ee046746: Computer Vision
HW-1
Features Descriptors

Daniela Boguslavsky 315720003
Shahar Rashty 312465305

Due date: 20.4.23

Part 1 - Keypoint Detector

1.1 Load image:

1.2 Gaussian pyramid :

What is the shape of GaussianPyramid matrix? (6,139,98)

```
GaussianPyramid.shape

(6, 139, 98)
```

6=number of levels at the pyramid

139X98 = image size, Hight* Width (same as the input image at gray scale).

In the results we got 6 pictures that each one represent a level in the Gaussian Pyramid. As the level increases, the image become more blurred due to the blur created by a Gaussian kernel.

1.3 DoG pyramid:

We implement createDoGPyramid – the function gets a pyramid of Gaussians, as implemented in createGaussianPyramid (each level of the pyramid contains the image convolved with gaussians with different sigma, each sigma is larger than the previous one) and levels of the pyramid . CreateDoGPyramid returns the difference between each two consecutive levels of gaussians which can be used as estimation to the Laplacian operator.

1.4 Edge Suppression

Since edges are not desirable for feature extraction as they are not as distinctive and do not provide a substantially stable localization for keypoints we implement a function to remove them from our features.

The function takes in DoGPyramid generated in the previous section and returns PrincipalCurvature, a matrix of the same size where each point contains the curvature ratio R for the corresponding point in the DoG pyramid in order to so we first calculated the Hessian using Sobel:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Then find R:

$$R = \frac{TR(H)^2}{Det(H)} = \frac{(\lambda_{min}+\lambda_{max})^2}{\lambda_{min}\lambda_{max}}$$

We added small epsilon (1e-8) to Det(H) to avoid division by zero and replaced all the negative determinant to epsilon so they will be removed when we will apply a threshold $R>\theta_r$.

1.5 Detecting extrema:

To detect corner-like, scale-invariant interest points, the DoG detector chooses points that are local extrema in both scale and space – we checked 8 “space neighbors” on the same level of the pyramid and 2 scale neighbors-pixel of the same location from adjacent levels.

For the first and last levels we only checked 1 scale neighbor.

In order to do that we created a list with all relevant neighbors, we found the max and min values of the list and compared them with the current pixel for checking if the current pixel is an extremum. Then we checked the two thresholds as described here:

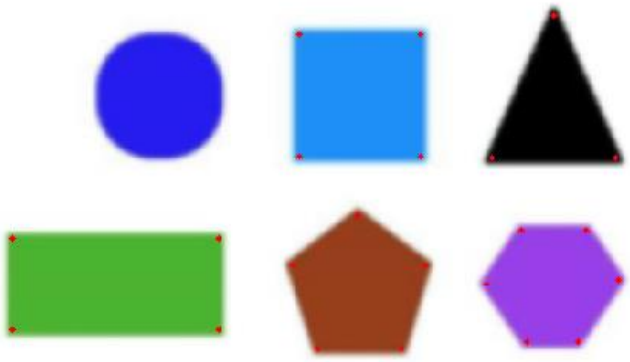
The threshold “theta_c” should remove any point that is a local extremum but does not have a Difference of Gaussian (DoG) response magnitude above this threshold. The threshold theta_r should remove any edge-like points that have too large a principal curvature ratio specified by “PrincipalCurvature”.

If the pixel Is larger than th_c and smaller than th_r we add it to the final output array.

1.6 putting it together:

The function DoGdetector, gathers all the functions from this question: createGaussianPyramid, createDoGPyramid, omputePrincipalCurvature and getLocalExtrema. The inputs are: image, sigma0, k, levels, th_contrast=0.03, th_r=12 and the output is the the local extrema locations and a Gaussian Pyramid.

Then we wrote a function “display_circles” to display the image and draw circles at each of the coordinates that we found , here is the results for sanitycheck image :



As we can see we managed to detect all corners and only them ,as this is fairly “simple” image.

The result we got for one of the real images provided:

And one we took outside:

We can see that this image contains much more corner and not all of them were recognized by our keypoint detector, we can try improve the results by fine tunning the thresholds.

For example here we want to get more keypoint so we should decrease th_contrast or increase th_r , we changed to - th_contrast=0.01, th_r=15 (instead of th_contrast=0.03, th_r=12) , and got:

Part 2 BRIEF Descriptor

2.1 Creating a Set of BRIEF Tests

We implemented makeTestPattern which takes in patchWidth-width of the patch, nbits-number of tests in the BRIEF descriptor and returns 2 vectors of nbits called compareX and compareY - linear indices into the patchWidth × patchWidth size .

We chose to implement the first method suggested at the article, (X, Y) ∼ i.i.d. Uniform(− S/2 , S/2) The (xi , yi) locations are evenly distributed over the flatten patch and tests can lie close to the patch border.

We then used this function to create the vectors and then saved them as 'testPattern.mat' which will be loaded later to use a fixed set of coordinates.

2.2 Compute the BRIEF Descriptor

Implemented computeBrief(im, GaussianPyramid, locsDoG, k, levels, compareX, compareY, patchWidth) , which compute the BRIEF descriptor for the detected key points.

First we loaded the test points from 'testPattern.mat' , and we looped over the keypoints ,

In each iteration we extract the key point location and level (at the DoG pyramid) from locsDoG, then we check if patch is within image bounds – checked if the point location +- half the patch width are within the image border. if it is , we extracted a patch from the image gray values around the key point location, flatten the patch and finally created the BRIEF descriptor by using the test points compareX, compareY as follow :

$$\rho(p; x, y) := \begin{cases} 1, & \text{if } p(x) < p(y) \\ 0, & \text{otherwise.} \end{cases}$$

$$x, y \in N^{S^2}$$

$$p \in R^{S^2}$$

We added the descriptor to an array called desc and the (x, y, level) to an array called locs.

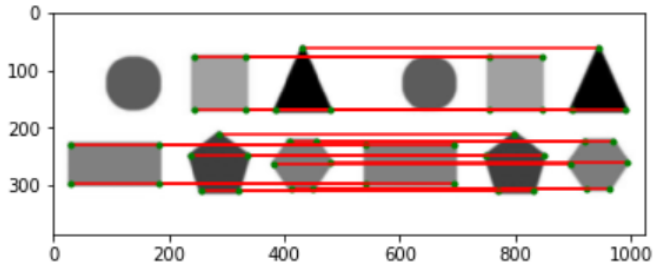
2.3 Putting it all Together

Implemented briefLite(im) which uses previous functions. It takes a gray image uses DoGdetector to get locsDoG, GaussianPyramid, then uses computeBrief to get location of key points and descriptors and returns them.

2.4 Check Point: Descriptor Matching

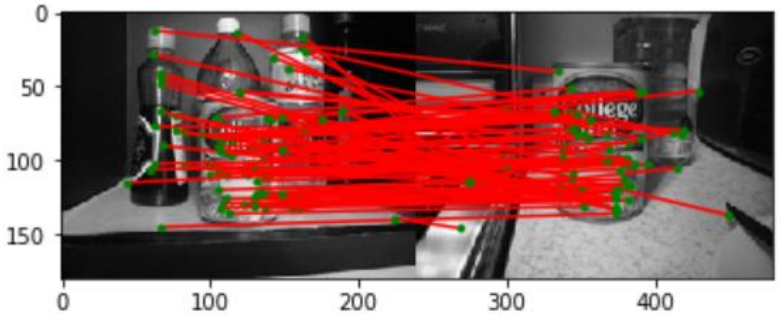
testMatch(im1,im2) takes 2 images converts them to gray and uses briefLite to get for each image the key points and descriptors. Then, uses briefMatch which performs the descriptor matching and finally uses plotMatches to draw two images side by side with their matching points.

First we performed sanity check, we tested match of a simple image with itself.

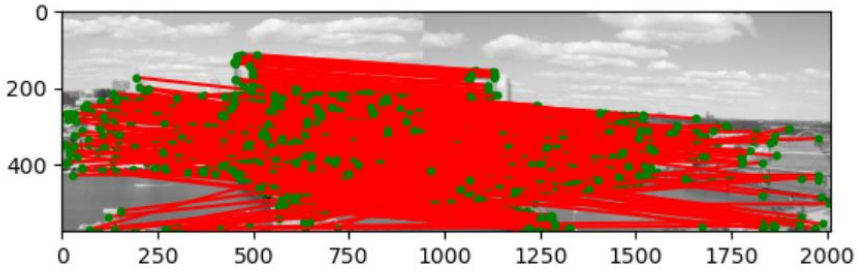


We can see that the BRIEF detector matched each point perfectly!

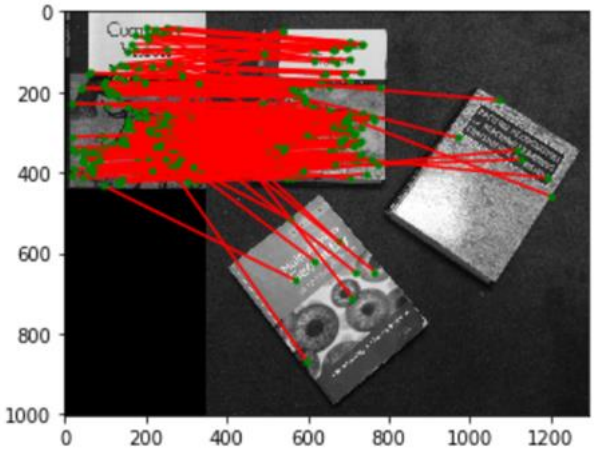
Then we tested more “complicated” images. The results:



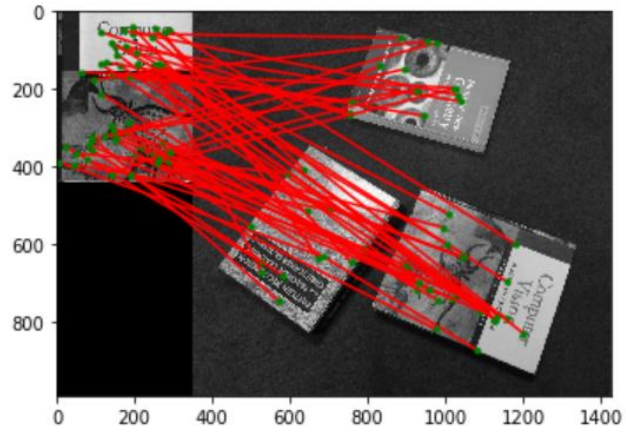
chickenbroth_02 Vs chickenbroth_03



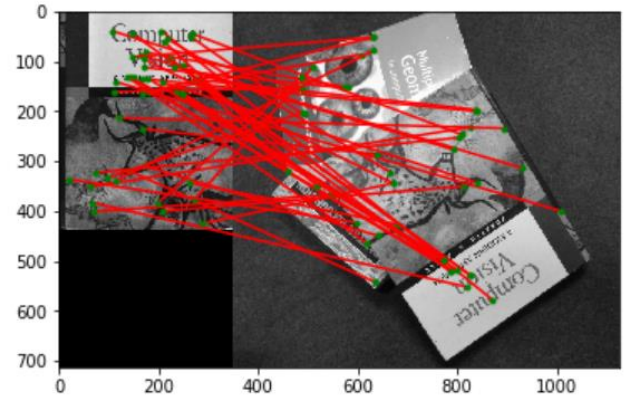
incline_L Vs incline_R



computer vision textbook cover page: scan scaled Vs floor



computer vision textbook cover page: scan scaled Vs floor rotate



computer vision textbook cover page: scan scaled Vs floor pile

After examine all the cases, it seems that the cases where the objects were rotated we got the worse results. We can see that espacially in the cases of the computer vision textbook cover page. It can be assumed that the BRIEF detector is not invariant to rotation,it is affected by rotation, we will examine than in the next sections. The best result were received for incline_L Vs incline_R, where there were many points that match in both images. We essume that the BRIEF detector is invariant to incline changes.

2.5 BRIEF properties

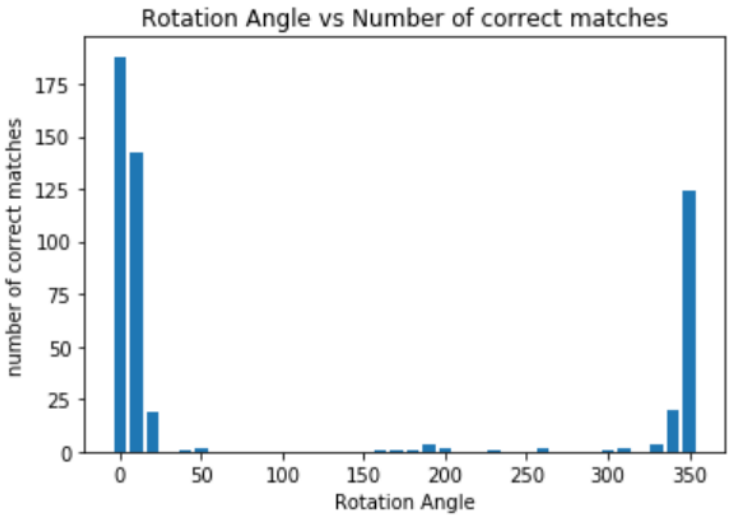
After testing the performance of BRIEF on the provided sample images, we can conclude that BRIEF is invariant to illumination changes. This is because BRIEF is constructed using the second derivative of an image, which only takes into account the differences between pixels and not their original brightness values.

To boost the invariance of the BRIEF to scale, we should include the Scale-Space Extrema Detection step from the SIFT algorithm. In this step a scale space pyramid of the image is created by smoothing the image with Gaussian filters repeatedly at different scales and then down sampling it. The extrema in this scale space pyramid are then detected by comparing each pixel with its neighbour pixels at the same and adjacent scales.

The Scale-Space Extrema Detection step helps to identify features that are invariant to scale changes as it detects key points at different scales and identifies the scale at which they are most stable.

In this section we wrote test script 'briefRotTest' that counts the number of correct matches of the image 'model chicken broth' as a function of rotation angle.

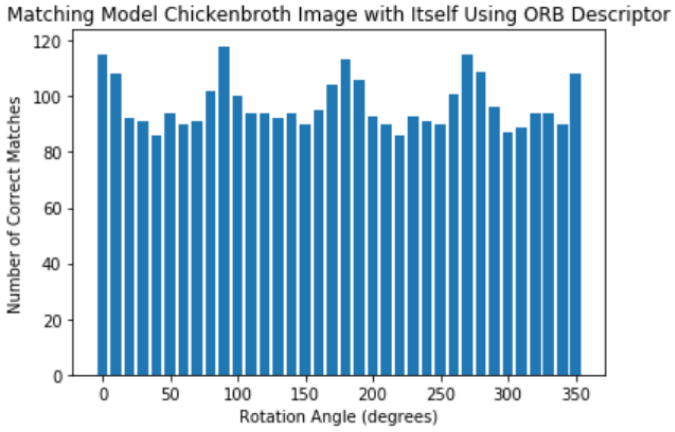
First we wrote the 'MakeMatch' function that takes in two images and returns the variables: locs (key points locations), desc (BRIEF descriptors) and matches as described in previous sections. Then, the function 'briefRotTest', takes in am image and a list of rotation angles and applies the given functions rotateImage, MakeMatch and checkRotLocs. Finally for each angle the number of correct matches is added to the correct match list. With this function we were able to construct a bar graph showing rotation angle vs the number of correct matches.



Acoording to the bar graph,we can see dramatic changes in the number of correct matched when changing rotation angle,we can see that for 0 degrees,no rotation,we got about 190 correct matches,for 10 degrees to either side (10 or 350 degrees rotation) we still got large number of matches,but when the image was rotated by more than 10 degrees the number of correct matches is much smaller. Thus, we can conclude that the BRIEF descriptor is not invariant for rotation.

2.6 Oriented Fast and Rotated BRIEF (ORB)

In this part we are using the ORB descriptor. We wrote a test script 'orbTest' that counts the number of correct matches of the image 'model chicken broth' as a function of rotation angle using the ORB descriptor. The bar graph showing rotation angle vs the number of correct matches that we have received using the ORB descriptor:



The result has shown substantial improvement where the angle increases. This is because ORB adds two main components to BRIEF. The first one is Oriented FAST (Features from Accelerated Segment Test). The FAST algorithm detects features by analysing the brightness values of surrounding pixels. It extends this by considering the orientation of the features as well. This allows ORB to detect and extract features at different orientations. The second one is extension includes rotating the pattern used to compare the pixel intensities. This ensures that the same descriptor can be used for features with different orientations, making ORB more robust to image rotation.

While ORB and SIFT are different algorithms, the ORB does borrow some components from SIFT. For example: SIFT detects key points at different scales and the ORB as well. SIFT uses a difference-of-Gaussian scale space pyramid and the ORB using a similar approach, although it uses a different method for scale detection. Another example is SIFT assigns an orientation to each key point based on the dominant orientation of gradient magnitude in its local neighbourhood. ORB also assigns an orientation to key points using the Oriented FAST.