Computer Vision Homework 1

Oron Barazani 305295818 & Roee Hadar 206390098

Part 1:

1.1 - The picture loaded is shown below:

1.2 - The visualization of the Gaussian Pyramid is added as a picture below:

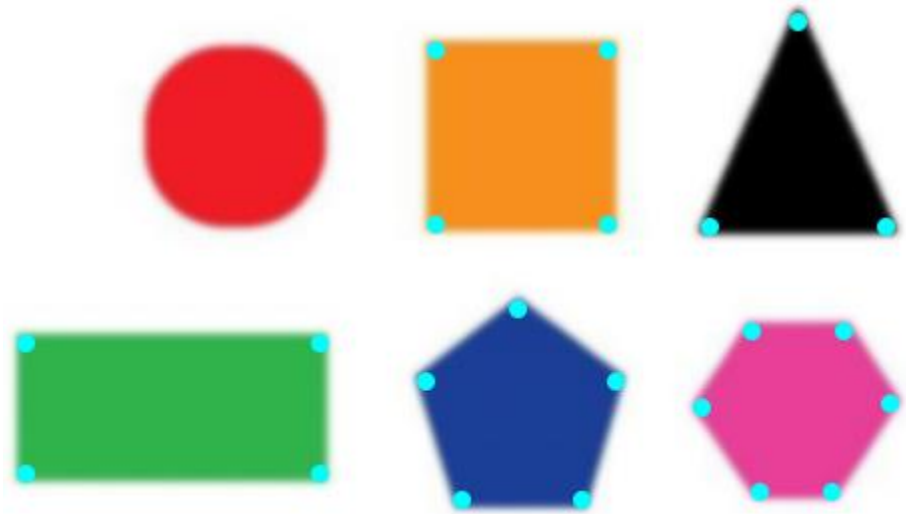
Moreover, the shape of the pyramid provided is $(6, 139, 98)$
As can be seen, each level of the pyramid makes the picture provided be blurrier. This is because of the key feature of the gaussian pyramid, which samples the picture with a gaussian kernel with a variation that changes in each layer. The pictures above are the outputs from each variation layer.

1.3 Creating the DoG pyramid from the given gaussian pyramid resulted in shape of $(5, 139, 98)$. As we can see, the height and width are the same as the original image. Moreover, each layer provides a blurrier image, as explained in the previous section.

1.4 - For the principal curvature calculation, we calculated the differentials of the DoG pyramid, developed the hessian matrix which represented the pyramid derivatives, calculated the eigenvalues of the matrix and used the equation given to us in the lecture $\frac{(\lambda_1+\lambda_2)^2}{\lambda_1\lambda_2}$ to achieve the curvature ratio R for each point.

1.5 - To check for extrema points we implemented a helping function called $check\_neighbors()$ which checks for extremum compared to its spatial and scale neighbors. It determines the neighbors scale, gets special neighbors in adjacent levels and current level, and of those neighbors determines the overall extremum using the extremum conditions. The main function initializes the '$locsDoG$' list which stores the detected extrema points, iterates over levels and spatial coordinates and checks thresholds and local extrema using the $check\_neighbors()$ function explained above.

1.6 - The results on the sanity check picture are shown below:



As can be seen, the edges were detected as expected.
For a real image, the results are added below:

As we can see in the real world, unlike simple geometric shapes there are many situations in which the algorithm fails to detect precisely the edges. However, considering the key features in the provided picture, such as changes between the car and the background, the detection of the wheel, the door to the right and the stand in the middle of the picture - the algorithm successfully shows the corners and edges. Due to changes in brightness on the floor, combined with the shape of the floor, the algorithm detects many points of interest where each brick ends, as expected from the algorithm.

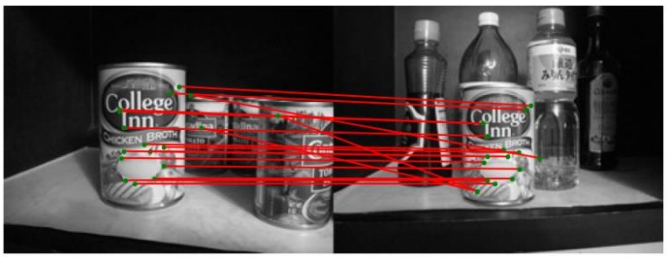
Part 2

2.1 We draw $nBits = 256$ samples for x, y pairs from discrete uniform distribution $x, y \sim U[0, patchWidth^2)$ where $patchWidth = 9$ and saved the test pattern for later use.

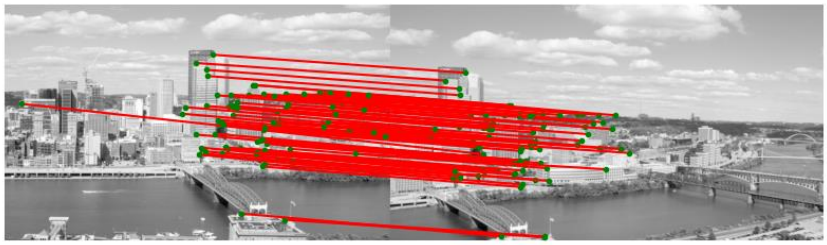
2.2 We wrote $computeBrief()$ function that check for each feature point check if it is in appropriate region (i.e. not in the margin) and then compute its binary descriptor according to the sample pattern from 2.1

2.3 We combined feature extraction form part 1 (DoG detector) to construct $briefLite()$ function which take image as input, detect its key-points, and compute appropriate descriptor for each key-point.

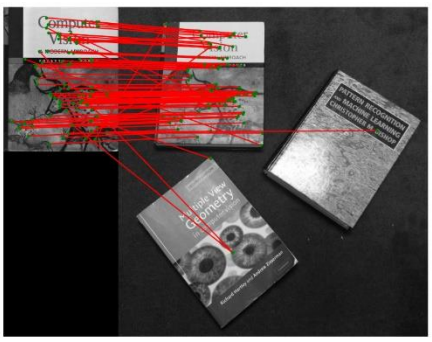
2.4 We used the provided functions $briefMatch()$ and $plotMatches()$ to test the new $briefLite()$ descriptor that we constructed in 2.3. Here are few examples:



In the first example we can see there is a match between the chicken broth can only, because it's the only item that appears in both pictures. Around 500 features were extracted through DoG detector and BRIEF managed to pick only those who are matching in both images with great success.



Same things happen for incline_L/R images. Only the portion of the image that appears in both, is being correctly match. This is pretty challenging for any feature matching mechanism, because the city has pattern of building which are difficult for the algorithm to comprehend and could be easily confused with one another. Another thing we noticed is the long run time for larger images, which may be problematic (42.1 [s])



In the book matching section, the books had good matching, where they were about the same orientation as the example image. On the other hand, the images where the book was rotated the matching was poor:

This is not surprising, because BRIEF is not rotation invariant algorithm.

2.5 Questions from the notebook:

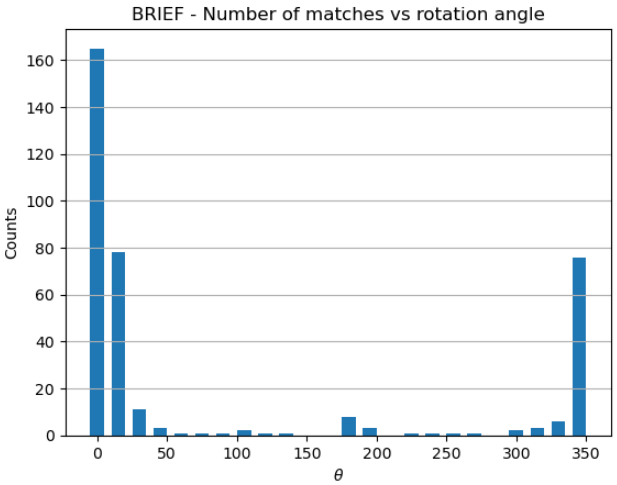
Q: Is BRIEF invariant to illumination changes?

A: No. BRIEF is relying on spatial intensity comparison which may be sensitive to illumination changes. The descriptor can be changed greatly for small intensity variation.

Q: Assuming we want to boost the invariance of BRIEF to scale, which part of the SIFT algorithm should we include?

A: We can add another dimension to the DoG pyramid and create extended scale-space - $DoG(x, y, \sigma, s)$ instead of ours $DoG(x, y, \sigma)$. The different s levels, called octaves, are subsampling of original image. This allows us to deal with scale variance of the features.

In the next part, we took the chicken broth image and test BRIEF rotation invariance, which was poor. The number of matches is higher for only small rotation. The matching is poor because the descriptor is relying on spatial sampling pattern which is very sensitive to rotation. There is an interesting peak around $180°$ rotation maybe because the can is symmetrical in the X axis.



2.6 The key difference that makes ORB rotation invariant is using descriptor rotation. It takes each feature and using rotation matrix to rotate the descriptor K times in 12 degrees increments. This way, each feature, has a set of K descriptor orientation, and become rotational invariant. This method of assigning orientation for each descriptor is inspired by SIFT.
There is interesting pattern in the bar graph: there are peaks at $90°$ intervals. Maybe because that other rotation angle requires interpolation of the rotated points, while in $90°$ it doesn't.