Orhun Görkem

Computer Vision Assignment 1 Report

1. Detection

1.1. Image Gradients

For gradient I_x(i,j), I(i,j+1) - I(i,j-1) should be calculated and divided by two for each pixel. Therefore, I used:

0	0	0
-0.5	0	0.5
0	0	0

as a kernel for convolution. With the same approach, I used:

0	-0.5	0
0	0	0
0	0/5	0

kernel to calculate I_y.

```
scipy.signal.convolve2d(img, kernel, mode='same', boundary='symm')
```

The piece of code is used to obtain image gradients. The dimensions of gradient matrices are the same as the dimensions of the given image.

1.2. Local Auto-correlation Matrix

$$M_p = \sum_{p' \in \mathcal{N}(p)} w_{p'} \begin{bmatrix} I_x(p')^2 & I_x(p')I_y(p') \\ I_y(p')I_x(p') & I_y(p')^2 \end{bmatrix}$$

To follow the formula of local auto-correlation matrix, the 2x2 auto-correlation matrix for each pixel is calculated from previously calculated gradients. These 2x2 matrices are accumulated in a 4d matrix called M which is initiated as follows.

```
M = np.zeros((len(img), len(img[0]), 2, 2))
```

Then I filled M with a nested for loop.

```
cov = np.array([[I_x[i][j] ** 2, I_x[i][j] * I_y[i][j]],[I_y[i][j] * I_x[i][j],
I_y[i][j] ** 2]])
M[i][j] = cov
```

Now Gaussian weights should be applied to each M[i][j]. I decided to apply Gaussian Blur with a 3x3 kernel.

```
cv2.GaussianBlur(sliced, (kernel_size,kernel_size), sigma, sigma,
cv2.BORDER_REPLICATE)
```

GaussianBlur function of OpenCV works with 2d matrices. So I needed to get slices from previously filled 4d M matrix. So I obtained M[i][j][0][0], M[i][j][0][1], M[i][j][1][0], and M[i][j][1][1] for all i,j respectively. Then applied the function above for each slice.

1.3. Harris Response Function

```
C = np.array([[np.linalg.det(M[i][j]) - k * np.trace(M[i][j])**2 for j in
range(len(img[0]))] for i in range(len(img))])
```

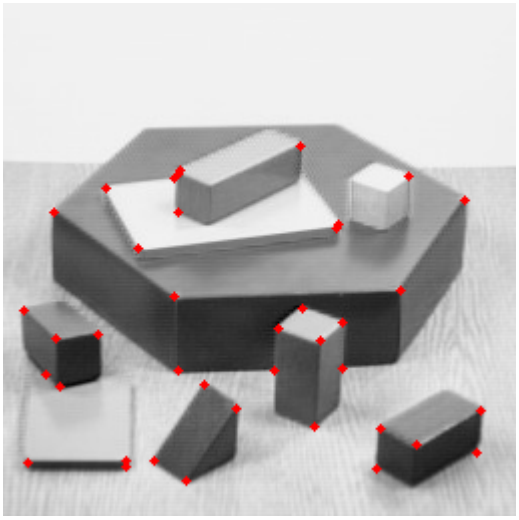
Matrix C is calculated according to the given Harris response formula.

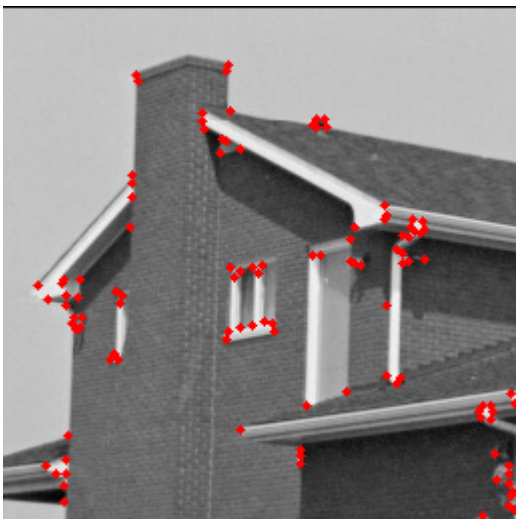
1.4. Detection

```
neighborhood = np.ones((3, 3), dtype = bool)
filtered = (scipy.ndimage.maximum_filter(C, footprint=neighborhood) == C)* 1
```

The code above filters the response matrix and assigns the local maximum indices to 1. Then for each pixel, I checked if the response exceeded the threshold and filter assigned the index to 1. The pixels that satisfy the conditions are added to corners. Corners and C are returned.

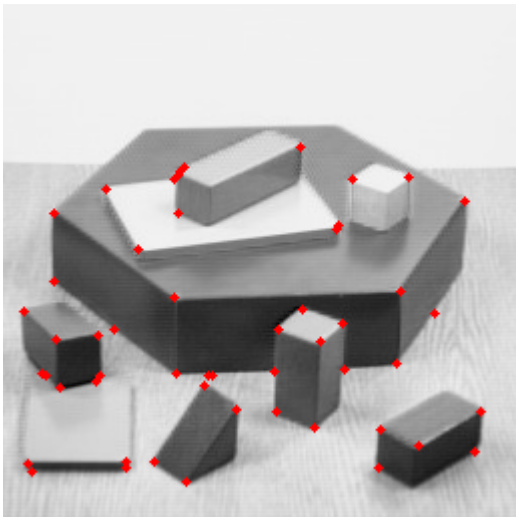
Orhun Görkem

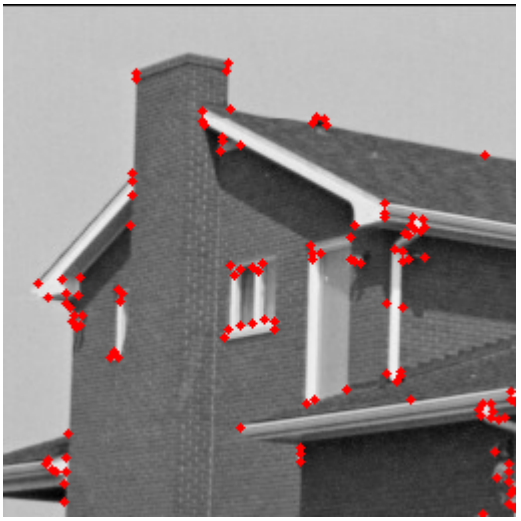




```
HARRIS_SIGMA = 1.0
HARRIS_K = 0.05
HARRIS_THRESH = 1e-5
MATCHING_RATIO_TEST_THRESHOLD = 0.5
```

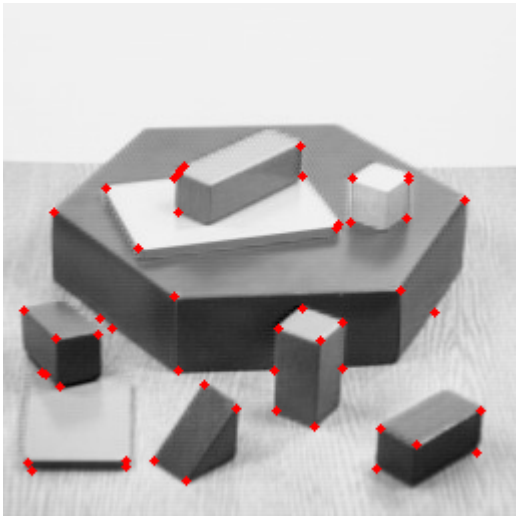
Best result

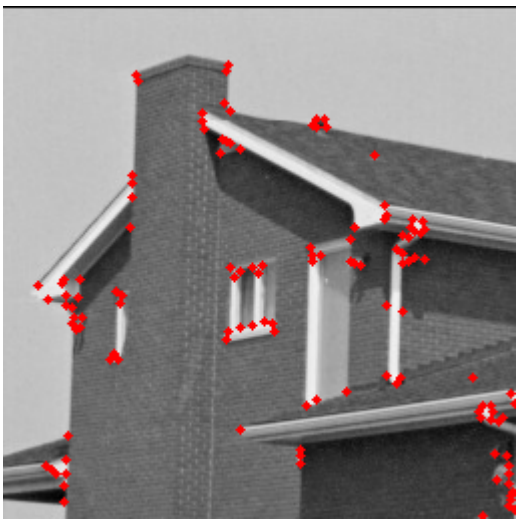




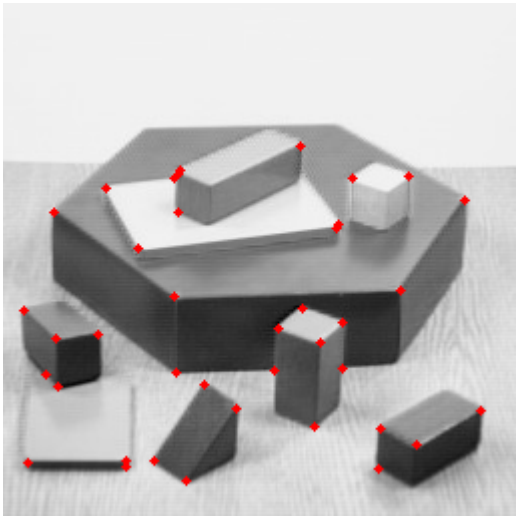
```
HARRIS_SIGMA = 1.0
HARRIS_K = 0.03
HARRIS_THRESH = 1e-5
MATCHING_RATIO_TEST_THRESHOLD = 0.5
```

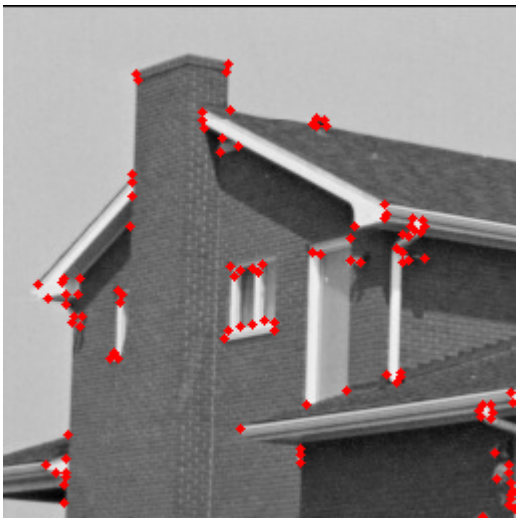
Orhun Görkem



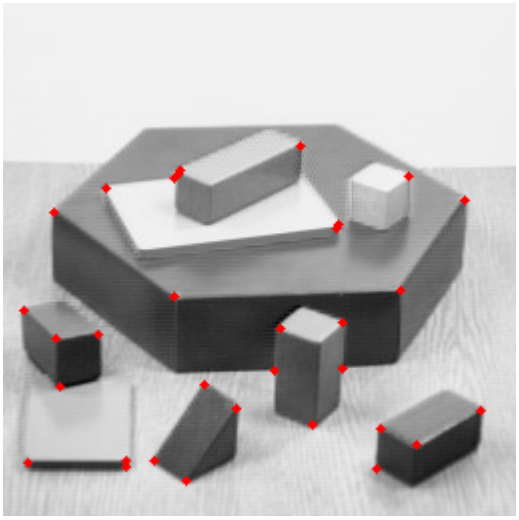


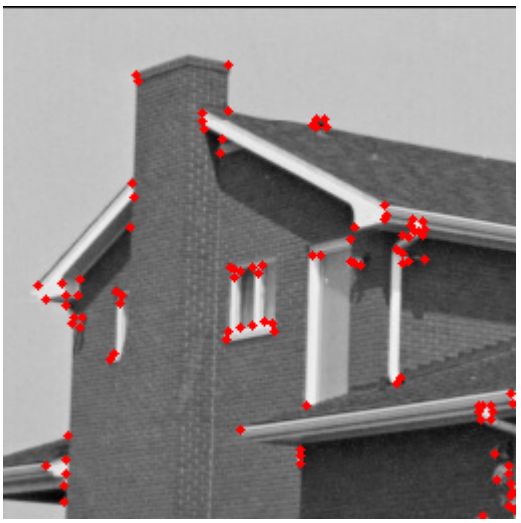
```
HARRIS_SIGMA = 1.0
HARRIS_K = 0.05
HARRIS_THRESH = 5e-6
MATCHING_RATIO_TEST_THRESHOLD = 0.5
```





```
HARRIS_SIGMA = 2.0
HARRIS_K = 0.05
HARRIS_THRESH = 1e-5
MATCHING_RATIO_TEST_THRESHOLD = 0.5
```





```
HARRIS_SIGMA = 1.0
HARRIS_K = 0.07
HARRIS_THRESH = 1e-5
MATCHING_RATIO_TEST_THRESHOLD = 0.5
```

When the results are analyzed, we see that if HARRIS_K, HARRIS_SIGMA, HARRIS_THRESH parameters are decreased, the number of key points increases, and the model becomes more assertive and faulty. When the parameters are increased, the number of keypoints decreases. The optimal tradeoff is having as many as key points while avoiding misjudgments.

Sigma parameter determines the blurriness of the M matrix. If it increases, the sharpness of the analysis decreases. This might cause the decrease in the number of keypoints.
Harris thresh parameter is the boundary of acceptance for key points. If we increase it too much, it means that we are expecting too much from points to be key points. So the number of key points decreases. K parameter also reduces the response of pixels, so it has a negative correlation between the number of key points.

2. Description and Matching

2.1. Local Descriptors

```
if 0 <= key[0]-patch_size <= key[0]+patch_size < len(img):
            if 0 <= key[1]-patch_size <= key[1]+patch_size < len(img[0]):
                filtered.append(key)
```

The code is used to eliminate keypoints that are too close to boundaries.

Orhun Görkem

2.2. SSD One-Way Nearest Neighbors Matching

$$SSD(p,q) = \sum_i (p_i - q_i)^2 \ .$$

The formula of SSD is given.
Vector interpretation: (p-q)^2 = p.p + q.q - 2.p.q

```
d1_d1 = (desc1*desc1).sum(axis=1).reshape((M,1))*np.ones(shape=(1,N))
d2_d2 = (desc2*desc2).sum(axis=1)*np.ones(shape=(M,1))
return d1_d1 + d2_d2 -2*desc1.dot(desc2.T)
```

In the code above, d1_d1 keeps the dot products of each vector in desc1 with itself. (It is like p.p but extended to the shape of given matrix with np.ones) Same holds for d2_d2. Then dot product of each vector in rows of desc1 and in columns of desc2 is multiplied with 2 and subtracted.
The resulting matrix accumulates the distances of each vector pair in descriptors. In other words:

```
distances = ssd(desc1, desc2)
```

distance[i,j] is the ssd between the descriptor of the first image and the jth descriptor of the second image.

For one-way matching:

```
col = np.argmin(distances[i])
matches += [[i,col]]
```

rows are iterated in distances. The minimum argument of each row is assigned to the descriptor of the row.

2.3. Mutual Nearest Neighbors / Ratio Test

For mutual nearest neighbors mathing:

```
col = np.argmin(distances[i])
if np.argmin(distances[:,col]) == i:
     matches += [[i,col]]
```

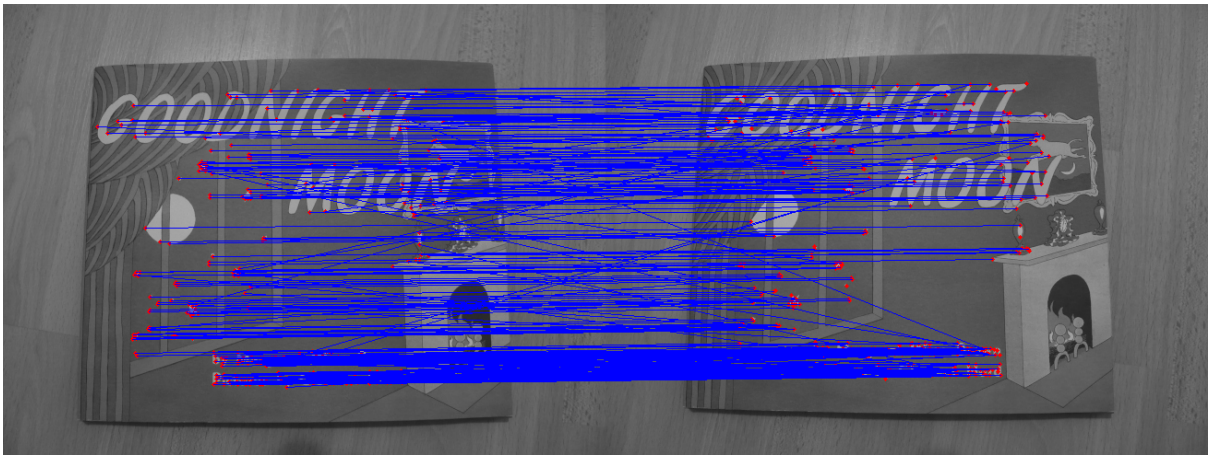
The minimum argument of the column is also checked to see if the matching is mutual.

When ratio test is added:

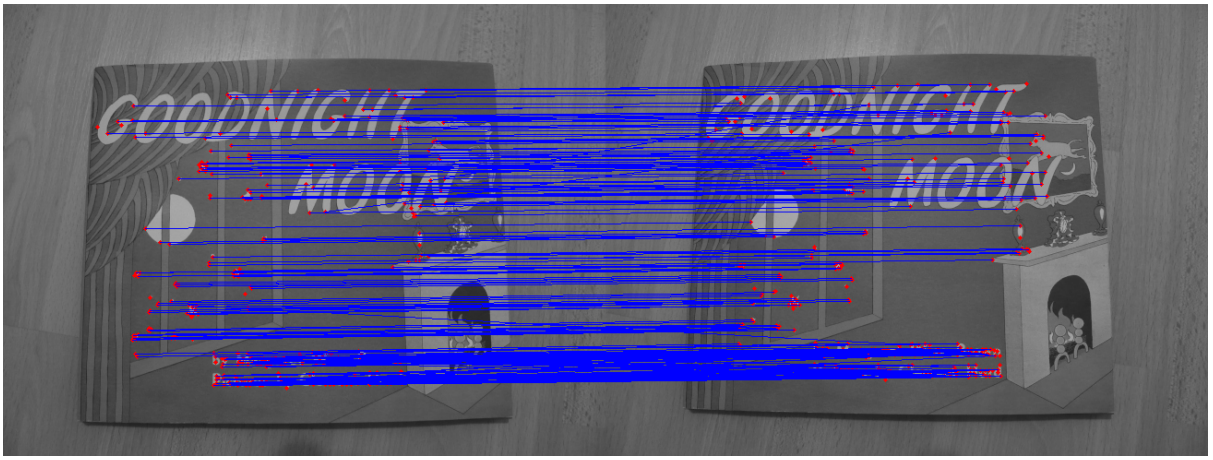
```
col = np.argmin(distances[i])
second = np.partition(distances[i],1)[1]
```

```
if distances[i,col] < ratio_thresh * second:
        matches += [[i,col]]
```

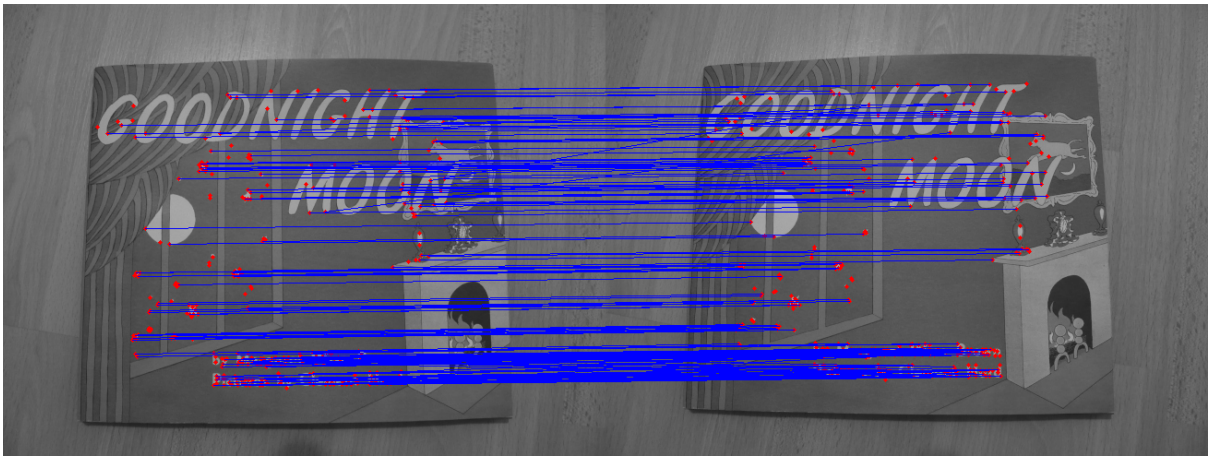
Second nearest distance is also found and the ratio between first and second distances are tested.



One way



Mutual



Ratio test (0.5)

Orhun Görkem

We see that results of one-way matching are more assertive and faulty. The diagonal matches should not appear as both images stand in the same direction vertically.
The number of matches in mutual nearest neighbors matching is fewer, but seems more accurate. Applying the ratio test decreased the number of matches even more, but this may not be necessary as mutual matching seems to be sufficiently accurate. The ratio applied in the ratio test can be tuned and increased to increase the number of matches when necessary. However, increasing the ratio closer to 1 converts the method to one-way matching and we may face inaccurate matches again.