Advanced Image Processing
Assignment #1

Shikhar Vashishth

M.Tech CSA - 13374

Problem 1

Implementation:

Gradient Based: Sobel filter, a gradient-based edge detector, has been used for detecting edges in the image. For implementing this method two sobel operators were created of 3x3 size for calculating gradient along horizontal and vertical directions.

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

These operators were convolved with the image to give gradients along x and y directions as I_x and I_y. Finally, gradient magnitude was calculated as and a threshold was chosen to get the edges

$$G = \sqrt{{I_x}^2 + {I_y}^2}$$

Laplacian of Gaussian For implementing LOG method, directly LOG filter of size (9x9) was computed using the following formula and with sigma value as 1.0. Then the filter was convolved with the image and based on the image a threshold was guessed to get the edges.

$$-\frac{1}{\pi\sigma^4}\left[1 - \frac{(x^2+y^2)}{2\sigma^2}\right] e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Noise For comparing the performance of the gradient-based and LOG algorithm at different level of noise, I have introduced Gaussian noise in the original image with mean zero and variable variance. Level of noise was increased by increasing the value of variance(sigma). A matrix of the size of image filled with normally distributed random numbers was created using 'randn' function provided by OpenCV and was added to the original image.

Selecting Threshold value For selecting an appropriate value of threshold, a number of threshold values were used and the one giving visually the best result was chosen. The following figures shows the outcome on using different thresholds for both the methods. The guesses for threshold values were made based on the range of values For gradient-based method the best output came at threshold value 300 whereas for the LOG method the best result came at around 17.5.

$$\text{Final output} = \text{Image} > \mathit{threshold}$$

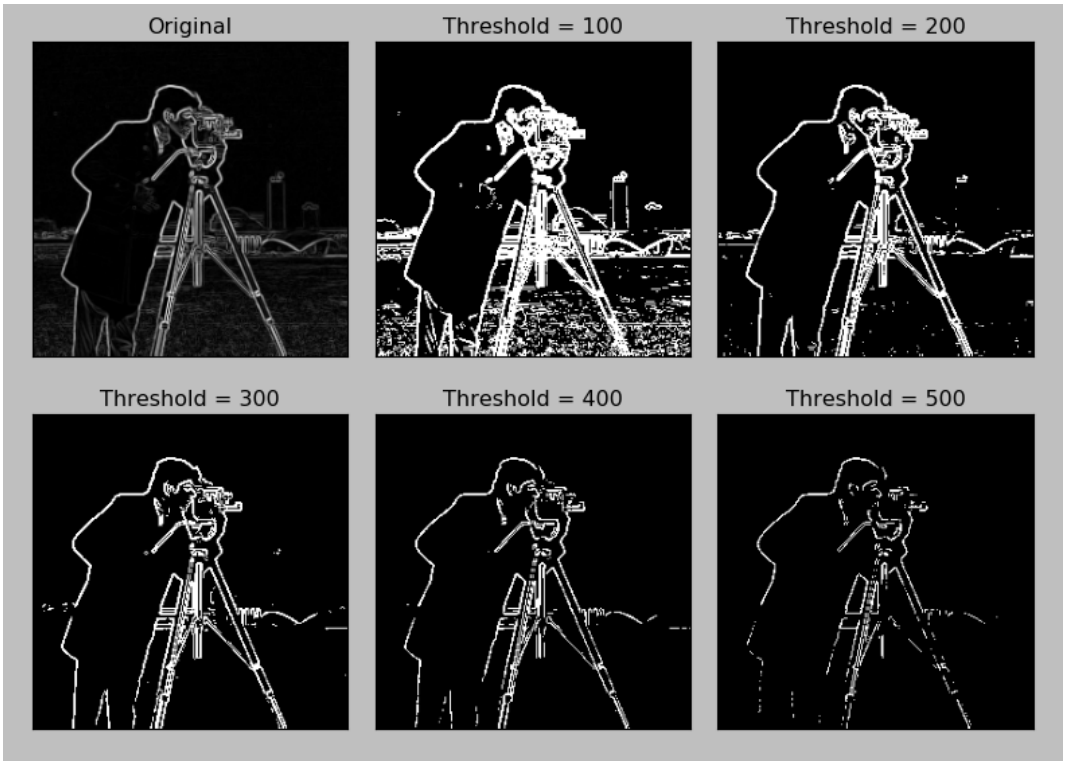


Figure 1: Selecting threshold for gradient based method

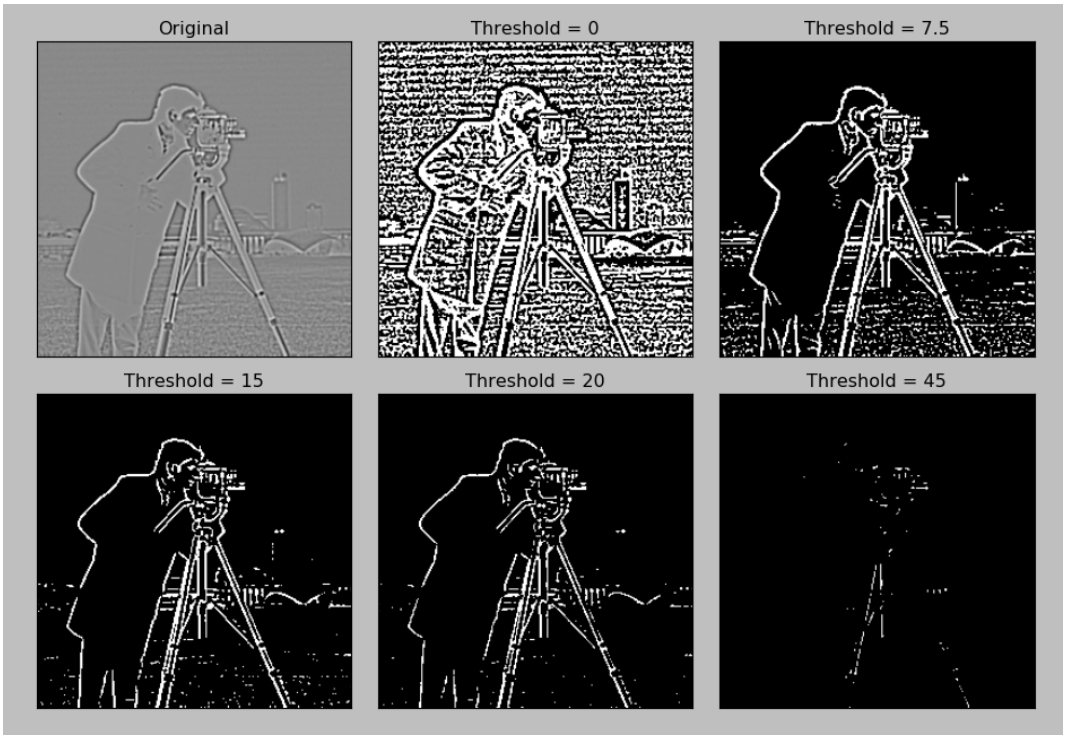


Figure 2: Selecting threshold for LOG method

Results:

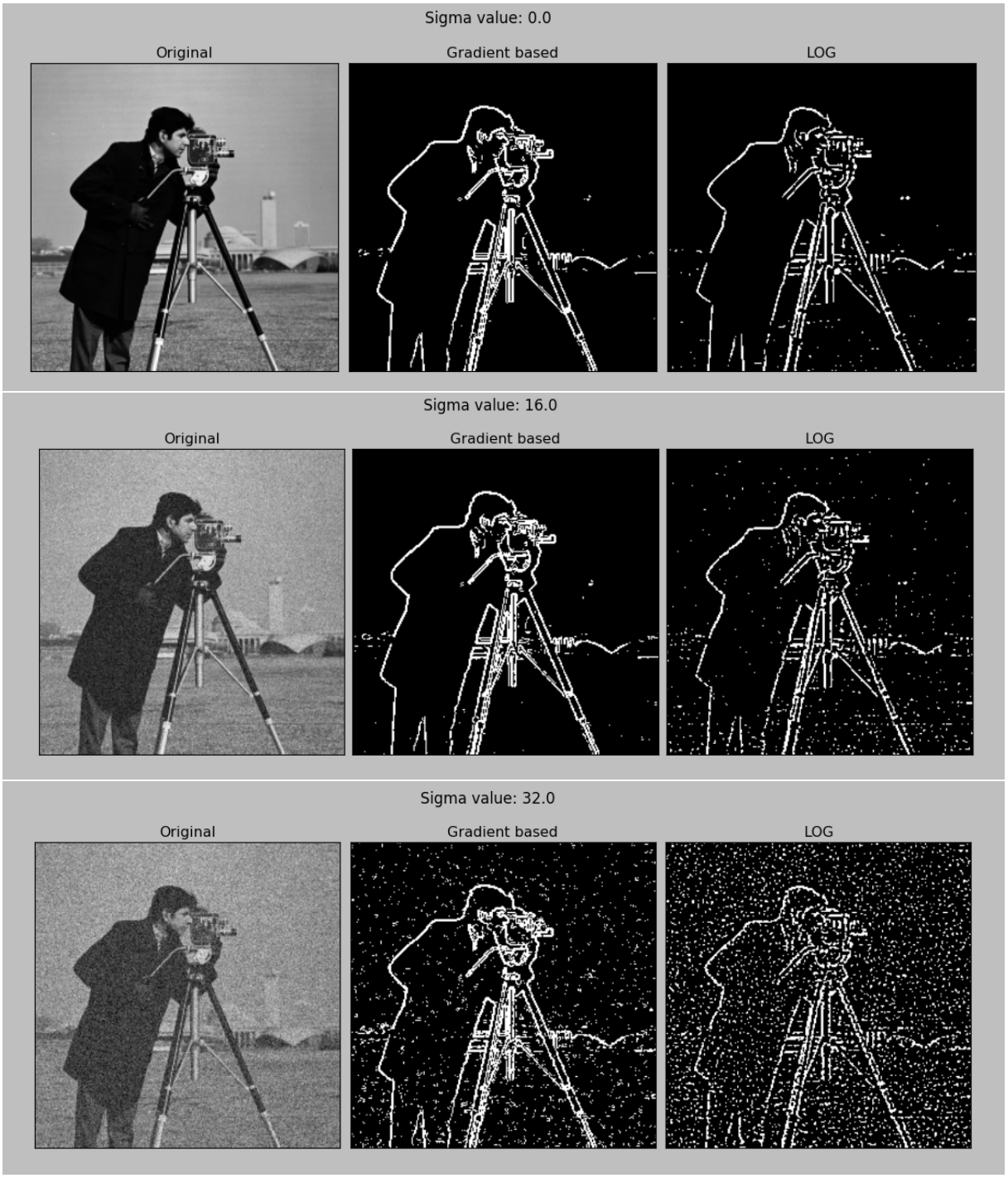
The following figures shows the performance of these two methods on an image with different noise levels. The observations from the results are:

1. LOG method is capable of detecting fine edges as compared to gradient-based methods because it is based on calculating 2nd derivative. But that makes it more susceptible to noise in the image as can be seen from the results. At sigma value 16.0 we can clearly see that LOG results has got corrupted by noise whereas the Sobel result is much more better.

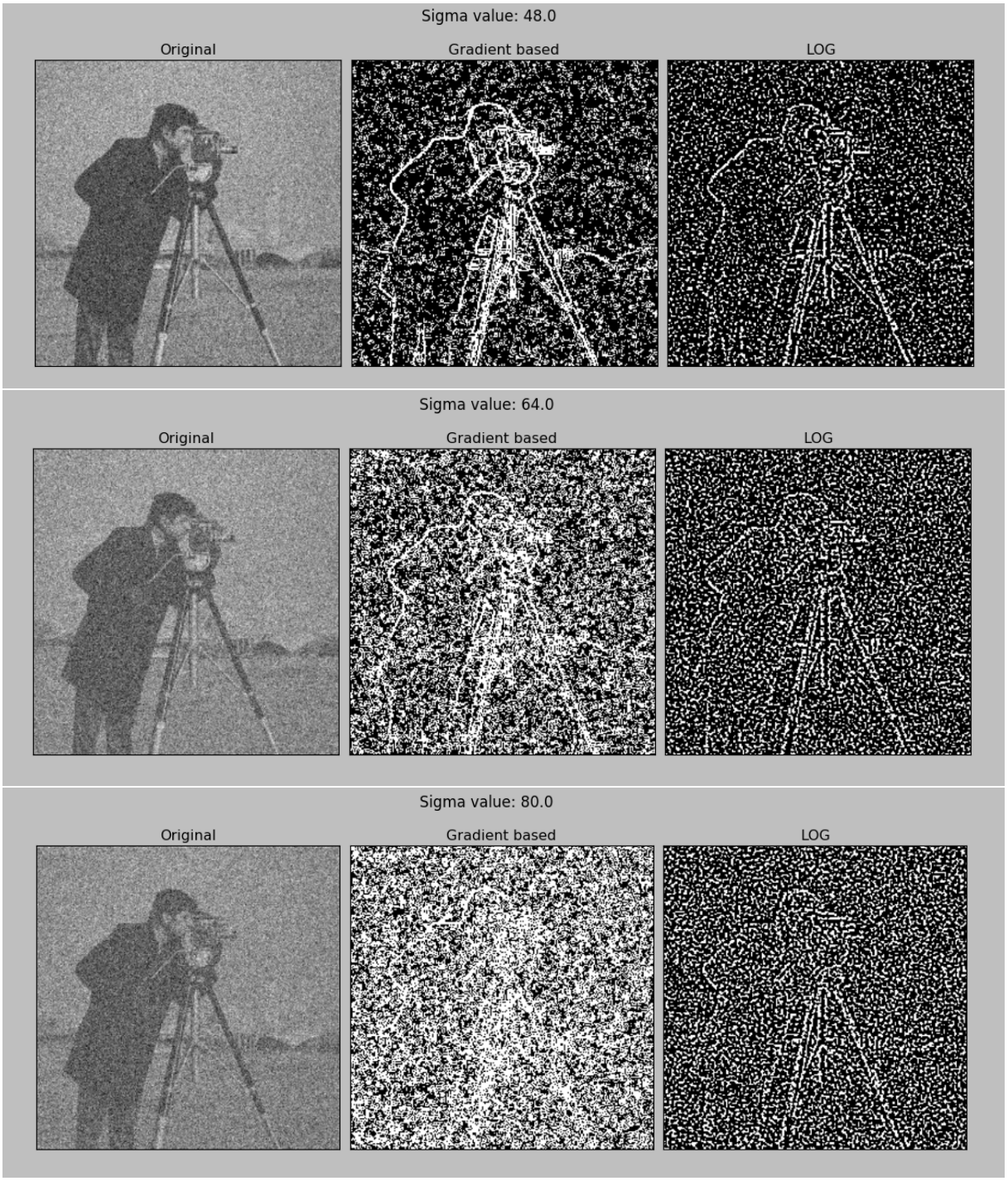
2. On the other hand, the gradient based method although is more robust to noise than LOG but produces thicker edges and misses out finer edges in the image.

3. Since, Sobel method was used for edge detection therefore, the diagonal edges detected are much more thicker than the horizontal and vertical edges because gradient magnitude is calculated as $\sqrt{{I_x}^2 + {I_y}^2}$

Comparing Gradient-based and LOG method for different noise levels



Comparing Gradient-based and LOG method for different noise levels



Problem 2

Implementation

I have used the standard implementation of the SIFT algorithm which comes with OpenCV library (opencv_contrib). For matching two images, I first computed the desciptors of both the images and then for each descriptor I find out its two best matches using k-nearest neighbor algorithm with k=2 so that ratio test as described by D.Lowe in his SIFT paper could be used. A match is accepted if:

$$\frac{SSD(f_1, f_2)}{SSD(f_1, f_2')} \leq 0.8$$

Affine transformation

The original image was transformed using a affine transformation. For the transformation the following matrix was used which scales the image by a factor of 1.2 and rotates it counter-clockwise by 30°.

$$M = \begin{bmatrix} 1.2 * \cos 30° & \sin 30° & 0 \\ \sin 30° & 1.2 * \cos 30° & 0 \end{bmatrix}$$

Noise

The salt-pepper noise was introduced in the image by blending it with a random matrix of the dimensions same as that of the image and with entries uniformly distributed between 0 and 255.

$$Noise = rand(dims, 0, 255)$$

$$Img = \alpha * Img + (1 - \alpha)Noise \ ; \ \alpha = 0.2$$

Changing illumination

For changing the illumination of the image a constant was added to it and the pixel intensities which exceeded 255 were set to 255.

Results:

The figure below shows the detected keypoints in the original and the transformed images using SIFT algorithm. The circles for the keypoints are drawn according to their size and along with their orientation. The number in the bracket shows the number of the keypoints in the image.

Observations:

1. The number of keypoints in the image increased as we introduced noise in the image because noise creates finer details which get detected as keypoints by the algorithm
2. Affine and occlusion transform reduced the number of keypoints because the part of the image got removed because of scaling, rotation and cropping.
3. The flipped image has almost the same number of descriptors as the original image but the orientation of all descriptors has also got flipped.
4. Image with changed illumination has less number of keypoints because some of details of image got lost because of the saturation of pixel intensity values.

Image index	Transf Type	# of Descriptors	Change in Descriptors
(0,0)	Original	145	0%
(0,1)	Noise	344	+137%
(1,0)	Affine & Occlusion	141	-2.75%
(1,1)	Noise & Affine	515	+255.17%
(2,0)	Illumination	65	-55.17%
(2,1)	Flip	144	-0.68%

Figure 3: Showing Keypoints with their size and orientation

Image index	Transf Type	# of Descriptors	Change in Descriptors
(0,0)	Original	206	0%
(0,1)	Noise	292	+41.74%
(0,2)	Affine & Occlusion	143	-30.58%
(1,0)	Noise & Affine	289	+40.29%
(1,1)	Illumination	65	-68.44%
(2,2)	Flip	213	+3.34%

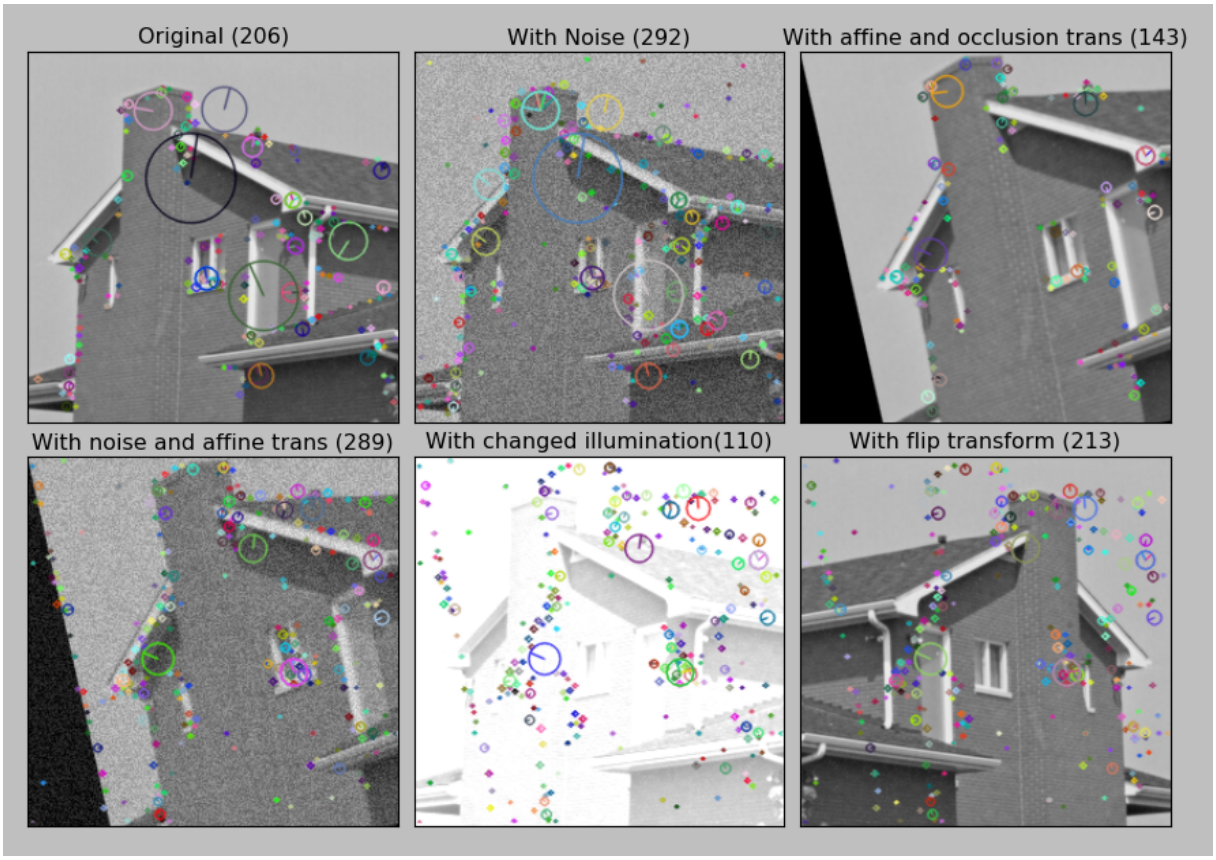


Figure 4: Showing Keypoints with their size and orientation

Matching The matching was done using k-nearest neighbor approach. The observations are as follows:

1. The first result, verifies the scale and rotational invariant nature of the SIFT algorithm which is the most important advantage with SIFT.
2. The algorithm performs well even in the presence of noise
3. The algorithm gave around 25% match when illumination of the image is changed. It shows that it is not invariant to change in illumination.
4. The algorithm performs poorly when comparing with flipped image which shows that although it is invariant to rotation but couldn't perform well when image is flipped.

Trans Type	**# of Matched Descriptors**	**% of matched Descriptors**
Affine & Occlusion	75	52.08%
Noise	79	54.48%
Noise & Affine	60	41.37%
Illumination	35	24.13%
Flip	39	26.89%

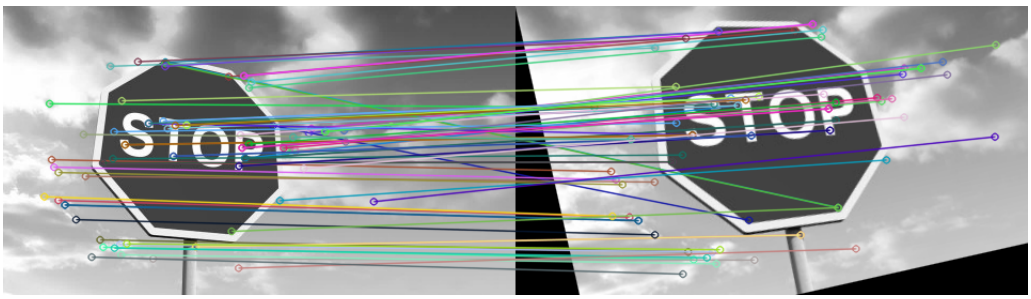


Figure 5: Matching with Affine transformed image

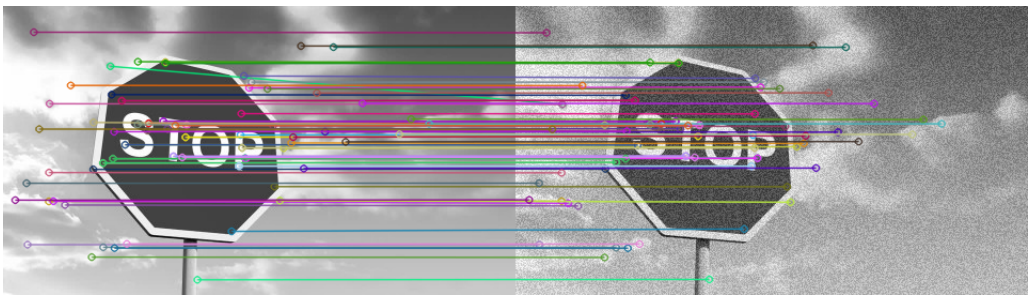


Figure 6: Matching with noisy image

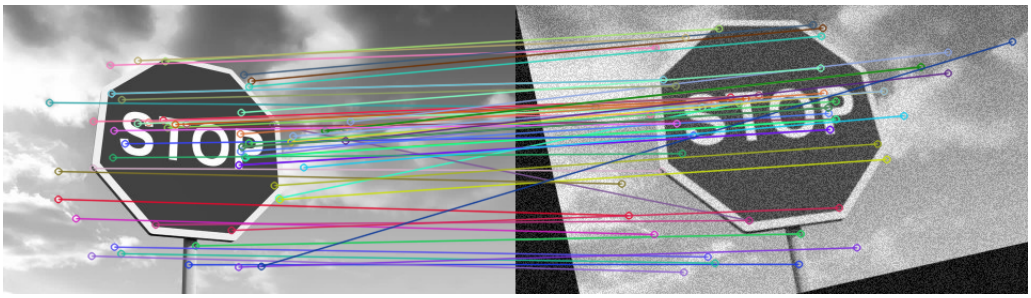


Figure 7: Matching with affine transformed and noisy image

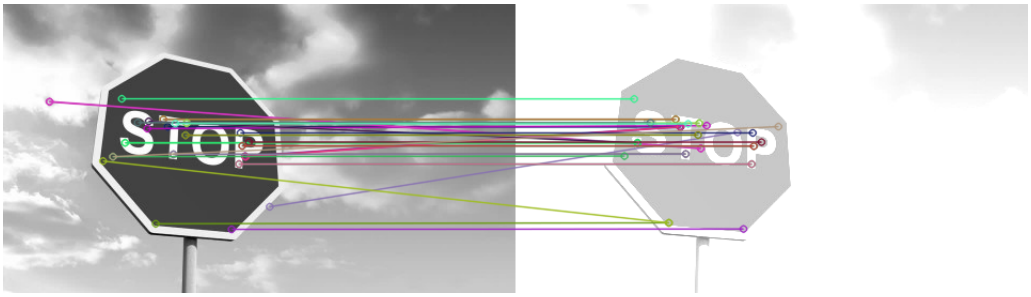


Figure 8: Matching with changed illumination

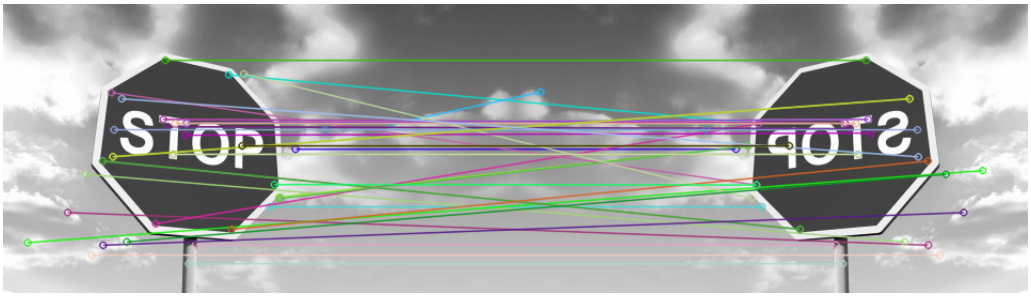


Figure 9: Matching with flipped image

Trans Type	**# of Matched Descriptors**	**% of matched Descriptors**
Affine & Occlusion	90	43.68%
Noise	95	46.11%
Noise & Affine	76	36.89%
Illumination	54	26.21%
Flip	52	25.24%

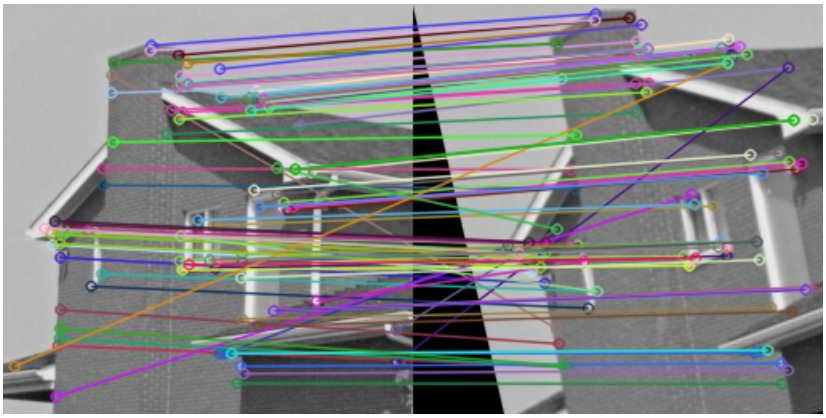


Figure 10: Matching with Affine transformed image

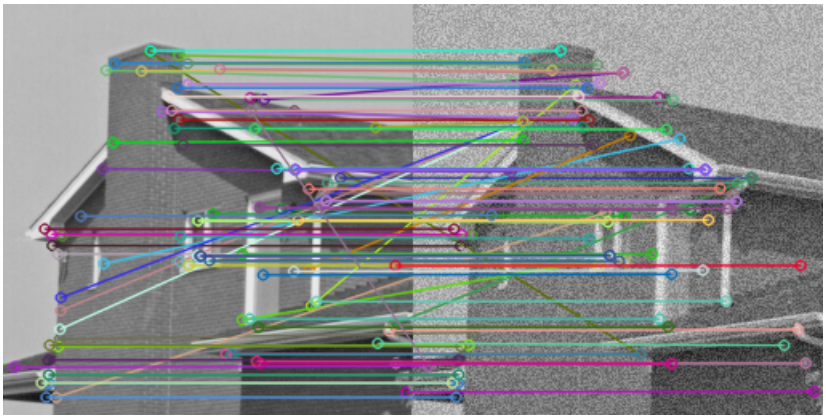


Figure 11: Matching with noisy image

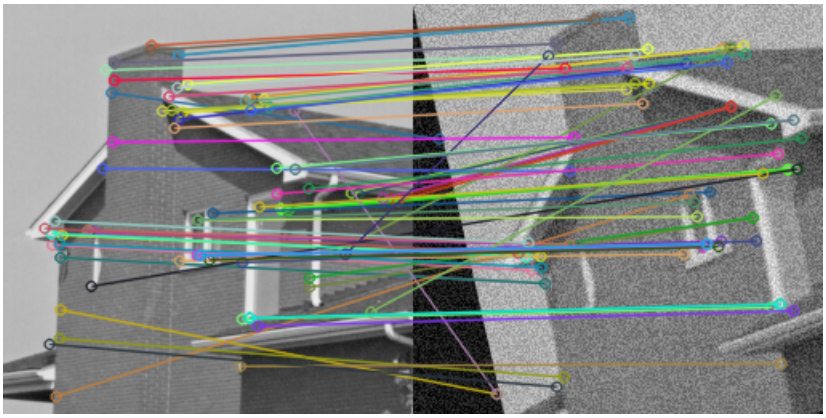


Figure 12: Matching with affine transformed and noisy image

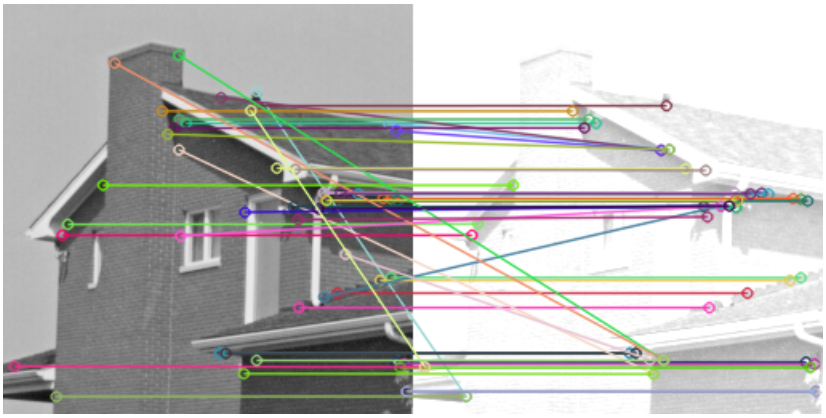


Figure 13: Matching with changed illumination

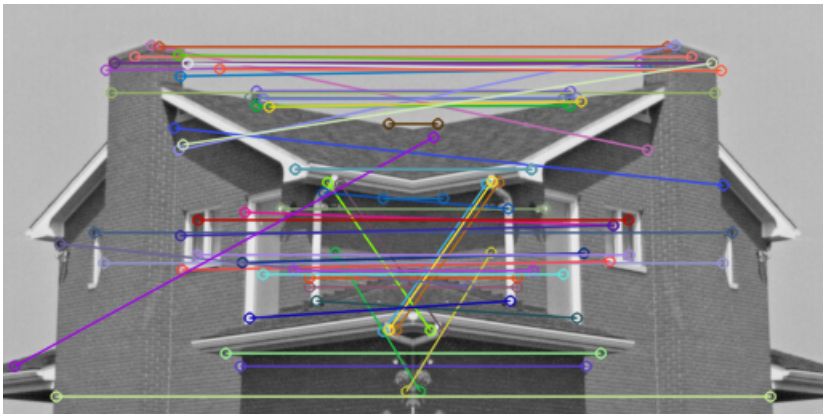


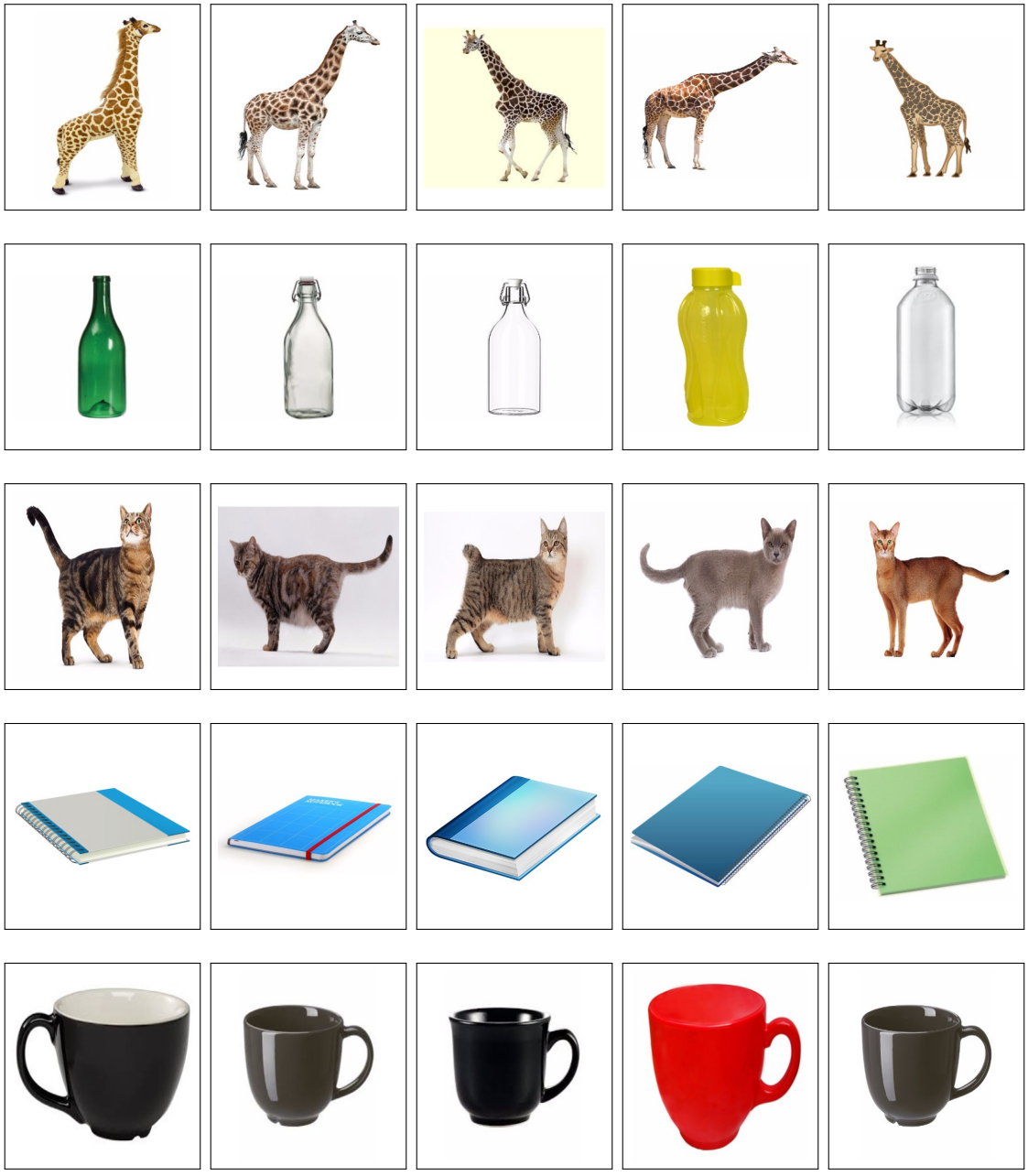
Figure 14: Matching with flipped image

Problem 3

Implementation:

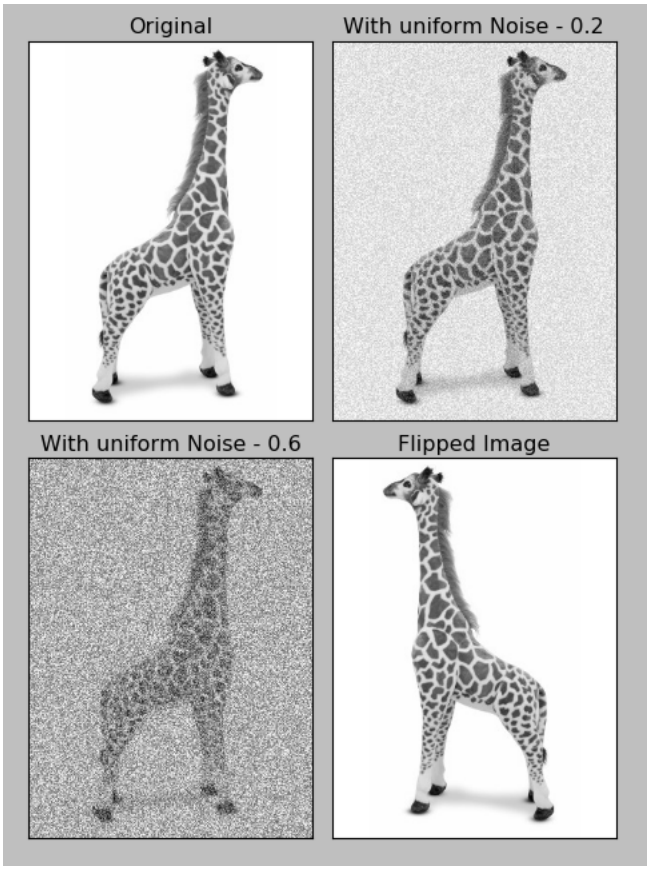
For demonstrating object recognition using SIFT algorithm, I took five images from five different categories namely, giraffe, cup, book, cat, and bottle as shown here:

Images used for Object detection



Model Transformation From all the images the first image from each category was selected as the model image. From each model image, three more images were generated by adding different degrees of noise and flipping the image. As shown here:

Transformations of a model image



Training classifier For training the classifier, descriptors of all the model images and the derieved images were computed using SIFT algorithm and were stored. For predicting category of a new image its descriptors were calculated and were matched with the stored descriptors using k-nearest neighbor approach. Then based on the number of matches the category of the image is predicted.

Results:

Using the technique mentioned above, the classifier was trained and was used for predicting the category of all the 25 images. The following table shows the performance of the classfier:

Image index	Actual Category	Predicted Category	Conclusion
(0,0)	Giraffe	Giraffe	Correct
(0,1)	Giraffe	Cup	Incorrect
(0,2)	Giraffe	Cat	Incorrect
(0,3)	Giraffe	Cat	Incorrect
(0,4)	Giraffe	Cup	Incorrect
(1,0)	Cup	Cup	Correct
(1,1)	Cup	Cup	Correct
(1,2)	Cup	Cup	Correct
(1,3)	Cup	Cup	Correct
(1,4)	Cup	Cup	Correct
(2,0)	Bottle	Bottle	Correct
(2,1)	Bottle	Cup	Incorrect
(2,2)	Bottle	Book	Incorrect
(2,3)	Bottle	Cup	Incorrect
(2,4)	Bottle	Bottle	Correct
(3,0)	Cat	Cat	Correct
(3,1)	Cat	Cat	Correct
(3,2)	Cat	Cat	Correct
(3,3)	Cat	Bottle	Incorrect
(3,4)	Cat	Cat	Correct
(4,0)	Book	Book	Correct
(4,1)	Book	Book	Correct
(4,2)	Book	Book	Correct
(4,3)	Book	Cup	Incorrect
(4,4)	Book	Giraffe	Incorrect

$$\textbf{Average accuracy} = \frac{\text{Total correct predictions}}{\text{Total number of images}} * 100 = \frac{15}{25} * 100 = 60\%$$

The above result shows the efficiency of SIFT algorithm for finding the right generic descriptors of any given image which can be used to recognize new images of the same type.

Poor performance on complex inputs In above setup all images have white background and the shapes of the images are considerably different but if we take more complex or similar looking images then the algorithm performs poorly. In the given categories if we replace

bottles with cycle which are much more complicated images then accuracy falls to 36%. Moreover, if intead of taking images with white backgroud we take images with background then also accuracy falls considerably. So, although the algorithm is efficient but is not the best choice in all situations.