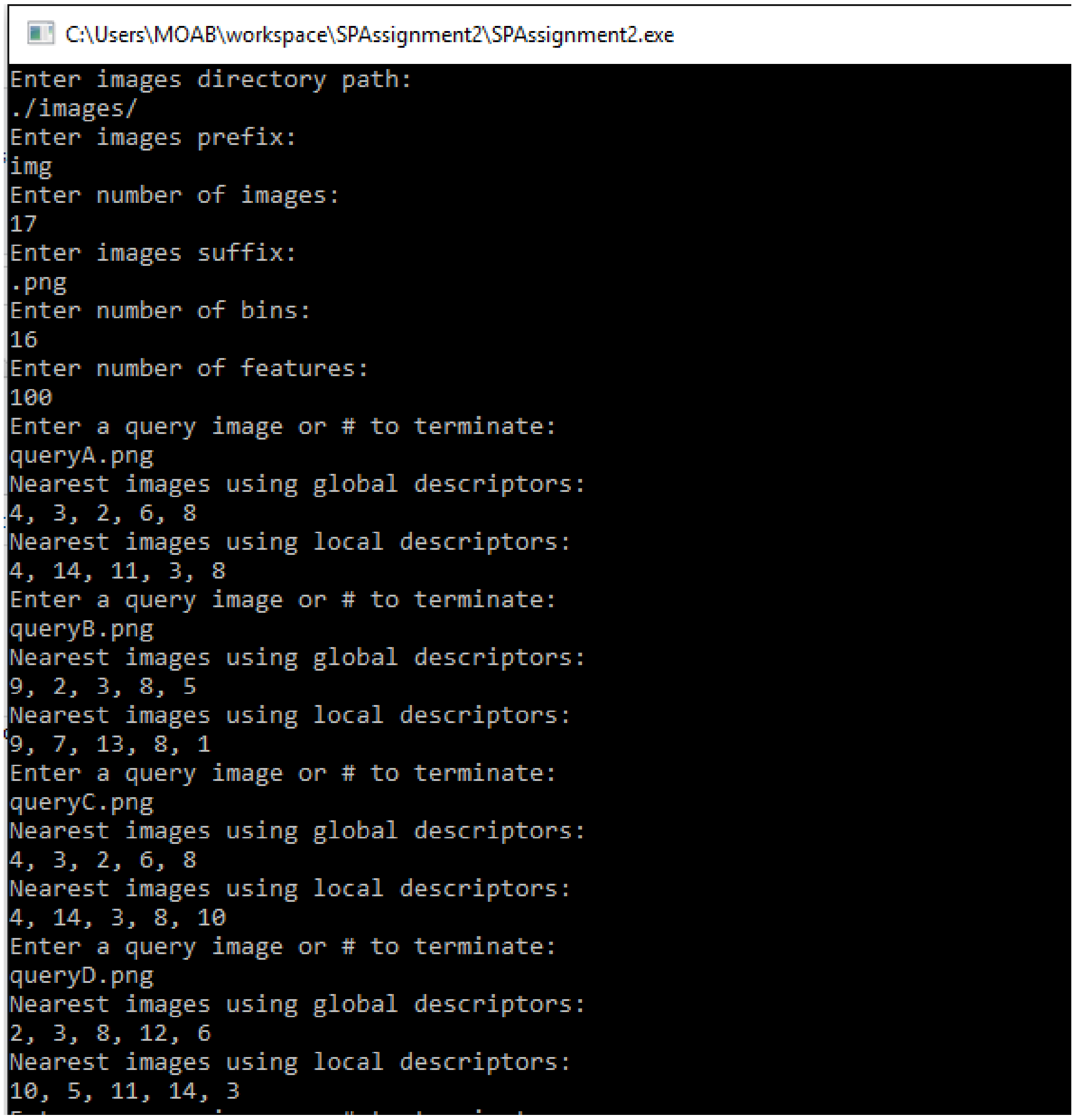
```
nova 45% make
g++ -std=c++11 -Wall -Wextra -Werror -pedantic-errors -DNDEBUG -I/usr/local/lib/
opencv-3.1.0/include/ -c main.cpp
g++ -std=c++11 -Wall -Wextra -Werror -pedantic-errors -DNDEBUG -I/usr/local/lib/
opencv-3.1.0/include/ -c main_aux.cpp
g++ -std=c++11 -Wall -Wextra -Werror -pedantic-errors -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 simply need to run the program **ex3** that was created after compilation is over. An example of a run is given in the figure below (next page). Notice the printing format and the expected results.



```
nova 44% ./ex2
Enter images directory path:
./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, 3, 2, 6, 8
Nearest images using local descriptors:
4, 14, 11, 3, 5
Enter a query image or # to terminate:
queryB.png
Nearest images using global descriptors:
9, 2, 3, 8, 5
Nearest images using local descriptors:
9, 7, 8, 1, 13
Enter a query image or # to terminate:
queryC.png
Nearest images using global descriptors:
4, 3, 2, 6, 8
Nearest images using local descriptors:
4, 14, 3, 10, 8
Enter a query image or # to terminate:
queryD.png
Nearest images using global descriptors:
2, 3, 8, 12, 6
Nearest images using local descriptors:
10, 5, 11, 14, 3
Enter a query image or # to terminate:
#
Exiting...
nova 45%
```

NOTE - The above result is for nova only, if you are working on windows with MinGW32 you should get the following result:



C:\Users\MOAB\workspace\SPAssignment2\SPAssignment2.exe

```
Enter images directory path:
./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, 3, 2, 6, 8
Nearest images using local descriptors:
4, 14, 11, 3, 8
Enter a query image or # to terminate:
queryB.png
Nearest images using global descriptors:
9, 2, 3, 8, 5
Nearest images using local descriptors:
9, 7, 13, 8, 1
Enter a query image or # to terminate:
queryC.png
Nearest images using global descriptors:
4, 3, 2, 6, 8
Nearest images using local descriptors:
4, 14, 3, 8, 10
Enter a query image or # to terminate:
queryD.png
Nearest images using global descriptors:
2, 3, 8, 12, 6
Nearest images using local descriptors:
10, 5, 11, 14, 3
```

Submission

Please submit a zip file named **id1_id2_assignment3.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
- SPPoint.h SPPoint.c SPBPriorityQueue.h and SPBPriorityQueue.c - You need to include your implementation of these modules as well (**from assignment 2**)
- 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!)**

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