**Project Report**

**Code Architecture**

The primary code files for the OMR project are a2.cpp and Sift.h. The structure of files are as follows:

**Sift.h**

This is a header file which contains a list of commonly used functions that are called throughout the program. The functions listed in it are

* compute\_sift () : It takes a greyscale image (CImg)as input, and returns a vector of SiftDescriptors. It uses function from the files sift.hpp and CImg.h for this purpose.

**a2.cpp**

This is the main code file . It is used to perform functions like Sift matching, image warping, homography estimation etc.

**comparingvalues (CImg<double> input\_image3,CImg<double> input\_image1,vector<SiftDescriptor> &descriptorsip, vector<SiftDescriptor> &descriptors, int taskno)** : This function is used to calculate the best features based on distance matching. It takes two images , their sift descriptors and a taskno. parameter and returns the no. of matching features as an output. Additionally if the taskno. is '1' then it also saves the matched features as a single image.

**drawSingleImage(vector<SiftDescriptor> &descriptors1,vector<SiftDescriptor> &descriptors2,CImg<double> input\_image3,CImg<double> input\_image1,CImg<double> input\_image2)** : This function calls comparingvalues function with the appropriate parameters.

**sortImages(vector<string> fileNames,CImg<double> queryimg,vector<SiftDescriptor> &descriptorsip)** : This function sorts the images in descending order of matching features. It take a vector of file names, the input image and its sift descriptor as input. It maks use of comparingvalues function.

**retrievalAlgorithm (vector<string> fileNames,string queryimg,vector<SiftDescriptor> &descriptorsip)** : This function finds the precision value for all images that are correctly matched from the same attraction. It take a vector of file names, the input image name and its sift descriptor as input.

**inverse\_warp ( CImg<double> &input)** :This function takes in an image and a transformation matrix as an input and gives the transformed image as an output. It uses the inverse warping method to eliminate holes in the output image. It is also based on the nearest neighbour interpolation.

**featureDistance(SiftDescriptor idescriptor, SiftDescriptor cdescriptor)** : It takes in two sift descriptors as input and gives the eucledian distance between them as the output.

**getMatchingFeature (vector <double> dRow, int features)**: It finds the closest neighbour to a given feature. It takes sift descriptor and its distance from the input sift descriptor as an input and return the closet neighbour only if it is closer than the second closest neighbour by some threshold value.

**allFeatureDistance(vector<SiftDescriptor> idescriptors, vector<SiftDescriptor> cdescriptors)** : this function takes a sift descriptors of input and comparing image as an input and return the list of matching features.

**findTransformationMatrix(vector<pair<int,int> > matches, vector<SiftDescriptor> descriptors, vector<SiftDescriptor> cmpdescriptors)** : It takes in the input and output sift descriptors and the set of matching features and finds the transformation matrix.

**getVote (CImg <double> transformationmatrix, vector<SiftDescriptor> descriptors,vector<SiftDescriptor> cmpdescriptors, matches, vector<SiftDescriptor>)** : It get a vote on a transformation matrix form all the matching descriptors to find out the number of inliners and outliners.

**ransac (vector<pair<int,int> > matches, vector<SiftDescriptor> descriptors, vector<SiftDescriptor> cmpdescriptors)**: It gives the transformation matrix .

STEPS TAKEN IN PART 1:

At first, we establish a matching routine that generates the SIFT descriptors for two images using the SIFT operator from the Cimg library. Then, it matches any two (almost) similar features based on the Euclidian distance.

(RESULT NEEDED FOR QUESTION 1)

It also generates one single image, superposition of the two input images, with lines connecting the matched features. A feature is matched only if the smallest Euclidian distance divided by the second smallest one is inferior to the threshold value 0.8.

We use this routine to query a specific image in a library. Based on the number of matched features as an indicator of how similar are two images, a list of sorted images ( in the decreasing order ) is generated.

( RESULT NEEDED FOR QUESTION 2 )

In order to assess the precision of the retrieval image algorithm described above, we tested it on a database of 100 images : each ten images corresponds to a touristic attraction. For each attraction, an image is chosen from the ten images. We run the retrieval algorithm on the 100 images. Then, a precision number is calculated as the percentage of images from the correct attraction in the ten first images. For a scenario of 100% precision, the ten first sorted images should correspond to the actual ten images of the attraction.

(RESULT NEEDED FOR QUESTION 3)

Another approach is used to retrieve images with a shorter running time. First, a projection of the descriptor in the first image with a small ( k<< 128 ) set of vectors ( summarization ) is calculated. Then, the same projection is calculated for all descriptors in the second image. A small subset of descriptors is pre-selected based on how close their projection is to the projection of the descriptor in the first image. Then, the feature matching is applied to this small subset.

STEPS TAKEN IN PART 2 :

First, an inverse warping function is implemented. Inverse warping has two main advantages over forward warping :

* The output image has no cracks or holes.
* All resulting pixels are within the domain of definition because of how the algorithm is constructed.

Because the corresponding pixels in the original image can be non-integer, this function uses the nearest neighbour interpolation.

The inverse warping function is tested with a picture under the filename Lincoln.png and a 3\*3 transformation matrix.

References :

Szeliski, Computer Vision: Algorithms and Applications. Springer, 2011