Assignment #3

6434480323 Ratchanon Panmas

Python notebook source code with syntax highlighter is available at

https://github.com/ratchanonp/imageprocessing/blob/main/Assignment_3/Assignment_3.ipynb

Source code มีการใช้งาน Library : OpenCV, NumPy และ Matplotlib

Smoothing filter

Smoothing filter to remove noise from the given images.

การทำ Smoothing filter 2 แบบใช้ Source code เดียวกันมีการแยกระหว่าง 2 algorithm โดยแบ่งเป็น 2 function คือ `average_kernel` และ `median_kernel` โดยจะเรียกใช้ผ่าน function `average_filter` และ `median_filter`

Source code

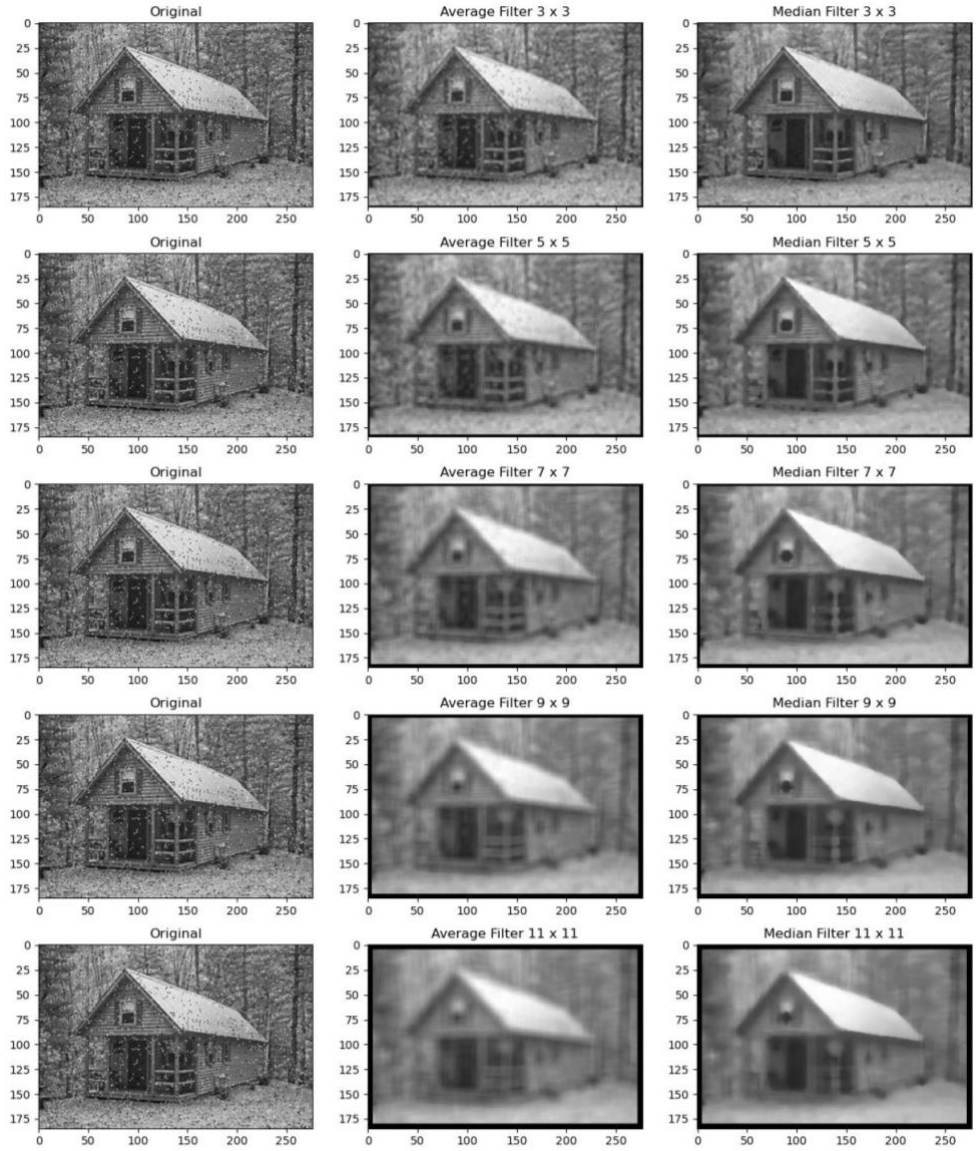
```
def get_kernel(img, x, y, kernelSize):
    padding = kernelSize//2
    return img[x-padding:x+padding+1, y-padding:y+padding+1]


def average_kernel(kernel):
    return np.mean(kernel)


def median_kernel(kernel):
    return np.median(kernel)


def apply_kernel(img, kernelSize, kernelFunc):
    # Get the dimensions of the image
    rows, cols = img.shape
    # Create a blank image of the same size as the original
    img_avg = np.zeros((rows, cols), dtype=np.uint8)
    # Get the kernel size
    k = kernelSize
    # Loop through the image
    padding = k//2
    xStart, xEnd = 0 + padding, rows - padding
    yStart, yEnd = 0 + padding, cols - padding

    for i in range(xStart, xEnd):
        for j in range(yStart, yEnd):
            # Get the kernel
            kernel = get_kernel(img, i, j, kernelSize)
            # Get the average of the kernel
            avg = kernelFunc(kernel)
            # Set the pixel value to the average
            img_avg[i, j] = avg

    return img_avg


def average_filter(img, kernelSize=3):
    return apply_kernel(img, kernelSize, average_kernel)


def median_filter(img, kernelSize=3):
    return apply_kernel(img, kernelSize, median_kernel)
```

```
# Read the image
noisy_img1 = cv2.imread("img/noisy_img1.jpg")

# Convert to grayscale
noisy_img1 = cv2.cvtColor(noisy_img1, cv2.COLOR_BGR2GRAY)

# Kernel size
kernelSizes = [3, 5, 7, 9, 11]

for kernelSize in kernelSizes:
    # Average filter
    img_avg = average_filter(noisy_img1, kernelSize)
    # Median filter
    img_med = median_filter(noisy_img1, kernelSize)

    # Plot the images
    plt.figure(figsize=(15, 15))

    plt.subplot(131)
    plt.imshow(noisy_img1, cmap="gray")
    plt.title("Original")

    plt.subplot(132)
    plt.imshow(img_avg, cmap="gray")
    plt.title("Average Filter")

    plt.subplot(133)
    plt.imshow(img_avg, cmap="gray")
    plt.title("Median Filter")


cv2.imwrite(f"output/noise_reduction/avg/noisy_img1_avg_{kernelSize}.j
pg", img_avg)

cv2.imwrite(f"output/noise_reduction/med/noisy_img1_med_{kernelSize}.j
pg", img_med)

    plt.show()
```

```
# Read the image
noisy_img2 = cv2.imread("img/noisy_img2.jpg")

# Convert the image to grayscale
noisy_img2 = cv2.cvtColor(noisy_img2, cv2.COLOR_BGR2GRAY)

# For each kernel size
for kernelSize in kernelSizes:
    # Apply the average filter
    img_avg = average_filter(noisy_img2, kernelSize)

    # Apply the median filter
    img_med = median_filter(noisy_img2, kernelSize)

    # Plot the images
    plt.figure(figsize=(15, 15))

    plt.subplot(131)
    plt.imshow(noisy_img2, cmap="gray")
    plt.title("Original")

    plt.subplot(132)
    plt.imshow(img_avg, cmap="gray")
    plt.title("Average Filter")

    plt.subplot(133)
    plt.imshow(img_avg, cmap="gray")
    plt.title("Median Filter")


cv2.imwrite(f"output/noise_reduction/avg/noisy_img2_avg_{kernelSize}.j
pg", img_avg)

cv2.imwrite(f"output/noise_reduction/med/noisy_img2_med_{kernelSize}.j
pg", img_med)

    plt.show()
```

ผลลัพธ์จากการทำ Averaging filter และ Median filter

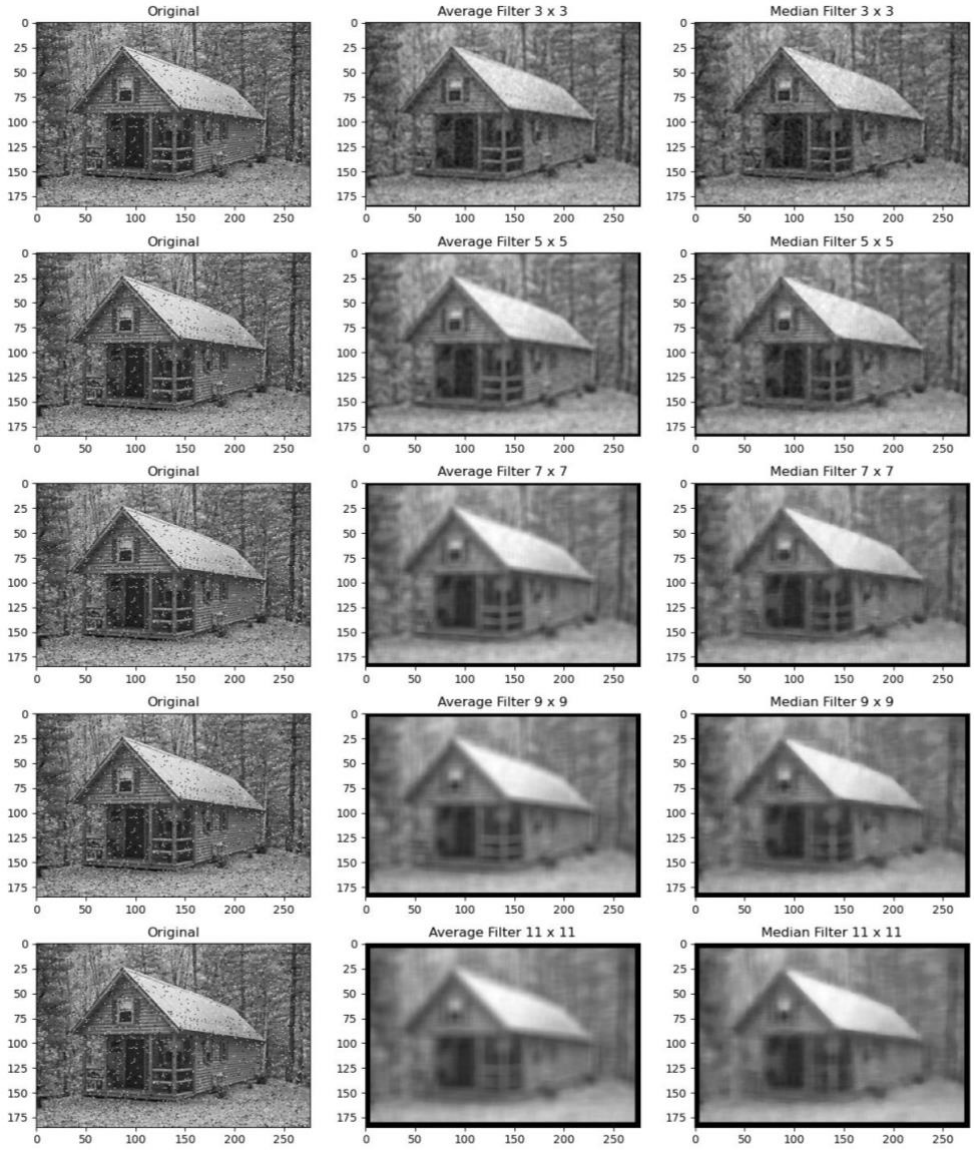
noisy_img1.jpg



ในภาพ noisy_img1.jpg จะสังเกตได้ว่าภาพมีทั้ง Salt noise และ Pepper noise จากการทดลอง Smoothing ภาพนี้พบว่า Median filter นั้นให้ผลลัพธ์ของการ Smoothing ที่ดีกว่าสามารถทำให้ Salt & Pepper noise ภายในรูปภาพหายได้โดยที่มีความ Blur น้อยกว่า Average filter โดยเพียง Median filter ขนาด 3x3 ก็สามารถลด Salt & Pepper noise ได้แล้ว

หากเปรียบเทียบผลลัพธ์ของ Average filter และ Median filter ที่ขนาด 3x3 พบว่า Median filter ที่ขนาด 3x3 ลบ Noise ได้เกือบหมดและยังคงไว้ซึ่งรายละเอียดภายในภาพ ถ้าหากสังเกตที่ Average filter พบว่ายังมี Noise ภายในภาพและมีการสูญเสียรายละเอียดมากกว่า Median filter อย่างเห็นได้ชัด

noisy_img2.jpg



ในภาพ noisy_img2.jpg นั้นจะสังเกตได้ว่ามี Salt & Pepper noise เหมือนภาพ noisy_img1.jpg แต่มีเยอะและอยู่ใกล้ชิดกันมากกว่า ผลลัพธ์จากการทำ Smoothing ทั้ง 2 algorithm พบว่า Salt & Pepper noise ส่วนใหญ่หายไปที่ Filter ขนาด 5x5 โดยถ้าหาก filter มีขนาดใหญ่กว่า 5x5 ไปแล้วจะเห็นภาพที่ค่อนข้างภาพเสียรายละเอียดส่วนใหญ่

โดยเมื่อเปรียบเทียบผลลัพธ์ที่ได้จาก Filter ขนาด 5x5 ทั้ง 2 algorithm พบว่า Median filter ก็ยังคงให้รายละเอียดของภาพที่ได้กว่า Average filter รวมทั้งยังลด Noise ได้มากกว่าอีกด้วย

ทั้ง 2 Algorithm นั้นมีการใช้ Kernel ทำให้มีระยะ Padding โดยทางผมนั้นได้ใช้วิธีการให้ระยะ Padding ที่ไม่ได้ผ่าน kernel ที่ทำการ filter นั้นเป็นสีดำโดยสามารถสังเกตได้โดยบริเวณขอบรูปนั้นจะมีกรอบสีดำปรากฏ

จากการทดลองข้างต้นโดยใช้ภาพที่มี Salt & Pepper noise สามารถบอกว่า Median filter เป็น algorithm ที่สามารถทำ Image smoothing ได้ดีโดยที่รายละเอียดของภาพยังคงอยู่ ซึ่งแตกต่างจาก Average filter ที่ทำให้ภาพนั้นเบลอ

Sharpening Filter

Source code ส่วนของ Utility Functions สำหรับเรียกใช้งาน

```
def kernelize(img, kernel):
    """
    Apply the kernel to the image
    """
    return cv2.filter2D(img, -1, kernel)

def sharpen(img, kernel):
    """
    Sharpen the image by subtract the kernelized image from the original one
    """
    return cv2.subtract(img, kernel)

def display_images(images, rows=1, cols=1):
    """
    Display multiple images with their titles
    images: list of dictionaries containing the image title and position (row, col)
    """
    fig, axes = plt.subplots(rows, cols, figsize=(15, 8))
    # axis off for all subplots
    for ax in axes.ravel():
        ax.axis("off")

    if len(axes.shape) == 1:
        axes = np.expand_dims(axes, axis=0)

    for i, image in enumerate(images):
        img, title, position = image["img"], image["title"], image["position"]
        row, col = position

        axes[row, col].imshow(img, cmap="gray")
        axes[row, col].set_title(title)
        axes[row, col].axis("off")

    plt.show()
```

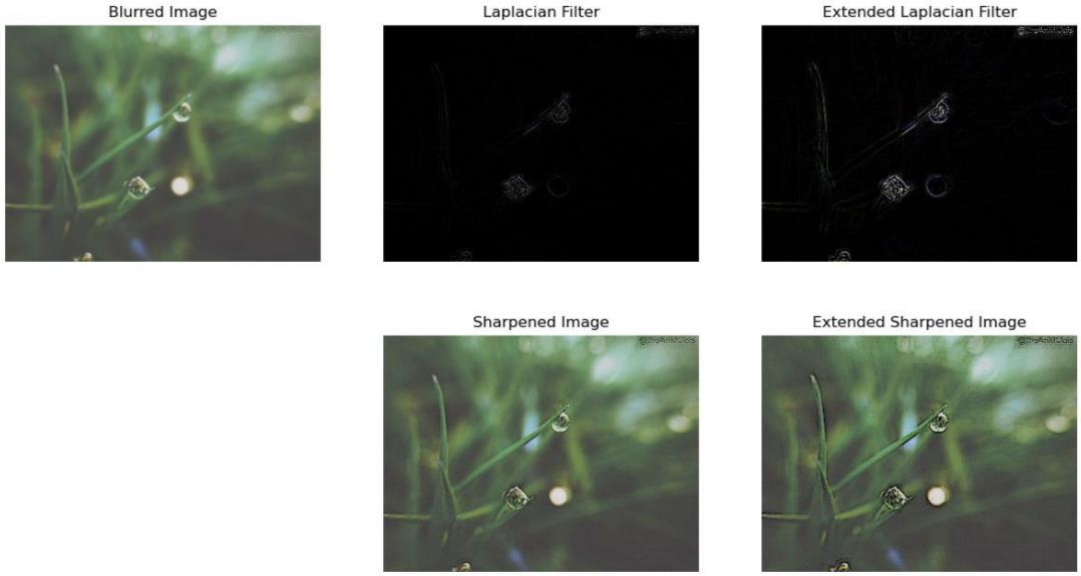
มีการใช้ Built-in function ของ OpenCV ในการ Apply filter กับรูปภาพแทนการเขียน Iteration ผ่านรูปภาพเอง

Laplacian filter

Source code

```
blurred_img = cv2.imread("img/blurred_image.jpg")

# Kernel
laplacian_kernel = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])
extended_laplacian_kernel = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]])

# For each R G B channel
laplacian_img = np.zeros_like(blurred_img)
extended_laplacian_img = np.zeros_like(blurred_img)

shapened_img = np.zeros_like(blurred_img)
extended_shapened_img = np.zeros_like(blurred_img)

for i in range(3):
    laplacian_img[:, :, i] = kernelize(blurred_img[:, :, i], laplacian_kernel)
    extended_laplacian_img[:, :, i] = kernelize(blurred_img[:, :, i], extended_laplacian_kernel)

    shapened_img[:, :, i] = sharpen(blurred_img[:, :, i], laplacian_img[:, :, i])
    extended_shapened_img[:, :, i] = sharpen(blurred_img[:, :, i], extended_laplacian_img[:, :,
i])


images = [
    {"img": blurred_img, "title": "Blurred Image", "position": (0, 0)},
    {"img": laplacian_img, "title": "Laplacian Filter", "position": (0, 1)},
    {"img": extended_laplacian_img, "title": "Extended Laplacian Filter", "position": (0, 2)},
    {"img": shapened_img, "title": "Sharpened Image", "position": (1, 1)},
    {"img": extended_shapened_img, "title": "Extended Sharpened Image", "position": (1, 2)},
]

display_images(images, rows=2, cols=3)

cv2.imwrite("output/sharpened/laplacian/blurred_img_laplacian.jpg", laplacian_img)
cv2.imwrite("output/sharpened/laplacian/blurred_img_extended_laplacian.jpg",
extended_laplacian_img)
cv2.imwrite("output/sharpened/laplacian/blurred_img_sharpened.jpg", shapened_img)
cv2.imwrite("output/sharpened/laplacian/blurred_img_extended_sharpened.jpg",
extended_shapened_img)
```

ผลลัพธ์จากการทำ Image Sharpening โดยใช้ Laplacian Filter



จากโจทย์ให้รูปภาพเป็นรูปภาพสี จึงตัดสินใจทำ Image Sharpening เป็นรูปภาพสีโดยแยกทำ Image Sharpening เป็น 3 Channels RGB โดยได้ทำ Laplacian filter 2 แบบคือแบบธรรมดาและ Extended Laplacian โดยหากสังเกตตัว Filter ที่ได้ จะพบว่า Extended Laplacian Filter นั้นจะมีรายละเอียดมากกว่า Laplacian ปกติ ทำให้เมื่อนำ Filter ไปใช้งานแล้วและเปรียบเทียบผลลัพธ์ที่ได้จะเห็นว่า Extended Laplacian Filter นั้นให้ภาพที่ขอบของ Object คมชัดกว่า และยังทำให้บริเวณพื้นหลังของภาพมี Contrast ที่เพิ่มขึ้นด้วย หากสังเกตเจาะจงไปในบริเวณที่เป็นหยดน้ำจะเห็นว่าขอบของหยดน้ำใน Extended Laplacian filter นั้นชัดเจนกว่า แต่จะพบว่าบริเวณหยดนั้นเกิด Noise ของสีที่เกิดจากการทำ Image Sharpening เกิดขึ้น

Gradient filter

Sobel filter

Source code

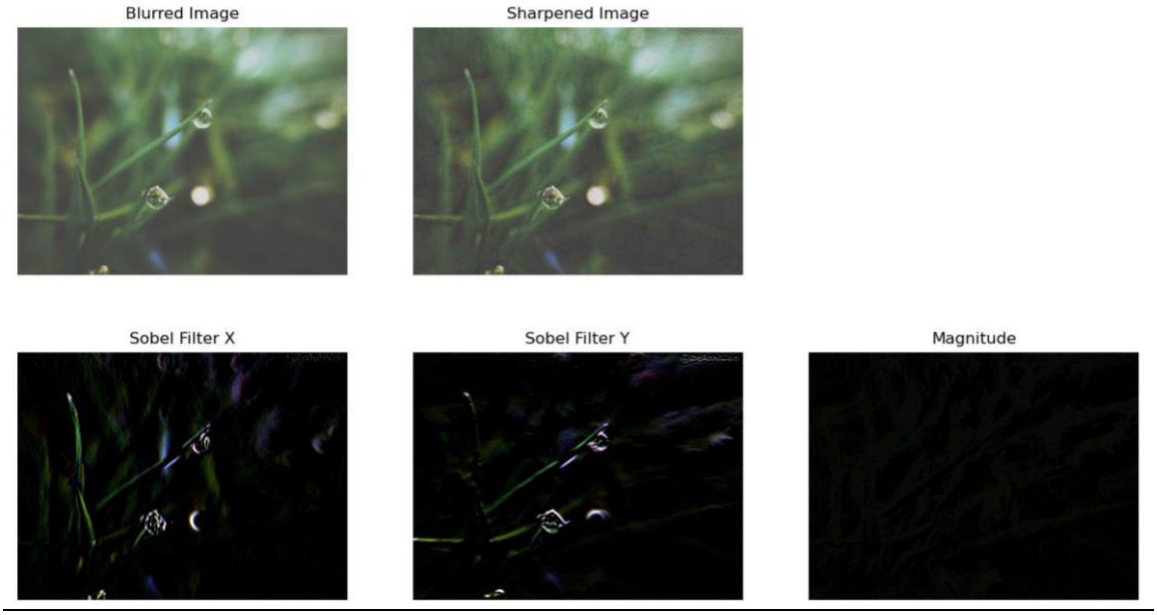
```
# Sobel
sobel_kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
sobel_kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

# For each R G B channel
sobel_img_x = np.zeros_like(blurred_img)
sobel_img_y = np.zeros_like(blurred_img)

magnitude = np.zeros_like(blurred_img)

shapened_img_sobel = np.zeros_like(blurred_img)

for i in range(3):
    # Apply the kernel
    sobel_img_x[:, :, i] = cv2.filter2D(blurred_img[:, :, i], -1, sobel_kernel_x)
    sobel_img_y[:, :, i] = cv2.filter2D(blurred_img[:, :, i], -1, sobel_kernel_y)

    # Add the result to the original image
    magnitude[:, :, i] = np.sqrt(sobel_img_x[:, :, i]**2 + sobel_img_y[:, :, i]**2)

    # Add the result to the original image
    shapened_img_sobel[:, :, i] = blurred_img[:, :, i] - magnitude[:, :, i]

images = [
    {"img": blurred_img, "title": "Blurred Image", "position": (0, 0)},
    {"img": sobel_img_x, "title": "Sobel Filter X", "position": (1, 0)},
    {"img": sobel_img_y, "title": "Sobel Filter Y", "position": (1, 1)},
    {"img": magnitude, "title": "Magnitude", "position": (1, 2)},
    {"img": shapened_img_sobel, "title": "Sharpened Image", "position": (0, 1)},
]

display_images(images, rows=2, cols=3)
cv2.imwrite("output/sharpened/sobel/blurred_img_sobel_x.jpg", sobel_img_x)
cv2.imwrite("output/sharpened/sobel/blurred_img_sobel_y.jpg", sobel_img_y)
cv2.imwrite("output/sharpened/sobel/blurred_img_magnitude.jpg", magnitude)
cv2.imwrite("output/sharpened/sobel/blurred_img_sharpened_sobel.jpg", shapened_img_sobel)
```

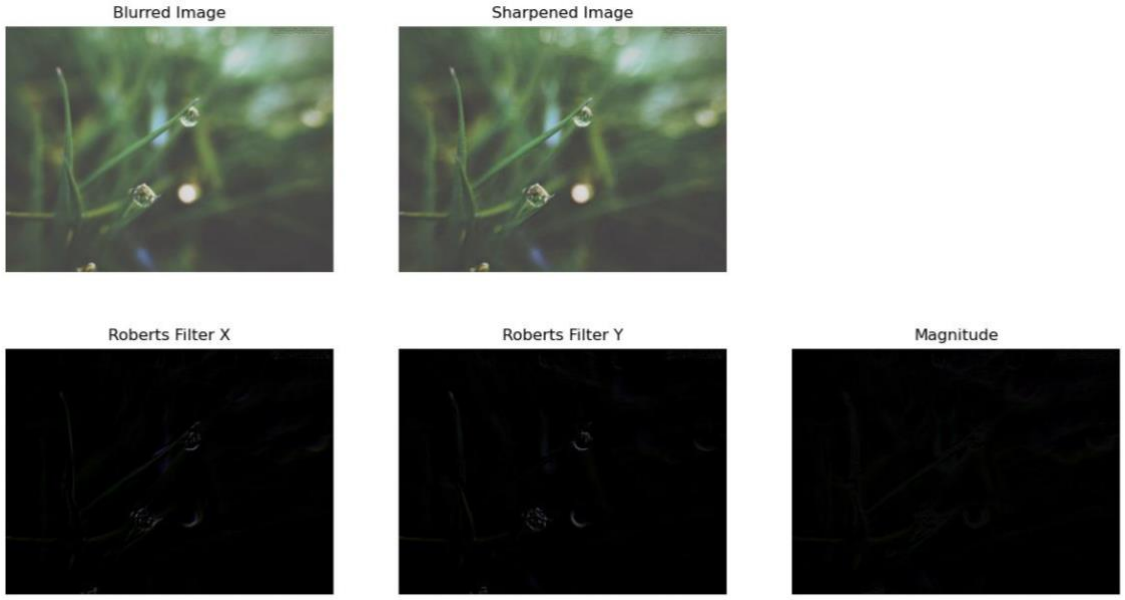
Robert filter

Source code

```
# Roberts
roberts_kernel_x = np.array([[1, 0], [0, -1]])
roberts_kernel_y = np.array([[0, 1], [-1, 0]])

# For each R G B channel
roberts_img_x = np.zeros_like(blurred_img)
roberts_img_y = np.zeros_like(blurred_img)

magnitude = np.zeros_like(blurred_img)

shapened_img_roberts = np.zeros_like(blurred_img)

for i in range(3):
    # Apply the kernel
    roberts_img_x[:, :, i] = cv2.filter2D(blurred_img[:, :, i], -1, roberts_kernel_x)
    roberts_img_y[:, :, i] = cv2.filter2D(blurred_img[:, :, i], -1, roberts_kernel_y)

    # Add the result to the original image
    magnitude[:, :, i] = np.sqrt(roberts_img_x[:, :, i]**2 + roberts_img_y[:, :, i]**2)

    # Add the result to the original image
    shapened_img_roberts[:, :, i] = blurred_img[:, :, i] - magnitude[:, :, i]

images = [
    {"img": blurred_img, "title": "Blurred Image", "position": (0, 0)},
    {"img": roberts_img_x, "title": "Roberts Filter X", "position": (1, 0)},
    {"img": roberts_img_y, "title": "Roberts Filter Y", "position": (1, 1)},
    {"img": magnitude, "title": "Magnitude", "position": (1, 2)},
    {"img": shapened_img_roberts, "title": "Sharpened Image", "position": (0, 1)},
]

display_images(images, rows=2, cols=3)
cv2.imwrite("output/sharpened/robert/blurred_img_roberts_x.jpg", roberts_img_x)
cv2.imwrite("output/sharpened/robert/blurred_img_roberts_y.jpg", roberts_img_y)
cv2.imwrite("output/sharpened/robert/blurred_img_roberts_magnitude.jpg", magnitude)
cv2.imwrite("output/sharpened/robert/blurred_img_sharpened_roberts.jpg", shapened_img_roberts)
```

ผลลัพธ์จากการทำ Image Sharpening โดยใช้ Gradient Filter 2 แบบ





ผลลัพธ์จากการทำ Image Sharpening โดยใช้ Gradient filter นั้นพบว่าทำให้ภาพนั้น Sharpened ขึ้นแต่ไม่ได้ทำให้ Sharpened ขึ้นเท่ากับ Extended Laplacian Filter โดย Gradient filter ทั้ง 2 แบบนั้นยังทำให้พื้นหลังของภาพมี contrast มากขึ้น

เมื่อเปรียบเทียบระหว่าง Gradient filter 2 แบบ Robert filter นั้นให้ภาพที่คมชัดมากกว่า Sobel filter โดยสามารถสังเกตได้บริเวณหยดน้ำจะเห็นได้ว่าจะมีขอบที่ชัดมากกว่า

สรุปผลคือการทำ Image Sharpening นั้นสามารถทำได้ทั้ง Laplacian filter และ Gradient filter โดย algorithm ที่ตอบโจทย์ของการทำ Image Sharpening ของรูปนี้ที่สุดคือ Extended Laplacian filter แต่ยังพบข้อเสียของการทำ Image Sharpening แบบสีอยู่ คือจะเกิด Noise ที่เป็นค่าผิดพลาดจากการนำค่า RGB ในแต่ละ Channel ไปเข้า filter แบบต่างๆ จะเห็นความผิดปกติได้ชัดบริเวนหยดน้ำ