Lab Description:

The Scale-Invariant Feature Transform (SIFT) bundles a feature detector and a feature descriptor. The detector extracts from an image a number of frames (attributed regions) in a way which is consistent with (some) variations of the illumination, viewpoint and other viewing conditions. The descriptor associates to the regions a signature which identifies their appearance compactly and robustly.

![Original image](image1)
![Piece 1](image2)
![Piece 2](image3)

Figure 1: Example of using SIFT and descriptor matching to find the correct translation and rotation for puzzle matching.

The image shown above shows an example of using SIFT to find the correct translation and rotation for the pieces.

**Step 1.** We will be using VLFeat for this lab. Download VLFeat from vlfeat.org. The VLFeat open source library implements popular computer vision algorithms specializing in image understanding and local features extraction and matching. Algorithms include Fisher Vector, VLAD, SIFT, MSER, k-means, hierarchical k-means, agglomerative information bottleneck, SLIC superpixels, quick shift superpixels, large scale SVM training, and many others. It is written in C for
efficiency and compatibility, with interfaces in MATLAB for ease of use, and detailed documentation throughout. It supports Windows, Mac OS X, and Linux. The latest version of VLFeat is 0.9.20.

**Step 2.** Setup VLFeat. You will first need to navigate to the toolbox folder and run vl_setup in the command window. This will set up the environment needed to run the VLFeat commands. Read in the image as follows. The vl_sift command requires a single precision gray scale image. It also expects the range to be normalized in the [0,255] interval (while this is not strictly required, the default values of some internal thresholds are tuned for this case). The image I is converted in the appropriate format by

```matlab
>> im1 = single(rgb2gray(im));
```

**Step 3.** Compute the SIFT descriptors for the puzzle. We compute the SIFT frames (keypoints) and descriptors by

```matlab
>> [f, d] = vl_sift(im1,’PeakThresh’,5);
```

The matrix f has a column for each frame. A frame is a disk of center f(1:2), scale f(3) and orientation f(4). We visualize the features by:

```matlab
>> figure; imshow(im); holdon; h1 = vl_plotframe(f(:, sel));
```

Play with the PeakThresh value and see what it does to the number of SIFT points.

![Figure 2: Example of SIFT descriptor visualization using a threshold of 5.](image)

**Step 4.** Compute the SIFT descriptors for the pieces by following the instructions above.

**Step 5.** Compute the matches using the following method. For each descriptor in da (da being the descriptor d from above which is the output of vl_sift), vl_ubcmatch finds the closest descriptor in db (descriptor from a different image) (as measured by the L2 norm of the difference between them). The index of the original match and the closest descriptor is stored in each column of matches and the distance between the pair is stored in scores.
Matches also can be filtered for uniqueness by passing a third parameter to `vl_ubcmatch` which specifies a threshold. Here, the uniqueness of a pair is measured as the ratio of the distance between the best matching keypoint and the distance to the second best one (see `vl_ubcmatch` for further details).

```matlab
>> [matches, scores] = vl_ubcmatch(da, db, 1.5);
```

Alter the ratio of the distance to the second best one, to refine your matching method.

**Step 5.** Visualize the results. Use the following code to visualize the results. You may need to rework your variable names to match the code.

```matlab
>> numMatches = size(matches, 2)
>> dh1 = max(size(im2, 1) - size(im, 1), 0);
>> dh2 = max(size(im, 1) - size(im2, 1), 0);
>> figure; clf;
>> imagesc([padarray(im1, dh1, 'post') padarray(im2, dh2, 'post')]);
>> o = size(im, 2);
>> line([f1(1, matches(1, :)) f2(1, matches(2, :)) + o], [f1(2, matches(1, :)) f2(2, matches(2, :))]);
>> title(sprintf('%d tentative matches', numMatches));
>> axis image off;
```

**Step 6.** Calculate the displacement of piece 1 and piece 2 from the original image using the `f1`, `f2` and `matches` variables e.g. something like this `f1(1, matches(1, :))` grabs the x positions of the keypoints in image 1. How much do you have to move piece 1 to fit it into the puzzle (x and y displacement)? How much do you have to rotate piece 1 to fit it into the puzzle? What about piece 2, translation and rotation? (Hint, rotation is in radians, and should be something close to 0 and 12 degrees. X, Y translation varies, from almost 0 to 4-500 pixels).

**Deliverables:** Submit on Blackboard.