EE604 Assignment 1

Question 1

1. **Load and Preprocess the Image**: Load the image using `PIL`, convert it to a numpy array in RGB format, and reverse the channels for OpenCV (BGR).
2. **Raindrop Detection (Mask Creation)**: Set upper and lower brightness thresholds to identify raindrops and create a binary mask using `cv2.inRange`.
3. **Mask Application and Removal**: Apply the mask using `cv2.bitwise_and` to isolate raindrops, and subtract this mask from the original image to remove the raindrops.
4. **Mask Refinement**: Dilate the mask to cover larger raindrop areas and apply median filtering to reduce noise and artifacts.
5. **Inpainting**: Use `cv2.inpaint` to restore the removed raindrop regions and display/save the final image.

```python
import numpy as np
import cv2
from PIL import Image

# Load image and convert it to numpy array (RGB)
img = Image.open('image.png')
img_cv2 = np.array(img.convert('RGB'))[..., ::-1]

# Thresholds to detect raindrops (high brightness areas)
low_limit = np.array([140, 140, 140])
high_limit = np.array([250, 250, 250])

# Create a binary mask using the defined thresholds
mask = cv2.inRange(img_cv2, low_limit, high_limit)

# Generate the mask as an RGB image to be used for subtraction
rgb_removed = cv2.bitwise_and(img_cv2, img_cv2, mask=mask)

# Subtract the identified raindrops from the original image
img_no_raindrops = img_cv2 - rgb_removed

# Apply dilation to enlarge the masked regions
structure_element = np.ones((3, 3), np.uint8)
dilated_mask = cv2.dilate(mask, structure_element, iterations=2)

# Smooth the image with a median filter to reduce noise
median_smoothed = cv2.medianBlur(img_cv2, 7)
```

EE604 Assignment 1

```
# Apply inpainting to the masked areas to interpolate missing information
final_img_inpaint = cv2.inpaint(median_smoothed, dilated_mask,
inpaintRadius=3, flags=cv2.INPAINT_TELEA)

# Convert BGR to RGB for final display
final_inpaint_rgb = final_img_inpaint[..., ::-1]

# Display final image in OpenCV window
cv2.imshow('Inpainted Image', final_inpaint_rgb[:, :, ::-1])
cv2.imwrite('output.png', final_inpaint_rgb[:, :, ::-1])
cv2.waitKey(0)
cv2.destroyAllWindows()
```

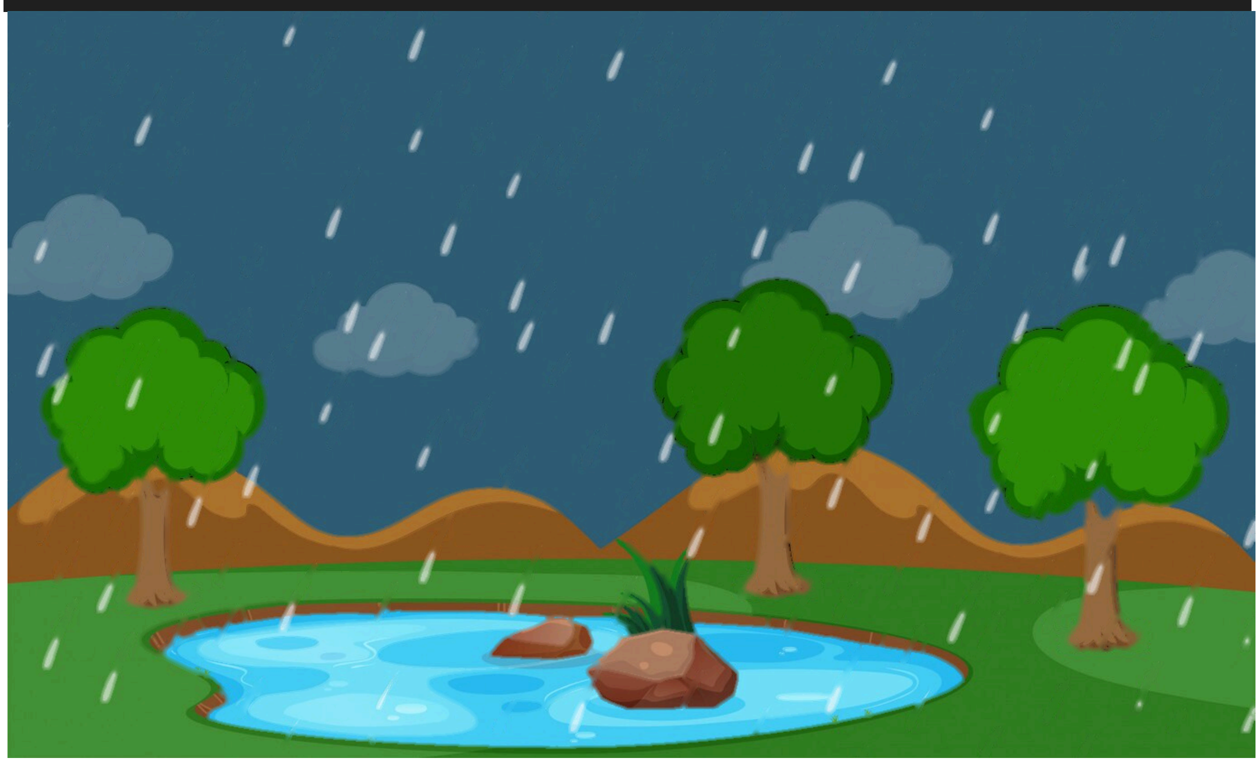
EE604 Assignment 1

Before I was doing this

```
import cv2
import numpy as np

# Load the image
image_path = "output9_image.png"
image = cv2.imread(image_path)

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

# Apply GaussianBlur to reduce noise
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)

# First pass: Detect larger raindrops with Canny edge detection
edges_large = cv2.Canny(blurred_image, 50, 150)

# Use dilation and erosion to highlight raindrop contours (adjust
parameters)
kernel_large = np.ones((3, 3), np.uint8)
dilated_large = cv2.dilate(edges_large, kernel_large, iterations=2)
eroded_large = cv2.erode(dilated_large, kernel_large, iterations=1)

# Inpaint the detected areas from the first pass
inpainted_image = cv2.inpaint(image, eroded_large, inpaintRadius=7,
flags=cv2.INPAINT_TELEA)

# Second pass: Detect smaller, fainter raindrops with adjusted Canny
thresholds
edges_small = cv2.Canny(blurred_image, 30, 100)  # More sensitive to faint
edges

# Use smaller kernel to refine the detection of smaller raindrops
kernel_small = np.ones((2, 2), np.uint8)
dilated_small = cv2.dilate(edges_small, kernel_small, iterations=1)
eroded_small = cv2.erode(dilated_small, kernel_small, iterations=1)

# Apply inpainting again for the second pass
```

EE604 Assignment 1

```
final_inpainted_image = cv2.inpaint(inpainted_image, eroded_small,
inpaintRadius=5, flags=cv2.INPAINT_TELEA)

# Save the result
output_path = "output10_image.png"
cv2.imwrite(output_path, final_inpainted_image)

# Display the final result
cv2.imshow("Final Inpainted Image", final_inpainted_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



1. **Load and Preprocess the Image**: The image is loaded, converted to grayscale, and blurred using GaussianBlur to reduce noise.
2. **Edge Detection for Larger Raindrops**: You used the Canny edge detection with a high threshold to detect the edges of larger raindrops.
3. **Morphological Operations (Larger Raindrops)**: Applied dilation and erosion with a larger kernel to better highlight the contours of larger raindrops.

4. **Inpainting (First Pass)**: Performed inpainting on the detected large raindrop areas using `cv2.inpaint` with a larger radius.
5. **Edge Detection for Smaller Raindrops**: Performed a second Canny edge detection with lower thresholds to catch smaller, fainter raindrops.
6. **Morphological Operations (Smaller Raindrops)**: Applied similar dilation and erosion with a smaller kernel to refine the detection of smaller raindrops.
7. **Inpainting (Second Pass)**: Inpainted the smaller raindrop areas for a more refined restoration.

Improvements:

- **Two-Pass Detection**: You added a second pass with adjusted parameters for detecting smaller, fainter raindrops, improving overall precision in detecting raindrops of varying sizes.
- **Refined Morphological Operations**: By using different kernel sizes for larger and smaller raindrop detections, the contours of the raindrops are more accurately highlighted, reducing over-processing.
- **Gradual Inpainting**: The two-step inpainting method (large raindrops first, then small) ensures a better and more natural restoration by handling larger disruptions first before moving to finer details.

Question 2

1. **Create the Image Canvas**: Initialize a 512x512 white canvas using NumPy to serve as the background for drawing the logo.
2. **Draw Concentric Circles**: Use the `draw_curve` function to draw three concentric circles representing the outer, middle, and inner rings of the logo.
3. **Draw Spokes**: Compute angles and use trigonometric functions to draw 24 spokes between two radii, forming the internal structure of the logo.
4. **Add Decorative Curves and Circles**: Add additional curves and small circles in various positions to enhance the design, ensuring symmetry and balance.
5. **Display and Save the Image**: Render the final image using OpenCV's `imshow` and save it to a file.

```
import numpy as np
import cv2
import math
```

EE604 Assignment 1

```python
# Function to draw a curve based on polar coordinates
def draw_curve(image, center, radius, angle_start, angle_end, color,
thickness, verbose=False):
    # Adjust for angle wrapping
    if angle_end < angle_start:
        angle_end += 360

    # Number of points to approximate the curve
    num_points = 1000
    step = (angle_end - angle_start) / num_points

    # List to store curve points
    curve_points = []

    # Generate the curve points based on angle and radius
    for i in range(num_points + 1):
        angle = angle_start + i * step
        angle = angle % 360  # Keep angle within 0-360 degrees
        x = int(center[0] + radius * math.cos(math.radians(angle)))
        y = int(center[1] + radius * math.sin(math.radians(angle)))
        curve_points.append((x, y))

    # If verbose mode is enabled, print the first and last points
    if verbose:
        print(f"Initial : {curve_points[0]}")
        print(f"Final : {curve_points[-1]}")

    # Draw the curve by connecting the points
    for i in range(len(curve_points) - 1):
        cv2.line(image, curve_points[i], curve_points[i + 1], color,
thickness)

# Define the dimensions of the canvas
width = 512
height = 512

# Create an empty white image (logo)
logo = np.ones((height, width), dtype=np.uint8) * 255
```

EE604 Assignment 1

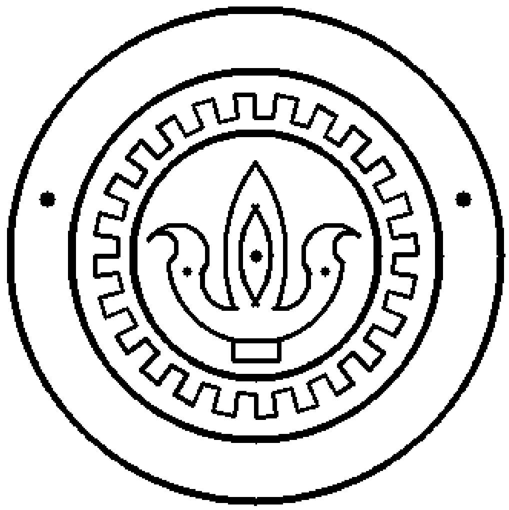
```
# Define the center coordinates of the logo
x_centre = width // 2
y_centre = height // 2

# Draw three concentric circles to form the outer, middle, and inner rings
outer_r = 160
draw_curve(logo, (x_centre, y_centre), outer_r, 0, 360, 0, 4)

middle_r = 120
draw_curve(logo, (x_centre, y_centre), middle_r, 0, 360, 0, 4)

inner_r = 80
draw_curve(logo, (x_centre, y_centre), inner_r, 0, 360, 0, 4)

# Define two radii to create spoke shapes
r1 = 90
r2 = 105

# Define step angle to draw 24 spokes
step = 360 // 24

# Draw the spokes using lines between two radii
for i in range(24):
   angle = i * step

   # Calculate points for the pentagonal spoke sections
   point_1_x = int(x_centre + r1 * math.cos(math.radians(angle)))
   point_1_y = int(y_centre + r1 * math.sin(math.radians(angle)))

   point_2_x = int(x_centre + r1 * math.cos(math.radians(angle + step /
2)))
   point_2_y = int(y_centre + r1 * math.sin(math.radians(angle + step /
2)))

   point_3_x = int(x_centre + r2 * math.cos(math.radians(angle + step /
2)))
```

```
    point_3_y = int(y_centre + r2 * math.sin(math.radians(angle + step / 
2)))

    point_4_x = int(x_centre + r2 * math.cos(math.radians(angle + step)))
    point_4_y = int(y_centre + r2 * math.sin(math.radians(angle + step)))

    point_5_x = int(x_centre + r1 * math.cos(math.radians(angle + step)))
    point_5_y = int(y_centre + r1 * math.sin(math.radians(angle + step)))

    # Draw lines between the calculated points to form the spokes
    cv2.line(logo, (point_1_x, point_1_y), (point_2_x, point_2_y), 0, 2)
    cv2.line(logo, (point_2_x, point_2_y), (point_3_x, point_3_y), 0, 2)
    cv2.line(logo, (point_3_x, point_3_y), (point_4_x, point_4_y), 0, 2)
    cv2.line(logo, (point_4_x, point_4_y), (point_5_x, point_5_y), 0, 2)

# Draw smaller curved sections on either side of the inner circle
draw_curve(logo, (x_centre+inner_r, y_centre), 100, 160, 217, 0, 2)
draw_curve(logo, (x_centre-inner_r, y_centre), 100, 323, 20, 0, 2)

# Additional curves to enhance the design near the middle
draw_curve(logo, (x_centre+51, y_centre), 20, 140, 310, 0, 2)
draw_curve(logo, (x_centre-51, y_centre), 20, 230, 40, 0, 2)

draw_curve(logo, (x_centre+15, y_centre+15), 20, 330, 100, 0, 2)
draw_curve(logo, (x_centre-15, y_centre+15), 20, 80, 210, 0, 2)

# Another set of curves at the center
draw_curve(logo, (x_centre, y_centre), 57, 5, 75, 0, 2)
draw_curve(logo, (x_centre, y_centre), 57, 105, 175, 0, 2)

# Calculate the position for the small circles
right_x = int(x_centre + 140 * math.cos(math.radians(step)))
right_y = int(y_centre - 140 * math.sin(math.radians(step)))

left_x = int(x_centre - 140 * math.cos(math.radians(step)))
left_y = int(y_centre - 140 * math.sin(math.radians(step)))

# Draw additional lines and rectangles near the center
```

EE604 Assignment 1

```
cv2.line(logo, (x_centre+15, y_centre+55), (x_centre-15, y_centre+55), 0,
2)
cv2.line(logo, (x_centre+15, y_centre+55), (x_centre+15, y_centre+68), 0,
2)
cv2.line(logo, (x_centre-15, y_centre+55), (x_centre-15, y_centre+68), 0,
2)
cv2.line(logo, (x_centre+15, y_centre+68), (x_centre-15, y_centre+68), 0,
2)

# Additional curves on both sides of the center
draw_curve(logo, (x_centre+61, y_centre-5), 10, 110, 310, 0, 2)
draw_curve(logo, (x_centre-61, y_centre-5), 10, 210, 55, 0, 2)

# Final curves to complete the design
draw_curve(logo, (x_centre-40, y_centre), 50, 320, 40, 0, 2)
draw_curve(logo, (x_centre+40, y_centre), 50, 140, 220, 0, 2)

# Draw small black circles at specific positions
cv2.circle(logo, (right_x, right_y), 5, 0, -1)
cv2.circle(logo, (left_x, left_y), 5, 0, -1)
cv2.circle(logo, (x_centre, y_centre), 4, 0, -1)
cv2.circle(logo, (x_centre+45, y_centre+10), 3, 0, -1)
cv2.circle(logo, (x_centre-45, y_centre+10), 3, 0, -1)

# Display the generated logo in a window
cv2.imshow("IITK Logo", logo)

# Wait for a key press and close the window
cv2.waitKey(0)
cv2.destroyAllWindows()

# Save the logo as an image file
cv2.imwrite('iitk_logo.png', logo)
```

EE604 Assignment 1



Currently, I was able to do it only up to here. I also generated characters for the English alphabet. But I hadn't positioned them in the image.