Robot Vision [06-25024]
Summative Assignment 1

Name: Kai Meller
Username: klm222
Study program: MSc Artificial Intelligence and Machine Learning

March 17, 2023

Submission deadline:	**17:00pm (BST), Friday, 17 March 2023**
Instructor:	Dr. Hyung Jin Chang
	Dr. Jianbo Jiao
Total marks:	100
Contribution to overall module mark:	25%
Submission Method:	This assignment must be submitted through Canvas.

Part 1

Question 1.1 I loaded Malards.jpg into Matlab as a 600x629x3 matrix. I then isolated the red channel as a 600x629 matrix. I loaded and applied both Roberts kernels to this image. I calculated the L2 norm of these two results as a third image. I then thresholded the absolute value of these three images to only show edges with value 50 or higher, as shown below :

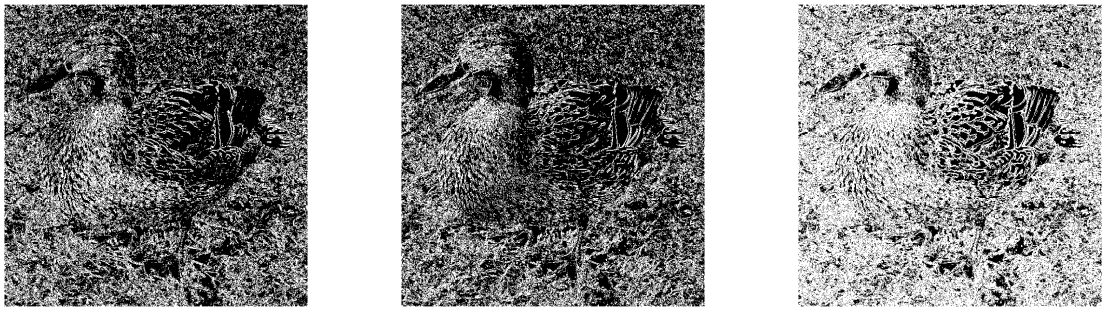

Figure 1: Thresholded images : Roberts A (left), Roberts B (middle), Euclidean Norm of Roberts A and Roberts B (right).

Question 1.2 I loaded the given filter kernels as matrices. I then grayscaled Malards.jpg, and convolved it with the filter kernels A and B :

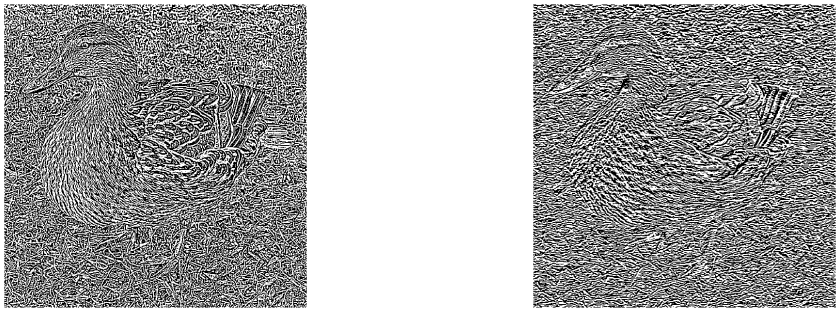

Figure 2: Malards.jpg grayscaled and convolved with A (left) and B (right).

A is an isotropic kernel as the values only depend on the distance from the center, whereas B is anisotropic as the value depends on the direction from the center. For these reasons, A is unable to find edges, but B is able to. Specifically, B could be used to find horizontal edges in images because the gradient at the center of the kernel is vertical. Additionally, as the signs of the vertical layers are $+--++-$, kernel B would be best suited at finding points surrounded by three parallel one-pixel-thick horizontal edges.

Question 1.3 Earth.jpg is loaded and grayscaled. The difference of Gaussians for $size = 15 \times 15, \mu_1 = 0, \sigma_1 = 1, \mu_2 = 0, \sigma_2 = 2\sqrt{2}$, is calculated using my *DiffGaus* function. The kernel is then convolved with the grayscaled image. Since the convolved image contains negative values, I map every pixel onto the range $[0, 1]$ with my custom utility function *zero_one*, which retains relative brightnesses across the image.

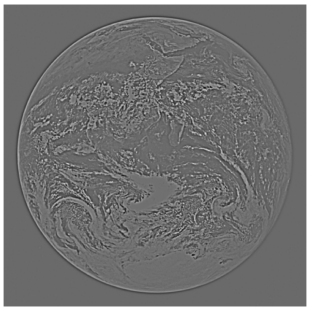

Figure 3: Earth.jpg convolved with the approximate Laplacian of Gaussian kernel.

DiffGaus takes in the size of the desired filter and the parameters of the two Gaussians, and returns the difference of the two Gaussian kernels as a DoG kernel. Assuming each kernel represents the entire Gaussian distribution, this is equivalent to applying each Gaussian kernel to the image separately, and then calculating the difference between the two images. Although this is never the case as Gaussian distributions are infinite, an appropriate filter size can be chosen such that the approximation represents most of the data.

Question 1.4 Again, the image is loaded and grayscaled. Parameters are used to define the range of standard deviations desired, as well as the size of the filter. Empirically, I chose to create 8 DoGs, requiring 9 Gaussian filters with standard deviations ranging from 0 (original image) to $2^4 = 16$.

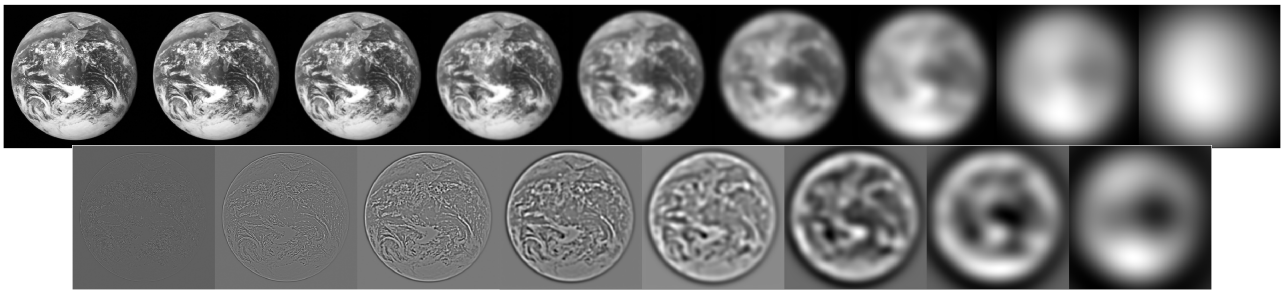

Figure 4: Gaussians applied to Earth.jpg with varying standard deviations (top), and the differences between these convolved images (bottom).

I created another custom utility function called *pad_resize* which takes in an image, the scale to resize to, and the size of the filter to be applied, and returns the image, having been scaled and then padded enough to make the convolution possible without loss of scale.

I use a for loop to generate each convolved image, and store them in a cell array. I then created 2 montages to display the results of the Gaussians, and their respective DoGs.

Part 2

Question 2.1 My *convolve* function takes in the image I as a matrix of doubles and the filter F, and pads the image using my custom utility *mypad*. *mypad* takes in the image and the filter, and pads I with 0s such that the filter center can be placed on each corner of I in the padded image (odd size filter), or in the first midpoint from each corner of I in the padded image (even size filter). I use 4 nested for loops to apply the filter centered at every pixel in the original image, and store the weighted sum in the output matrix $I2$.

Figure 5: Old_house.jpg (left) and Butterfly.jpg (right) grayscaled and convolved with a 15×15 Gaussian filter with $\sigma = 5$.

Question 2.2 I create a mean filter using the code $mean_filter = ones(15)/225$ which creates a 15×15 matrix of 1s, and then normalises to make the magnitude of the filter 1.

Figure 6: Old_house.jpg (left) and Butterfly.jpg (right) grayscaled and convolved with a 15×15 Mean filter.

Question 2.3 Much alike *convolve* function, my *simpleMedian* uses 4 nested for loops to cycle through each combination of pixels in I and in the filter area. It creates a temporary list of every value in the filter area, and sets the value in the output image $I2$ to be the median of this list. I chose to reduce the size of the output image by 1 pixel in each dimension that the filter is even, as opposed to increasing the image size by 1, or maintaining the same image size with an a-symmetrical output image. I made this decision as, for the median filter, any pixel on the border would be set to the median value in the area, which would be the padding value of 0. This would create an artificial black border on each side of the image if I were to increase the size of the image by 1, or on two edges if I were to keep the image the same size.

Figure 7: Old_house.jpg (left) and Butterfly.jpg (right) grayscaled and with a 12×12 Median filter applied.

Question 2.4 It looks like the median filter has done the best at removing the salt and pepper noise while maintaining sharpness in the image. This is because the noise has set some pixels to be extrema or minima, which makes very little difference to the median function. The median filter has, however, also removed most fine details such as the antennae of the butterfly because they are not the majority in the 12×12 grid against the background. The Gaussian blur and the mean filter, as one would expect, have blurred the image considerably, removing the noise, but also losing sharpness. The Gaussian filter is better than the mean filter as values nearer the center pixel have a stronger affect on the output value, and so more fine details are retained.

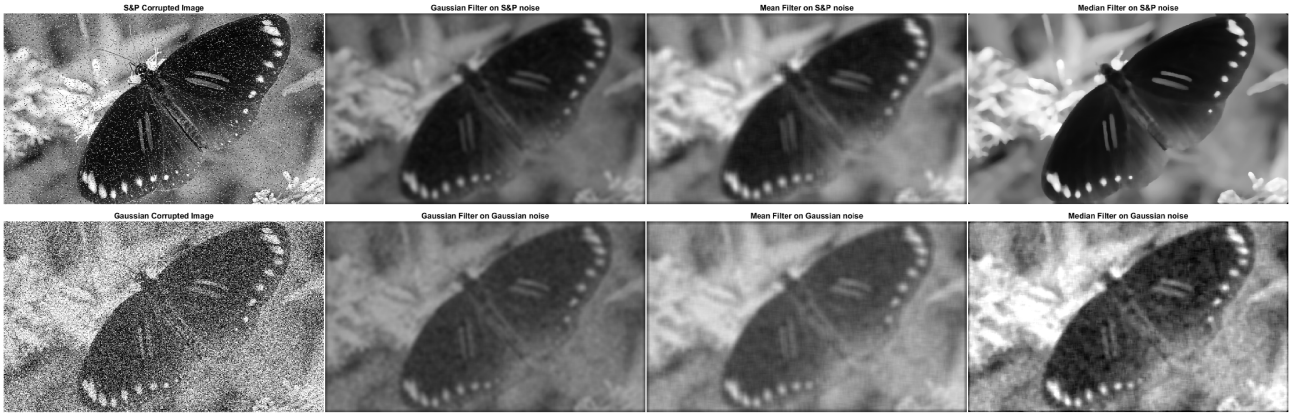

Figure 8: Salt and Pepper noise density = 0.09 (top), Gaussian Noise $\mu = 0.1, \sigma^2 = 0.2$ (bottom). 15×15 Gaussian filter with $\mu = 0, \sigma = 5$ (center-left), 15×15 mean filter (center-right), 12×12 median filter (right).

As for the Gaussian noise, the Gaussian filter is again better at removing the graininess than the mean filter, with the mean filter introducing vertical and horizontal artifacts as it does not respect direction. The median filter fails on Gaussian noise as the median of each area is affected substantially by the introduction of additional variance.

Part 3

Question 3.1 I started by applying each built-in edge detector to Dice.jpg. By comparing the results side-by-side, I decided to use the default Canny edge detector as it appeared to have the best edge map, BW. I used the built-in *imfindcircles* function to find circles in BW with diameter 11 ± 5 which gives a radius between 3 and 8. I then used these circle centers to remove all edges from BW within a 13×13 square around the center. I applied the Hough transform to the image, and found the peaks with the *houghpeaks* function. I used *houghlines* to get the lines from the Hough peaks to be displayed. I used *viscircles* to display the circles that were found.

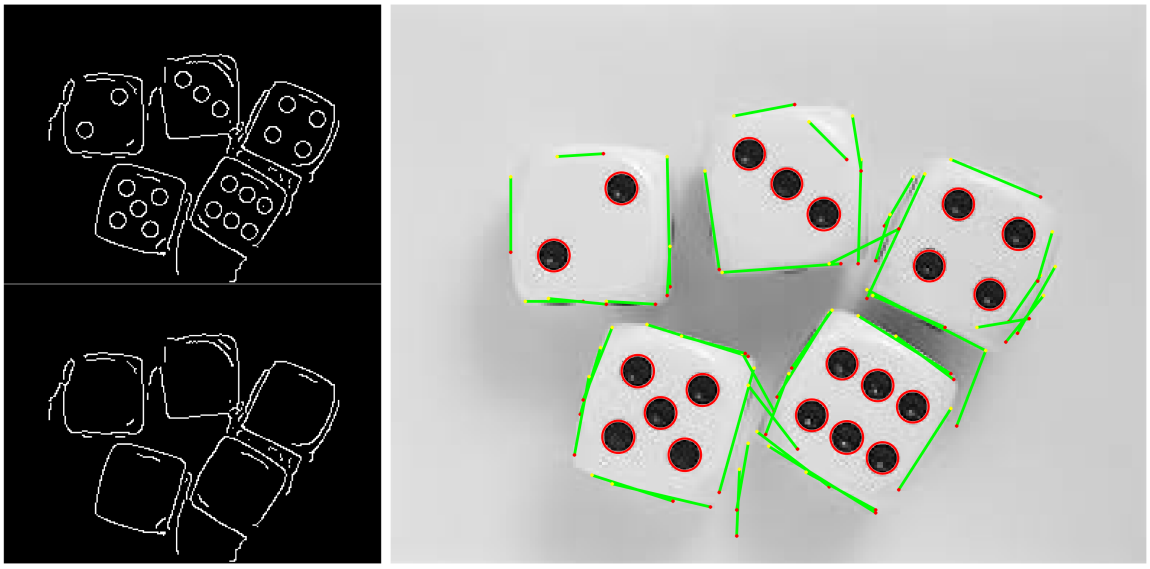

Figure 9: Result of Canny edge detector (top-left). Removing circles from Canny edge map (bottom-left). Circles (in red) and edges (in green) detected from and overlaid onto Dice.jpg (right).

The values of the parameters were decided largely by experimentation and comparing results.

In *imfindcircles*, and despite knowing the radius of each circle was between 5 and 6 pixels, I discovered that a range of 3 to 8 pixels with a sensitivity of 0.9 found every circle in the image with no erroneous circles, while setting the radius range to between 5 and 6 lead to fewer or no circles discovered.

I used the *NHoodsize* parameter for *houghpeaks* to limit the closeness of peaks discovered from the Hough transform. This parameter "... is the neighborhood around each peak that is set to zero after the peak is identified". I found this parameter immensely useful in limiting the number of line segments in a small area. The default value was set to be $size(H)/50$, rounded up to the nearest positive odd integer, which in this case was $[13, 5]$. I adopted the value $size(H)/20$, which is $[33, 9]$. As this parameter was very restrictive, I allowed up to 100 peaks to be found, with a threshold of $0.3\times$ the highest H value.

In *houghlines*, I used the parameters *FillGap* and *MinLength* to tweak the generated lines. Purely through experimentation, I found $FillGap = 5$ and $MinLength = 15$ worked well.

Sources : a large portion of this code was adapted from the code given in Lab 3 of this module.

Part 4

Question 4.1 For each feature detection algorithm, a subplot is created showing the 100 strongest features given by the algorithm.

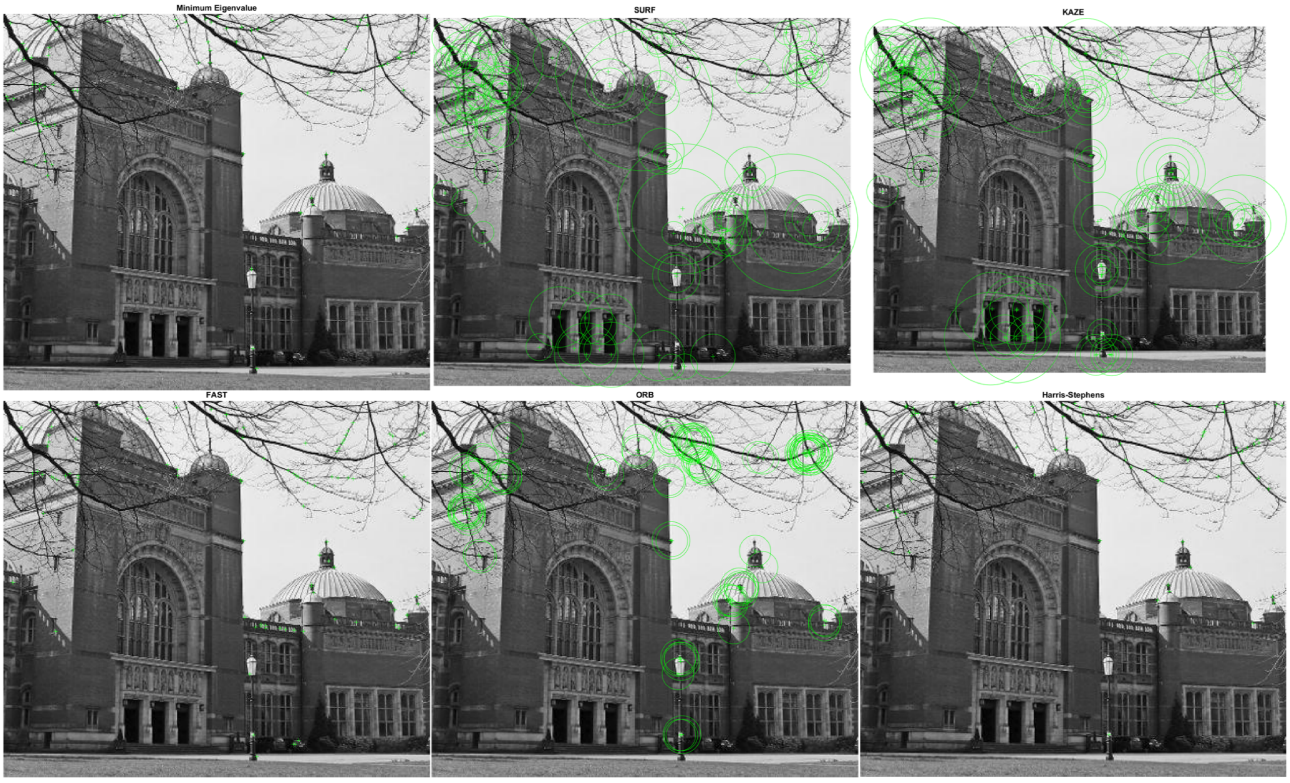

Figure 10: Uob_University_Square.jpg with Minimum Eigenvalue (top left), SURF, (top center), KAZE (top right), FAST (bottom left), ORB (bottom center), and Harris-Stephens (bottom right) feature detection algorithms applied.