CS484

Spring 2019

Section 1

Homework 2 Report & Specific Entry Points

Name: Ünsal Öztürk

ID: 21601565

The following is the documentation of the steps and the code necessary to read the data set, compute features, match features, compute affine transforms between the images, and finally image stitching and blending. The report also contains example instructions regarding how the script may be executed with a particular set of images, discussions about implementation choices, parameter tuning, and method drawbacks.

Sections and Output in Order of Appearance in cs484_hw2.pdf:

Section 3.1 Preparing Data

How to read in images using my script

I used the following MATLAB script to read the images in sequential order.

```
function images = read_images(dir_name)
%Use pattern matching to neatly read images from a directory.
imagefiles = dir(strcat(dir_name,'*.png'));
nfiles = length(imagefiles);
for ii=1:nfiles
   currentfilename = imagefiles(ii).name;
   currentimage = single(imread(currentfilename));
   images{ii} = currentimage;
end
```

This is, in my opinion, a very elegant solution to image I/O. It takes a string parameter, dir_name, which is the name of the directory plus the generic name of the images that will be read, similar to the format in which the data set is presented on the course website. Assume that the “goldengate” dataset is situated in the same directory as main.m, under a folder called “goldengate”. Also assume that the images follow the naming convention “goldengate-xx.png”. This script simply reads the directory and checks whether there are matching names with the parameter dir_name as its prefix. An example invocation is as follows:

```
images = read_images('goldengate\goldengate');
```

The return value is a cell array containing all the images with the pattern “goldengate-xx.png” under the directory “goldengate”, which should be located in the root directory of the code.

Section 3.2 Detecting Local Features

Getting Started

For this step, I made use of the VLFeat library, which was recommended in the homework assignment. In order to get the code running, the VLFeat library should be placed as follows in the root directory:

```
src\vlfeat
```

which should contain the binaries required for the MATLAB installation of VLFeat. In particular, the script runs the following command on start-up, which ensures that VLFeat is installed.

```
run('src\vlfeat\toolbox\vl_setup.m');
```

Essentially, the script won't work without the VLFeat binaries specifically placed inside the root directory in the described way.

After installing VLFeat, one can call the script I wrote using the "panaroma(3)" function, which takes in three parameters. The first parameter is the name of the directory and the pattern of the image files that are to be stitched. The second parameter is a string, which should only take the values

```
'sift'
```

```
'raw'
```

This parameter determines the descriptor that will be used to compute affine transformations from one image to another. The first parameter makes use of the SIFT descriptor while the second parameter uses a chi-squared-distance based raw grayscale pixel histogram descriptor (use with caution, causes memory issues sometimes!).

The third parameter is the threshold for the descriptor match scores. Any feature match higher than the specified threshold is discarded. This prevents noisy features to be influential in the panorama construction process.

Here is an example invocation of the function:

```
stitched = panaroma('goldengate\goldengate','sift',500);
```

This essentially constructs a panaroma from the images with the pattern "goldengate/goldengate-xx.png" using the SIFT descriptor and discards features that have a SSD higher than 500.

<u>Computing Feature Frames</u>

Feature frames and SIFT descriptors are computed easily via a single invocation of the VLFeat library. The following is the snippet from the script which computes the frames for every image in the dataset, along with their SIFT descriptors.

```
function [outf, outd] = compute_sift(images)
for i= 1:length(images)
    [f,d] = vl_sift(images{i});
    outf{i} = f;
    outd{i} = d;
end
```

The following is an example feature set on the first image in the goldengate dataset.

[This space has been intentionally left blank]

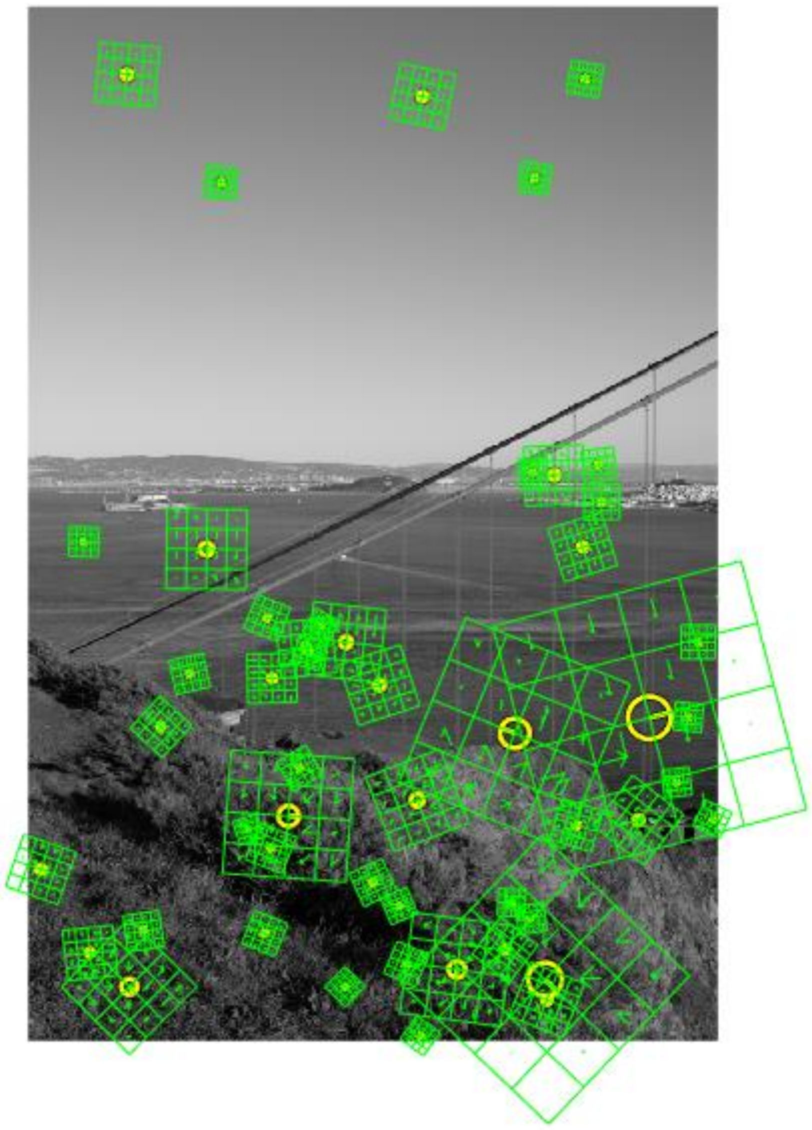


Figure 1:SIFT frames/features on goldengate-00.png

The scale and the local orientation are clearly visible in the figure. The features are randomly selected from the set of all SIFT features.

3.3 Describing Local Features

Descriptors

In this assignment, we are asked to use two different descriptors, one being the classic 1x128 SIFT and the other one being a 1x256 raw pixel-based descriptor based on histograms. The SIFT descriptors are easy to compute with VLFeat, since descriptors are computed for each feature point when feature points are computed.

The implementation for the raw pixel-based descriptor is located in the file raw_pixel_descriptor.m. The descriptor is computed as follows: The image for which the descriptors are to be computed is firstly padded with zeroes to avoid out-of-bounds problems. Then, for every SIFT feature point on the image, an oriented patch with size proportional to the scale of the SIFT feature is considered, and the pixels contained within the patch are arranged on a histogram with 256 bins. In the implementation, the rotation is achieved by considering a patch twice as large as the original patch to be extracted, later to be rotated in the inverse direction of the orientation of the corresponding feature to attain rotation invariance. Then, a patch half the size of the patch in consideration is extracted from the centre of the aforementioned patch. The following images describe the process of extraction:

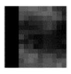


Figure 2: Patch to be rotated

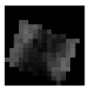


Figure 3: Rotated Patch

Figure 4: Extracted Patch

The descriptor is a 1 by 256 vector, and has the number of entries for the histogram for a pixel of particular value.

Section 3.4: Feature Matching

The feature matching is done via computing and minimizing the Euclidean distance between the feature points of two images for the SIFT descriptor. For the raw-pixel based descriptor, a form of the chi-squared distance is considered.

The feature matching is done via the VLFeat library. The following is the invocation of this function:

```
[matches{i},scores{i}] = vl_ubcmatch(images_d{i},images_d{i+1});
```

For the SIFT descriptor, the following is an example matching between two images in the goldengate dataset.

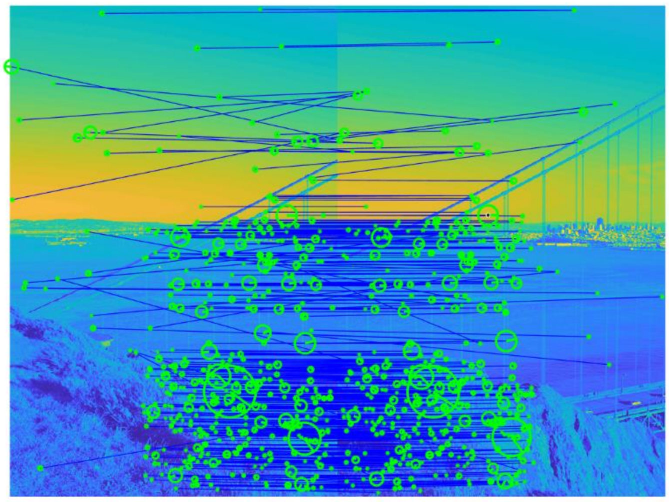


Figure 5: Matched Features

For the raw-pixel based descriptor, the matching is computed in the same manner. The histograms are normalized in the sense that they have their mean subtracted from the values and then scaled by the reciprocal of the standard deviation of its entries. Then the same SSD metric is applied to the 1x256 histogram features for feature matching.

The other important aspect of the implementation is that it simply classifies some of the matches as noisy matches and then discards them. The amount that is discarded can be modified with the threshold parameter of the panorama(3) function.

Section 3.5 Image Registration

Image registration takes place in the file “compute_transforms.m”. Essentially, the script simply tries to find an affine transformation between the matched coordinates of two consecutive images. For this purpose, it uses a method called estimateGeometricTransform(), which is provided by MATLAB. The method uses the RANSAC algorithm to compute a linear regression in the presence of outliers. The following is the set of features plotted for two consecutive images:

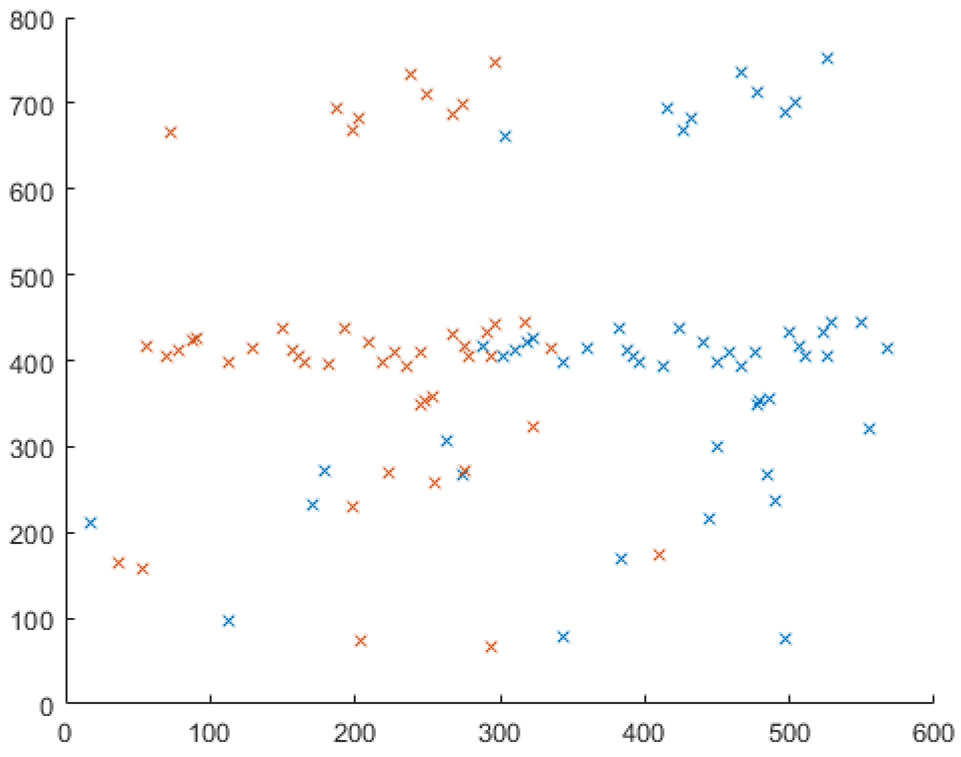
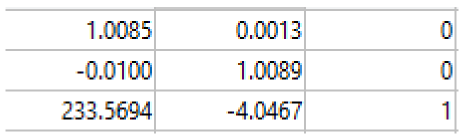


Figure 6: Scatter plot for matched feature coordinates

The following is the transformation computed by the RANSAC implementation provided by MATLAB.



1.0085	0.0013	0
-0.0100	1.0089	0
233.5694	-4.0467	1

Figure 7: Affine transformation between two image frames

After all individual transformations are computed (i.e. the transformation of the second image with respect to the first, the third image with respect to the second and so on), the script located in “stitch.m” cascades these transformations by chain-multiplying them (akin to a matrix stack), so that they may be applied in succession to the images that will be blended in the next stem.

Section 3.6 Blending

Blending takes place in the “stitch.m” file. After the coordinate system transforms are cascaded, the script “applies” the transformation to “ghost images” to estimate the size of the resulting panorama image. The lines 14 to 25 are directly taken from the MATLAB tutorial on feature-based image stitching. The source is cited in the script.

After the size has been estimated, the script initializes an empty black canvas on which the images will be transformed into the coordinate system of the first image, later to be placed on the said canvas. In a loop, the script normalizes the image and then warps the image according to its affine transform. In order to implement blending, the script also creates a mask in the shape of the original image and then warps it in the same way that it warps the image. This way, the pixels on which the image will be placed on are known. There is a separate variable, image_count, which keeps track of how many images have been placed on a particular pixel by simply taking the sum of the masks. At the end of the loop, the luminosity values of the pixels are normalized by dividing its pixels elementwise by the number of images that have been placed at that particular pixel. A simple modification to this algorithm may result in better blending: the overlap can be weighted inversely proportional to the distance to the center of an image using bilinear interpolation.

The following are the results of the blending/stitching for SIFT and raw pixel-based-descriptors.

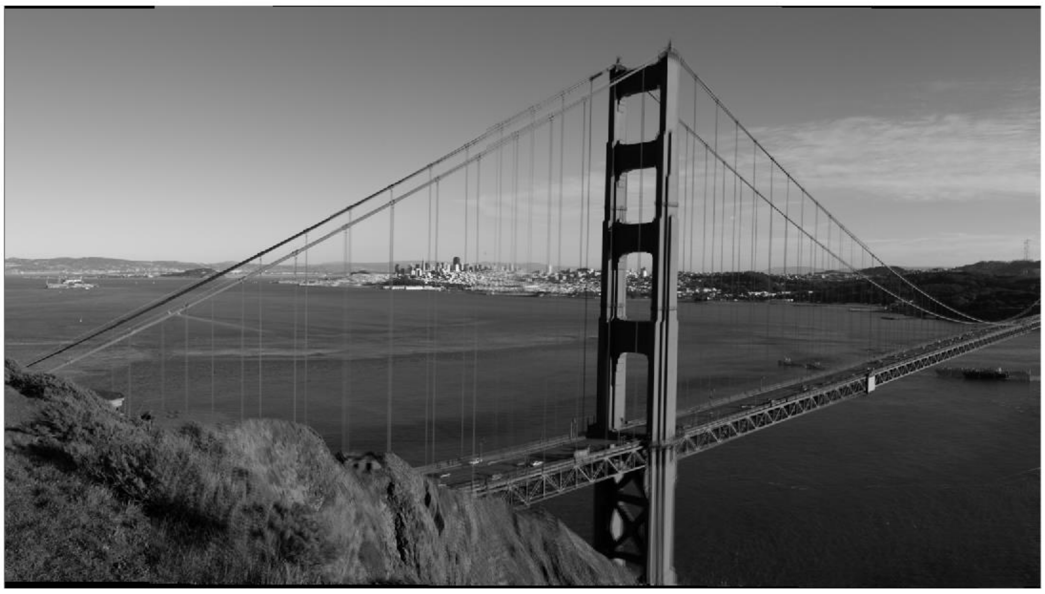


Figure 8: Goldengate stitched, using SIFT

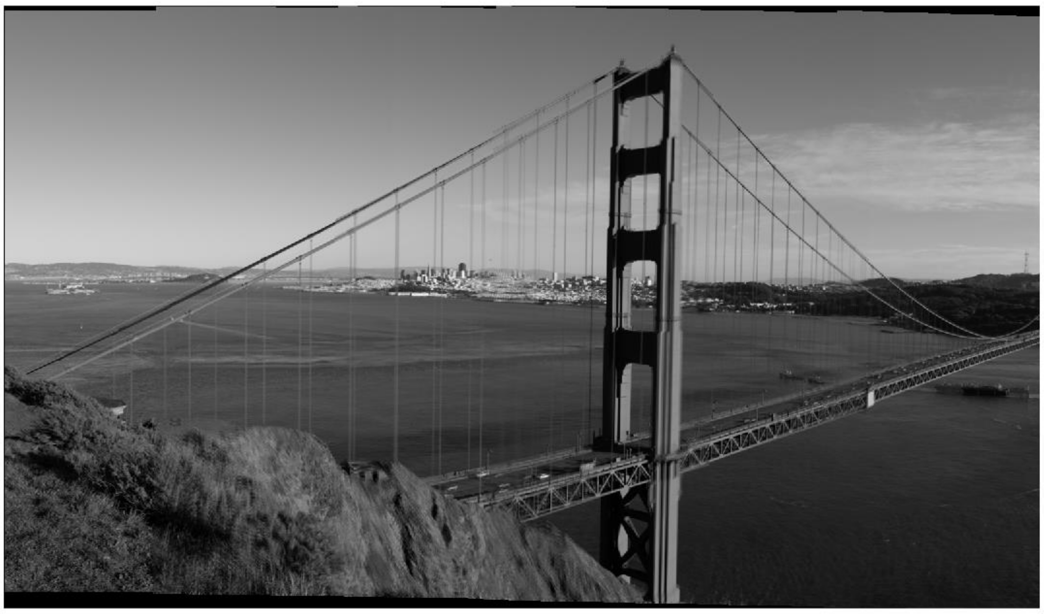


Figure 9: Goldengate stitched, using raw pixel values

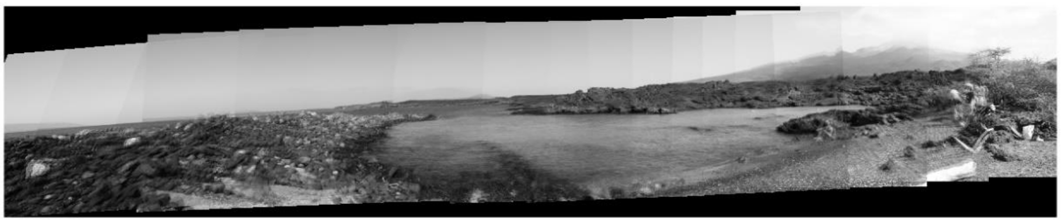


Figure 10: Fishbowl stitched, using SIFT

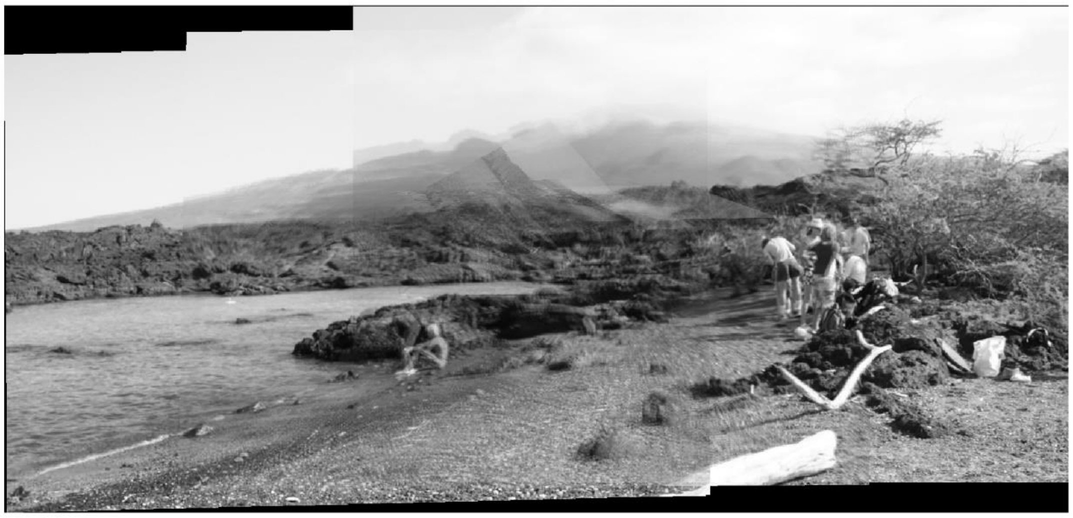


Figure 11: FIshbowl stitched, using raw-pixel data

Important:

To replicate these results, call the panorama function with the parameters below:

```
stitched = panaroma('goldengate\goldengate','sift',500);
stitched = panaroma('fishbowl\fishbowl','sift',900);
stitched = panaroma('goldengate\goldengate','raw',inf);
stitched = panaroma('fishbowl\fishbowl','raw',inf);
```

The thresholding parameter for the raw descriptor is infinity, this is because otherwise, the number of matched features is somewhat low. Having it as inf means that one is considering all the features possible.

Section 3.7 Discussion

Regarding the effect of the choice of the descriptor on the final output, it must be stated that choosing the raw-pixel based descriptor yields far better results in comparison to the SIFT descriptor for the goldengate dataset. This is because of the fact that the illumination throughout the scene stays somewhat relatively the same in between images, which means that in this case the descriptor is quite robust for stitching such groups of images. For the fishbowl dataset, however, there are lots of illumination changes towards the middle of the scene, as well as the textural repetition of the coast, which comes off as a repetition of edges, which results in “ghost” matches with the other descriptors. As seen in Figure 11, this causes very weak transformations in the middle, and lots of images with the coast are super-imposed on top of one another. Meanwhile, the SIFT descriptor does just fine, since it is illumination invariant, whereas the raw-pixel descriptor is not.

As for the simplicity of stitching, the goldengate dataset is much easier to stitch due to the lack of illumination variations, along with the fact that it has little tangential and angular distortion during the capture of the scene. Fishbowl on the other hand, should not even be modelled using affine transformations, since the camera clearly has some oblique movements. It would be far easier to register the images using a planar/projective transform.

<u>Difficulty of tasks:</u>

The assignment was not extremely difficult, however the raw-pixel based descriptor is quite possibly the worst descriptor that one can implement: it rarely works, it has too many boundary cases, and it is a bloody nightmare to work with it! Please spare the others the pain.