

**De La Salle University- Manila**

**Gokongwei College of Engineering**

Lab Activity Number : 3

Lab Activity Title : Smoothing and Blurring

Date Performed : 10/01/2024

Date Submitted : 10/22/2024

Instructor : Jay Robert B. del Rosario, PhD

Name of Student : Collado, Andrei Vincent V.

Student ID : 12149616

Subject / Section : LBYCPF3/ EQ1

Remarks:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Objectives:**

1. To understand the fundamental concepts of blurring techniques and apply different blurring methods such as averaging, Gaussian, median, and bilateral filtering
2. To learn the effect of kernel size on blurring intensity by experiminenting with various kernel sizes for each blurring technique to under
3. .

**Part I. Simulations**

**Example 1: convolutional.py**

**Codes:**

| import cv2  import numpy as np  img = cv2.imread('assets/wallpaper1.png')  rows, cols = img.shape[:2]  kernel\_identity = np.array([[0,0,0], [0,1,0], [0,0,0]])  kernel\_3x3 = np.ones((3,3), np.float32) / 9.0  kernel\_5x5 = np.ones((5,5), np.float32) / 25.0  cv2.imshow('Original', img)  output = cv2.filter2D(img, -1, kernel\_identity)  cv2.imshow('Identity filter', output)  output = cv2.filter2D(img, -1, kernel\_3x3)  cv2.imshow('3x3 filter', output)  output = cv2.filter2D(img, -1, kernel\_5x5)  cv2.imshow('5x5 filter', output)  cv2.waitKey() |
| --- |

**Output Screenshot:**

|  |
| --- |

**Example 2: blurring.py**

**Codes:**

| # USAGE  # python blurring.py --image ../images/trex.png  # python blurring.py --image ../images/beach.png  # Import the necessary packages  import numpy as np  import argparse  import cv2  # Construct the argument parser and parse the arguments  ap = argparse.ArgumentParser()  ap.add\_argument("-i", "--image", required = True,  help = "Path to the image")  args = vars(ap.parse\_args())  # Load the image and show it  image = cv2.imread(args["image"])  cv2.imshow("Original", image)  # Let's apply standard "averaging" blurring first. Average  # blurring (as the name suggests), takes the average of all  # pixels in the surrounding area and replaces the centeral  # element of the output image with the average. Thus, in  # order to have a central element, the area surrounding the  # central must be odd. Here are a few examples with varying  # kernel sizes. Notice how the larger the kernel gets, the  # more blurred the image becomes  blurred = np.hstack([  cv2.blur(image, (3, 3)),  cv2.blur(image, (5, 5)),  cv2.blur(image, (7, 7))])  cv2.imshow("Averaged", blurred)  cv2.waitKey(0)  # We can also apply Gaussian blurring, where the relevant  # parameters are the image we want to blur and the standard  # deviation in the X and Y direction. Again, as the standard  # deviation size increases, the image becomes progressively  # more blurred  blurred = np.hstack([  cv2.GaussianBlur(image, (3, 3), 0),  cv2.GaussianBlur(image, (5, 5), 0),  cv2.GaussianBlur(image, (7, 7), 0)])  cv2.imshow("Gaussian", blurred)  cv2.waitKey(0)  # The cv2.medianBlur function is mainly used for removing  # what is called "salt-and-pepper" noise. Unlike the Average  # method mentioned above, the median method (as the name  # suggests), calculates the median pixel value amongst the  # surrounding area.  blurred = np.hstack([  cv2.medianBlur(image, 3),  cv2.medianBlur(image, 5),  cv2.medianBlur(image, 7)])  cv2.imshow("Median", blurred)  cv2.waitKey(0)  # You may have noticed that blurring can help remove noise,  # but also makes edge less sharp. In order to keep edges  # sharp, we can use bilateral filtering. We need to specify  # the diameter of the neighborhood (as in examples above),  # along with sigma values for color and coordinate space.  # The larger these sigma values, the more pixels will be  # considered within the neighborhood.  blurred = np.hstack([  cv2.bilateralFilter(image, 5, 21, 21),  cv2.bilateralFilter(image, 7, 31, 31),  cv2.bilateralFilter(image, 9, 41, 41)])  cv2.imshow("Bilateral", blurred)  cv2.waitKey(0) |
| --- |

**Output Screenshot:**

|  |
| --- |

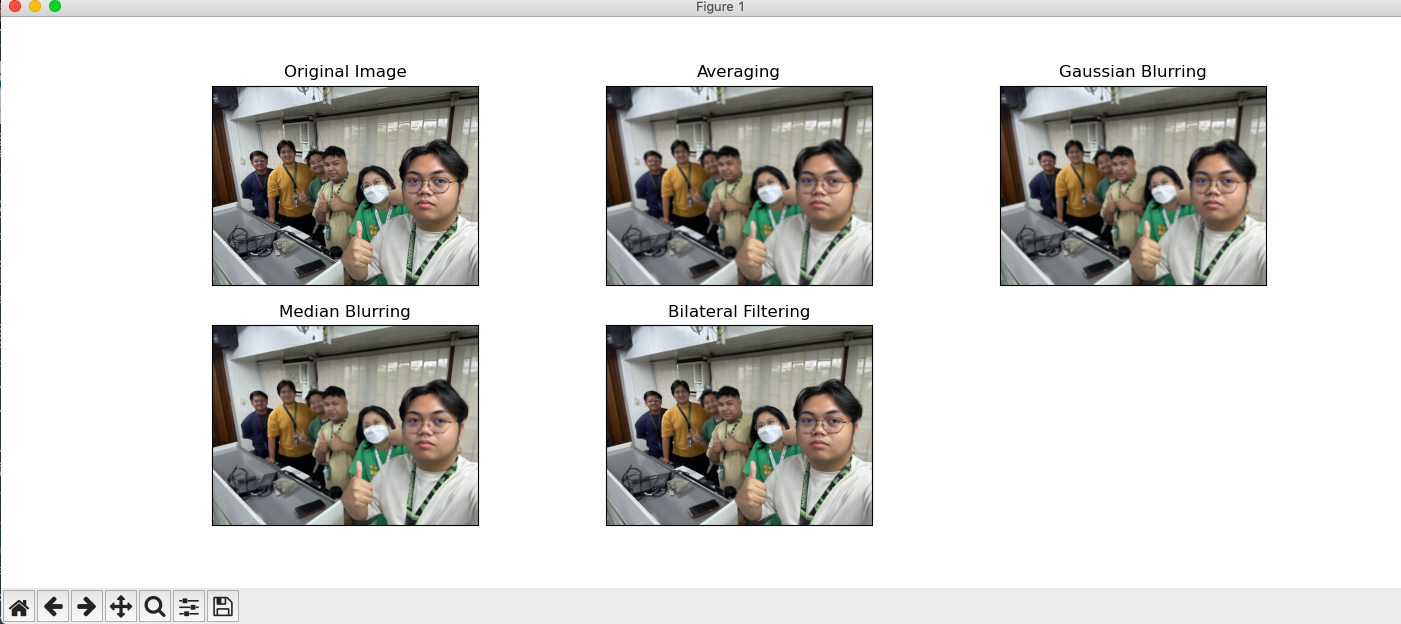
**Part II. Group Activity**

**Task 1:**

**Codes:**

| **import cv2**  **import numpy as np**  **import matplotlib.pyplot as plt**  **image\_path = 'group\_pic.jpg' # Replace with your image path**  **image = cv2.imread(image\_path)**  **# Check if the image is loaded**  **if image is None:**  **print("Error: Could not load image. Check the file path.")**  **else:**  **# Convert image to RGB**  **image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)**  **# Apply different blurring techniques**  **# Averaging blurring**  **blur\_avg = cv2.blur(image\_rgb, (15, 15))**  **# Gaussian Blurring**  **blur\_gaussian = cv2.GaussianBlur(image\_rgb, (15, 15), 10)**  **# Median Blurring**  **blur\_median = cv2.medianBlur(image\_rgb, 15)**  **# Bilateral Filtering**  **blur\_bilateral = cv2.bilateralFilter(image\_rgb, 15, 150, 150)**  **# Display the image outputs**  **titles = ['Original Image', 'Averaging', 'Gaussian Blurring', 'Median Blurring', 'Bilateral Filtering']**  **images = [image\_rgb, blur\_avg, blur\_gaussian, blur\_median, blur\_bilateral]**  **# Create a figure to display the results**  **plt.figure(figsize=(15, 10))**  **for i in range(5):**  **plt.subplot(2, 3, i+1)**  **plt.imshow(images[i])**  **plt.title(titles[i])**  **plt.xticks([]), plt.yticks([])**  **plt.show()** |
| --- |

****

**Output Screenshot:  
  
**

**Discussion/Q and A:**

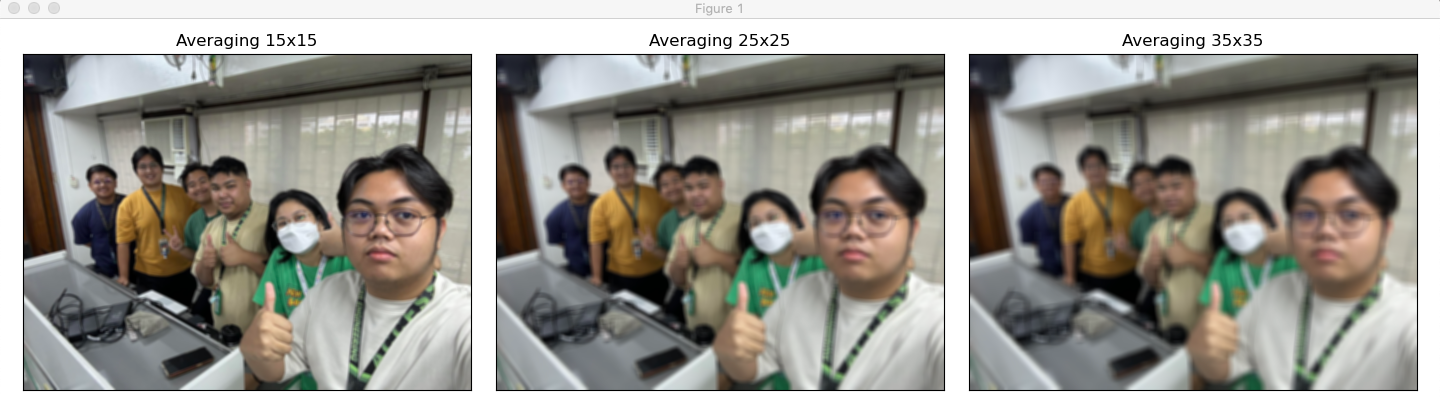
***Question: How do the different blurring techniques (averaging, Gaussian, median, bilateral) alter the appearance of the image, and which technique best preserves the edges while reducing noise?***

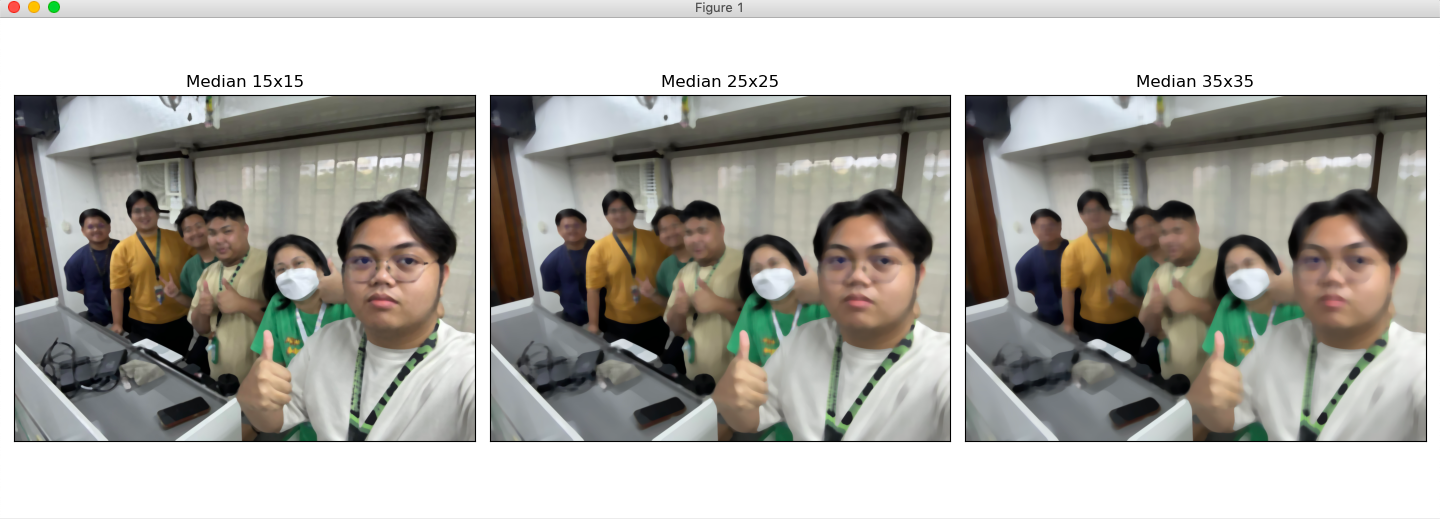
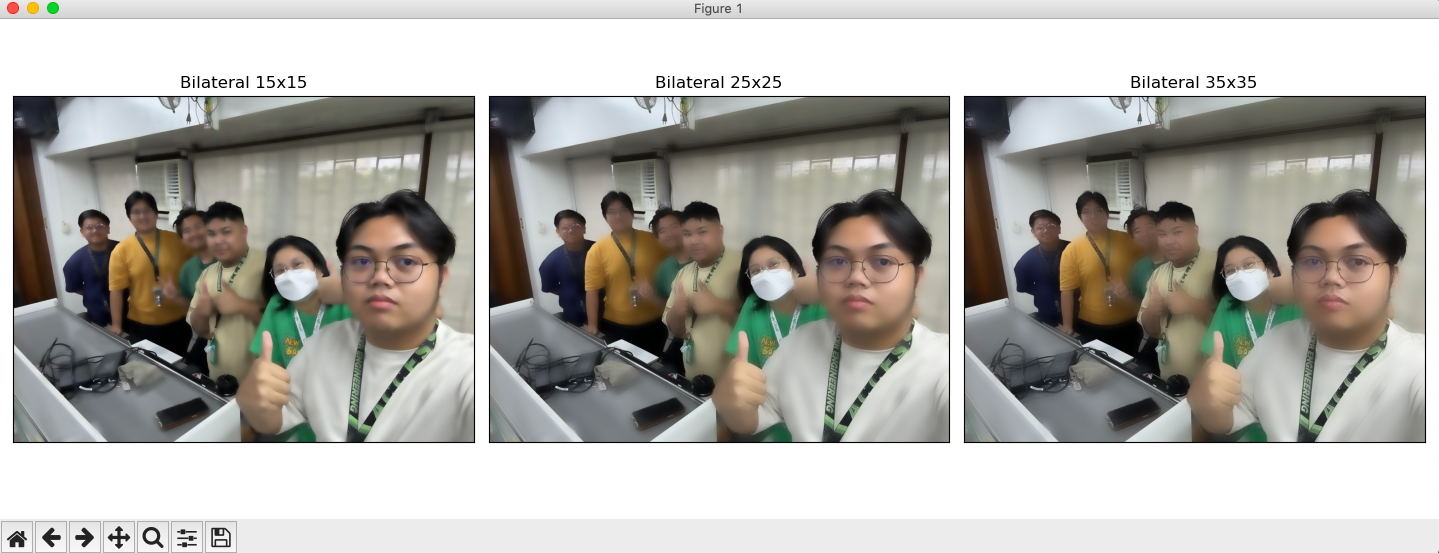
Answer: Blurring techniques influence images differently based on the type of blur applied, particularly affecting noise reduction and edge preservation. For instance, averaging—or box blur—involves calculating the average of the surrounding pixels, resulting in a uniform blur that reduces detail and softens edges. Gaussian blur computes a weighted average where nearby pixels have a greater impact, producing a smoother and more natural blur, though edges remain blurred. Median blur replaces each pixel with the median value from its neighboring pixels, making it highly effective at removing noise—especially salt-and-pepper noise—and it preserves edges much better than averaging or Gaussian blurring methods. Bilateral filtering reduces noise while maintaining edge sharpness by considering both spatial proximity and intensity similarity, making it the most efficient at preserving edges while smoothing out flat areas.

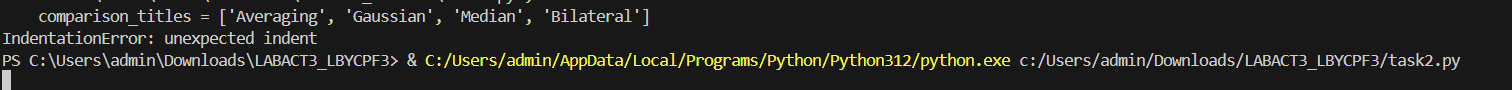
**Task 2:**

**Codes:**

| **import cv2**  **import numpy as np**  **import matplotlib.pyplot as plt**  **# Load the image**  **image\_path = 'group\_pic.jpg' # Replace with your image**  **image = cv2.imread(image\_path)**  **if image is None:**  **print("Error: Could not load image. Check the file path.")**  **else:**  **# Convert image to RGB**  **image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)**  **# Kernel sizes example**  **kernel\_sizes = [15, 25, 35]**  **# Function to display a set of images for filtering technique**  **def display\_filter\_results(filter\_name, images, kernel\_sizes):**  **plt.figure(figsize=(15, 5))**  **for i, img in enumerate(images):**  **plt.subplot(1, 3, i + 1)**  **plt.imshow(img)**  **plt.title(f'{filter\_name} {kernel\_sizes[i]}x{kernel\_sizes[i]}')**  **plt.xticks([]), plt.yticks([])**  **plt.tight\_layout()**  **plt.show()**  **# Blurring using Averaging**  **avg\_blurs = [cv2.blur(image\_rgb, (k, k)) for k in kernel\_sizes]**  **display\_filter\_results('Averaging', avg\_blurs, kernel\_sizes)**  **# Blurring using Gaussian**  **gaussian\_blurs = [cv2.GaussianBlur(image\_rgb, (k, k), 0) for k in kernel\_sizes]**  **display\_filter\_results('Gaussian', gaussian\_blurs, kernel\_sizes)**  **# Blurring using Median**  **median\_blurs = [cv2.medianBlur(image\_rgb, k) for k in kernel\_sizes]**  **display\_filter\_results('Median', median\_blurs, kernel\_sizes)**  **# Blurring using Bilateral Filtering (increasing parameters for stronger effect)**  **bilateral\_blurs = [cv2.bilateralFilter(image\_rgb, k\*2, 150, 150) for k in kernel\_sizes]**  **display\_filter\_results('Bilateral', bilateral\_blurs, kernel\_sizes)**  **# Compare all techniques side by side**  **kernel\_index = 1**  **# Prepare images for side by side comparison**  **comparison\_images = [**  **avg\_blurs[kernel\_index],**  **gaussian\_blurs[kernel\_index],**  **median\_blurs[kernel\_index],**  **bilateral\_blurs[kernel\_index]**  **]**  **comparison\_titles = ['Averaging', 'Gaussian', 'Median', 'Bilateral']**  **# Display side by side comparison**  **plt.figure(figsize=(15, 5))**  **for i, img in enumerate(comparison\_images):**  **plt.subplot(1, 4, i + 1)**  **plt.imshow(img)**  **plt.title(f'{comparison\_titles[i]} 25x25')**  **plt.xticks([]), plt.yticks([])**  **plt.tight\_layout()**  **plt.show()** |
| --- |

**Output Screenshot:  
  
Averaging  
**

**Gaussian  
  
Median  
  
Bilateral  
**

**25x25 comparison  
  
**

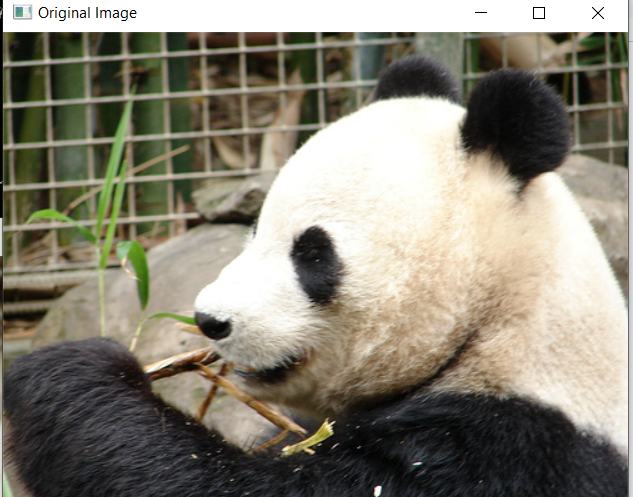
**Discussion/Q and A:  
*Question: How does increasing the kernel size in each blurring technique affect the clarity of fine details and the overall smoothness of the image?  
  
Answer:*** As the kernel size increases, each blurring technique uniquely affects the image's fine detail sharpness and overall smoothness. For averaging, or box blur, larger kernels result in a more pronounced blur, generally causing fine details to be lost and making the image smoother but less clear. Gaussian blur operates similarly; however, larger kernels create stronger smoothing effects while still maintaining a more natural appearance due to its weighted processing method. In the case of median blur, extremely large kernel sizes start to eliminate detailed data. Higher kernel sizes effectively remove more noise but can also distort fine details, particularly when the kernel becomes too large. With bilateral filtering, increasing the kernel size enhances smoothness in uniformly colored areas, but unlike the other techniques, it better preserves edges even as the kernel size grows—though excessively large kernels may still reduce detail.

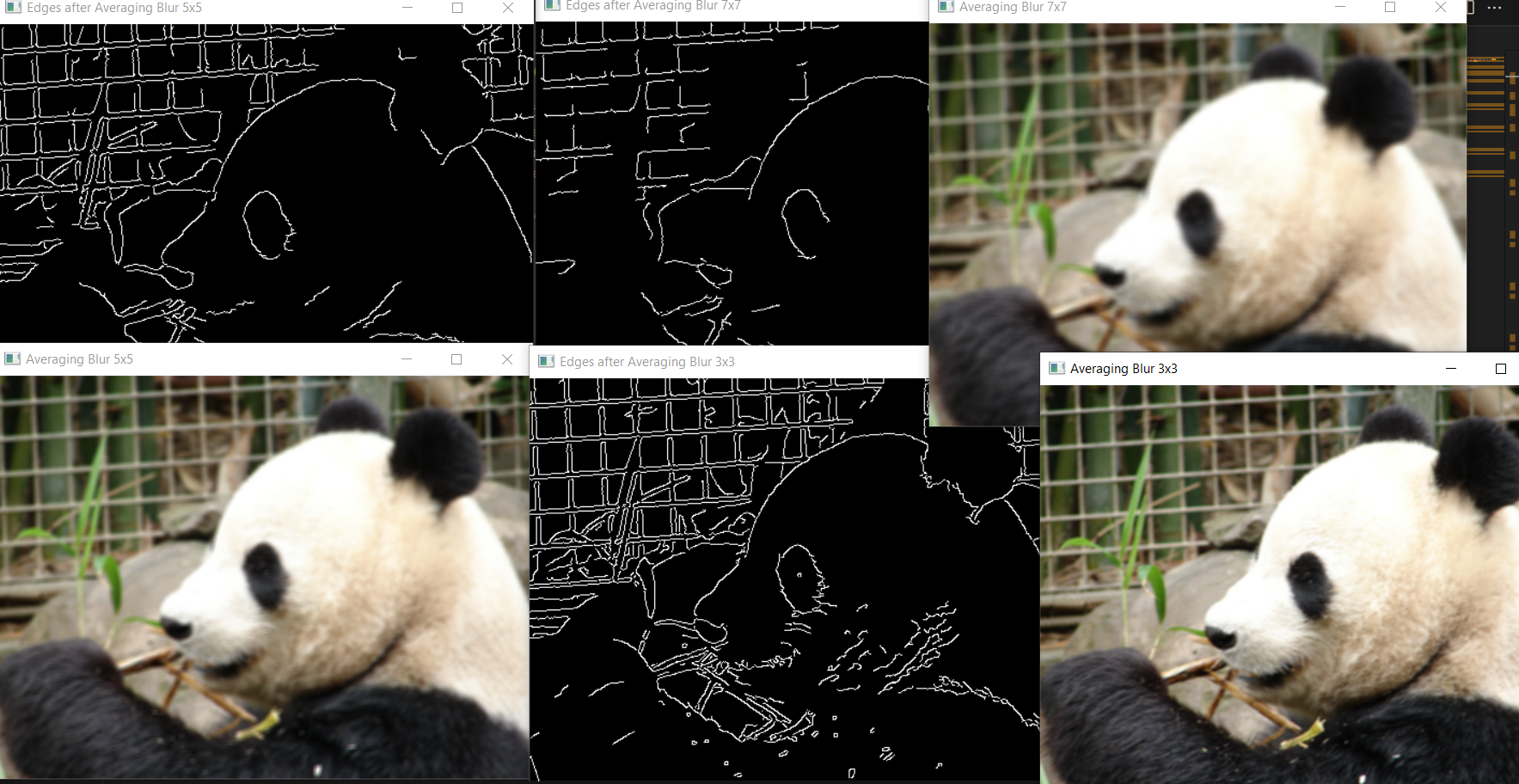
**Part III. Individual Activity**

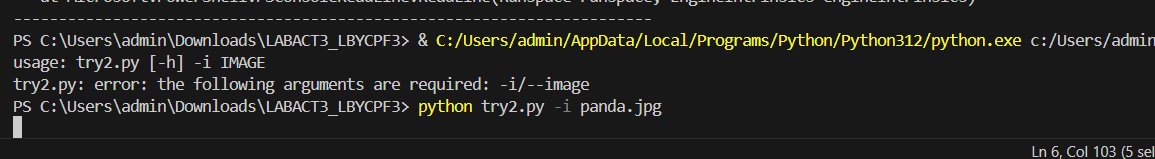
**Codes:**

| **import cv2**  **import argparse**  **import numpy as np**  **# Parse command-line arguments to get the image path**  **parser = argparse.ArgumentParser(description="Apply blurring techniques and edge detection to an image.")**  **parser.add\_argument("-i", "--image", required=True, help="Path to the input image.")**  **args = vars(parser.parse\_args())**  **# Load the image from the specified path**  **image = cv2.imread(args["image"])**  **# Check if the image was loaded successfully**  **if image is None:**  **print("Error: Image not loaded. Check the file path.")**  **exit()**  **# Display the original image**  **cv2.imshow("Original Image", image)**  **# Define a list of kernel sizes to use for blurring**  **kernel\_sizes = [3, 5, 7]**  **# Function to apply Canny edge detection**  **def apply\_canny(image):**  **edges = cv2.Canny(image, 100, 200)**  **return edges**  **# Apply Averaging Blur and perform edge detection**  **for k in kernel\_sizes:**  **# Apply averaging blur with the current kernel size**  **blurred\_avg = cv2.blur(image, (k, k))**  **# Perform edge detection on the blurred image**  **edges\_avg = apply\_canny(blurred\_avg)**  **# Display the blurred image and its edges**  **cv2.imshow(f"Averaging Blur {k}x{k}", blurred\_avg)**  **cv2.imshow(f"Edges after Averaging Blur {k}x{k}", edges\_avg)**  **cv2.waitKey(0)**  **cv2.destroyAllWindows()**  **# Apply Gaussian Blur and perform edge detection**  **for k in kernel\_sizes:**  **# Apply Gaussian blur with the current kernel size**  **blurred\_gaussian = cv2.GaussianBlur(image, (k, k), 0)**  **# Perform edge detection on the blurred image**  **edges\_gaussian = apply\_canny(blurred\_gaussian)**  **# Display the blurred image and its edges**  **cv2.imshow(f"Gaussian Blur {k}x{k}", blurred\_gaussian)**  **cv2.imshow(f"Edges after Gaussian Blur {k}x{k}", edges\_gaussian)**  **cv2.waitKey(0)**  **cv2.destroyAllWindows()**  **# Apply Median Blur and perform edge detection**  **for k in kernel\_sizes:**  **# Apply median blur with the current kernel size**  **blurred\_median = cv2.medianBlur(image, k)**  **# Perform edge detection on the blurred image**  **edges\_median = apply\_canny(blurred\_median)**  **# Display the blurred image and its edges**  **cv2.imshow(f"Median Blur {k}x{k}", blurred\_median)**  **cv2.imshow(f"Edges after Median Blur {k}x{k}", edges\_median)**  **cv2.waitKey(0)**  **cv2.destroyAllWindows()**  **# Apply Bilateral Filter and perform edge detection**  **for k in kernel\_sizes:**  **# Apply bilateral filter with the current diameter size**  **blurred\_bilateral = cv2.bilateralFilter(image, k, 75, 75)**  **# Perform edge detection on the blurred image**  **edges\_bilateral = apply\_canny(blurred\_bilateral)**  **# Display the blurred image and its edges**  **cv2.imshow(f"Bilateral Filter {k}", blurred\_bilateral)**  **cv2.imshow(f"Edges after Bilateral Filter {k}", edges\_bilateral)**  **# Wait until a key is pressed to close all windows**  **cv2.waitKey(0)**  **cv2.destroyAllWindows()** |
| --- |

**Output Screenshot:**

****

****

****

**Discussion/Q and A:**

**How do different blurring techniques and kernel sizes affect the clarity of edge detection results, and which combination provides the best balance between noise reduction and edge preservation?**

Different blurring techniques and varying kernel sizes impact edge detection by balancing noise reduction with edge clarity. Averaging blur smooths out both noise and edges, especially when larger kernels are used, which makes it less effective for edge detection purposes. Gaussian blur offers a more optimal trade-off by slightly preserving edge transitions while still reducing noise; however, using large kernels can eventually diminish edge sharpness. Median blur excels at preserving edges and effectively reduces noise, making it particularly suitable for edge detection with smaller kernels. Bilateral filters provide the best balance between smoothing uniform areas and maintaining sharp edges. Even when larger kernels are applied, bilateral filtering yields clearer images that are ideal for edge detection in noisy images.

**Why might certain blurring methods, such as bilateral filtering, perform better in maintaining edge details compared to others, and how does this impact the overall effectiveness of the edge detection process?**

Bilateral filtering blurs near edges much more effectively than any other filtering technique as it considers the spatial proximity and intensity difference of the pixels in the sampling of adjacent pixels. This brings about selective smoothing wherein the noise in flat regions is deleted without making the transitions at edges fuzzy. Of course, the simpler methods of averaging or Gaussian blur treat all the pixels within the kernel with no such selective consideration. Thus, an image is noise-reduced without hampering edges much, improving edge definitions with more accuracy. As bilateral filtering is effective in preserving detail at edges, it may be that well suited for applications that require higher sharpness in obtaining precise clarity in edge detection even for noisy images.

**Conclusion:**

Different blurring methods-most of which were on the intensity of averaging, Gaussian, median, and bilateral-blurring techniques used in the lab to experiment the kind of influence these have on the quality of the image, especially if they reduce the noise and preserve the edges as much as possible. Experiments carried out revealed that all the blurring methods are different from one another, depending on the kind of image type involved, as well as the kind of outcome desired. Averaging and Gaussian blurring tend to blur the edges and thus reduce their sharpness. Median Blurring. It was very effective for both noise removal and edges preservation. When considering salt-and-pepper noise, it has shown very appreciable results. In bilateral filtering, the maximum amount of noise removal took place along with the sharpness of the edges. This smoothed the relatively flat regions while maintaining the critical details, thus achieving optimal smoothing and feature preservation. Increased kernel size leads to a more pronounced effect of blurring for all methods. A drawback was that finer details are lost, especially for larger kernel sizes. This also preserved edges much better even at larger kernel sizes and was, therefore a method of choice for edge sensitive applications, particularly for pre-processing operations applied before certain edge detection algorithms, like in the case for Canny edge detection.

**Part IV. Peer Evaluation Form**

Review the rubric at the end of this rating sheet. Write the name of each of your group members in a separate column. For each person, indicate the extent to which you agree with the statement on the left, using the percentage points in the rubric (0-100%).

| Evaluation Criteria | Group member:  Name: | Group member:  Name: | Group member:  Name: | Group member:  Name: |
| --- | --- | --- | --- | --- |
| Contribute in a valuable way towards attainment of the objectives | 100 | 100 | 100 | 100 |
| Ability to function on multidisciplinary teams | 100 | 100 | 100 | 100 |
| Problem-solving | 100 | 100 | 100 | 100 |
| Attitude | 100 | 100 | 100 | 100 |
| Focus on the task | 100 | 100 | 100 | 100 |
| Overall Performance Rating | 100 | 100 | 100 | 100 |
| AVERAGE | 100 | 100 | 100 | 100 |

|  |  | |  | |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Criteria | | EXEMPLARY  (94-100) | | SATISFACTORY  (83-93) | | DEVELOPING  (70-82) | BEGINNING  (BELOW 70) |
| Contribute in a valuable way towards attainment of the objectives | | Team member is fully engaged with effective exchange of ideas towards the achievement of the objectives. | | Team member is engaged most of the time. The exchange of ideas towards the achievement of the objectives is effective most of the time. | | Team member is engaged but can be distracted. Ideas towards the achievement of the objectives are exchanged with encouragement. | Team member is only engaged with encouragement or not all team members are engaged. Ideas towards the achievement of the objectives are not exchanged effectively. |
| Ability to function on multidisciplinary teams | | Members working in a multidisciplinary team share respect for each other. All team members of the group feel free to ask questions and contribute. Conflicts are resolved with open dialogue and compromise. | | There is a general atmosphere of respect for members working in a multidisciplinary team. The majority of team members feel free to ask questions and contribute. Team members are generally able to resolve conflicts through open discussion. | | There is a general atmosphere of respect for Members working in a multidisciplinary team, but some team members of the group do not feel free to ask questions and contribute. Team members are generally able to resolve conflicts through open discussion with outside assistance. | Members working in a multidisciplinary team atmosphere is competitive and/or individualistic. Conflicts that arise are not dealt with or cannot be resolved and/or there are no effective group interactions. |
| Problem Solving | | Actively looks for and suggests solutions to problems. | | Refines solutions suggested by others. | | Does not suggest or refine solutions but is willing to try out solutions suggested by others. | Does not try to solve problems or help others solve problems.  Let’s others do the work. |
| Attitude | | Is never publicly critical of the project or the work of others. Always has a positive attitude about the task(s). | | Is rarely publicly critical of the project or the work of others. Often has a positive attitude about the task(s). | | Is occasionally publicly critical of the project or the work of other members of the group. Usually has a positive attitude about the task(s). | Is often publicly critical of the project or the work of other members of the group. Is often negative about the task(s). |
| Focus on the task | | Consistently stays focused on the task and what needs to be done. Very self-directed. | | Focuses on the task and what needs to be done most of the time. Other group members can count on this person. | | Focuses on the task and what needs to be done some of the time. Other group members must sometimes nag, prod, and remind to keep this person on task. | Rarely focuses on the task and what needs to be done. Let’s others do the work. |
| Overall Performance Rating | | Performance consistently  exceeds all group  requirements. | | Performance meets all group  requirements consistently and  sometimes exceeds  requirements. | | Performance meets all group  requirements. | Performance fails to meet  some group requirements. |