Assignment 1: Image Processing (EE604)

Chirag Garg (210288)

September 11, 2024

Question 1: Removing Raindrops from an Image

Problem Statement

The task is to write a program that removes raindrops from a given image so that the resulting image appears as if there was no rain in the scene.

Approach

To remove the raindrops from the image, the following steps were taken:

1. **Image Reading and Conversion**: The image is read using OpenCV and converted from BGR to RGB for better compatibility with matplotlib.
2. **Median Filtering**: A median filter is applied to reduce noise and smooth the image, helping in distinguishing the raindrops.
3. **Difference Calculation**: The difference between the original and the median-filtered image is calculated to isolate potential raindrop areas.
4. **Grayscale Conversion and Thresholding**: The difference image is converted to grayscale and a binary threshold is applied to create a mask that identifies raindrop locations.
5. **Dilation**: To connect nearby raindrops, the mask is dilated using a small kernel.
6. **Inpainting**: The identified raindrop areas are inpainted using OpenCV's Telea method to fill in the gaps where raindrops were detected.

Code

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def remove_rain_drops(image_path):
    # Read the image
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB for matplotlib

    # Apply median filter
    median = cv2.medianBlur(img, 23)

    # Subtract median from original to isolate potential rain drops
    diff = cv2.absdiff(img, median)

    # Convert difference to grayscale
    gray_diff = cv2.cvtColor(diff, cv2.COLOR_RGB2GRAY)

    # Threshold to identify rain drops
    _, rain_mask = cv2.threshold(gray_diff, 30, 255, cv2.THRESH_BINARY)

    # Dilate to connect nearby rain drops
    kernel = np.ones((3,3), np.uint8)
    dilated_mask = cv2.dilate(rain_mask, kernel, iterations=1)

    # Inpaint only the areas identified as rain drops
    result = cv2.inpaint(img, dilated_mask, 3, cv2.INPAINT_TELEA)

    return img, diff, result

def plot_image(image, title):
    plt.imshow(image)
    plt.title(title)
    plt.axis('off')
    plt.show()

# Usage
input_image = "rain.png"
original, subtracted, processed = remove_rain_drops(input_image)

plot_image(original, "Original Image")
plot_image(subtracted, "Subracted Image")
plot_image(processed, "Final image")
```

Results

Below is the original image, the Subtracted Image (Original - Filtered), and the processed image with raindrops removed.

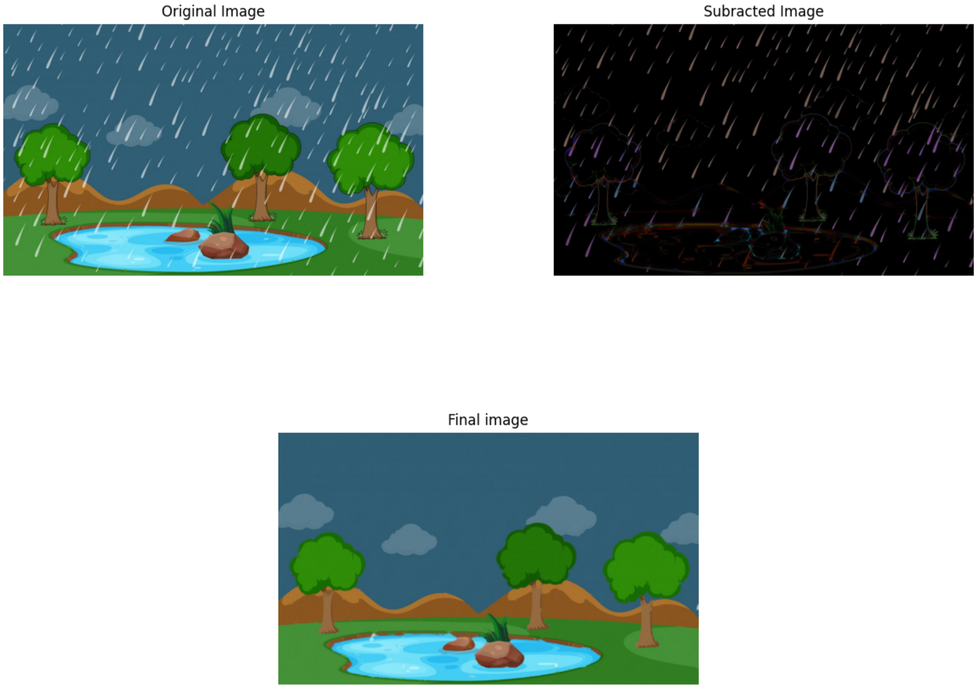

Figure 1: Images showing the original image, intermediate steps, and the final processed image.

Question 2: Recreating the IITK Logo

Problem Statement

The task is to recreate the IITK logo from scratch in binary format.

Approach

To recreate the IITK logo, the following approach was used:

1. **Canvas Creation**: A blank image with a white background is created using PIL.
2. **Circle Drawing**: A large circle is drawn to form the base of the logo.
3. **Text Placement**: English and Hindi texts are drawn along the circular path using custom functions to position each character appropriately.
4. **Drawing Additional Elements**: Concentric circles, black dots, arcs, and geometric shapes are added to complete the logo design.
5. **Gear Drawing**: A gear with specified parameters is drawn to fit within the circular layout of the logo.

Code

```
from PIL import Image, ImageDraw, ImageFont
import math

######################
# Create a blank image with white background
img = Image.new('RGB', (1000, 1000), 'white')
```

```
draw = ImageDraw.Draw(img)

# Draw a circle
center = (500, 500)
radius = 300
draw.ellipse([center[0] - radius, center[1] - radius, center[0] + radius, center[1] +
↪  radius], outline='black')

######################

## writing texts

def draw_text_on_circle(draw, text, center, radius, font, angle_start, angle_length,
↪  direction='clockwise'):
    chars = list(text)
    text_length = len(chars)
    angle_step = angle_length / text_length if direction == 'clockwise' else
    ↪  -angle_length / text_length

    for i, char in enumerate(chars):
        angle = math.radians(angle_start + i * angle_step)
        x = center[0] + radius * math.cos(angle)
        y = center[1] + radius * math.sin(angle)

        char_img = Image.new('RGBA', (100, 100), (255, 255, 255, 0))
        char_draw = ImageDraw.Draw(char_img)
        char_draw.text((50, 50), char, font=font, fill='black', anchor='mm')

        rotation_angle = -math.degrees(angle) + 90

        # Rotate letters in the upper half of the circle by 180 degrees
        if direction == 'clockwise':
            rotation_angle += 180

        rotated_char_img = char_img.rotate(rotation_angle, expand=1)

        x_pos = int(x - rotated_char_img.width // 2)
        y_pos = int(y - rotated_char_img.height // 2)

        img.paste(rotated_char_img, (x_pos, y_pos), rotated_char_img)

# Draw English text
english_font_path = r'C:\Windows\Fonts\arial.ttf'  # Path to a suitable font
english_font_size = 35  # Adjust this to make the English text larger
english_font = ImageFont.truetype(english_font_path, english_font_size)
draw_text_on_circle(draw, "INDIAN INSTITUTE OF TECHNOLOGY KANPUR", center, radius +
↪  30, english_font, angle_start=190, angle_length=205, direction='anticlockwise')

# Draw Hindi text
hindi_font_path = r'C:\Users\HP\OneDrive\Desktop\EE604\Assignment_1\NotoSerifDevanaga⌋
↪  ri-VariableFont_wdth,wght.ttf'
hindi_font_size = 40
hindi_font = ImageFont.truetype(hindi_font_path, hindi_font_size)

hindi_text = "\u092D\u093E\u0930\u0924\u0940\u092F
↪  \u092A\u094D\u0930\u094C\u0926\u094D\u092F\u094B\u093F\u0917\u0915\u0940
↪  \u0938\u0902\u0938\u094D\u0925\u093E\u0928 \u0915\u093E\u0928\u092A\u0941\u0930"
```

```
draw_text_on_circle(draw, hindi_text, center, radius + 20, hindi_font,
↪  angle_start=210, angle_length=120, direction='clockwise')

# Draw concentric inner and outer circles to surround the text
outer_text_radius = radius + 65
inner_text_radius = radius - 90
draw.ellipse([center[0] - outer_text_radius, center[1] - outer_text_radius, center[0]
↪  + outer_text_radius, center[1] + outer_text_radius], outline='black')
draw.ellipse([center[0] - inner_text_radius, center[1] - inner_text_radius, center[0]
↪  + inner_text_radius, center[1] + inner_text_radius], outline='black')

######################

# Draw black dots
dot_radius = 10
dot_dist= radius + 30  # Distance of dots from center
angle1 = math.radians(200)  # angle for first dot
angle2 = math.radians(205 + 135)  # angle for second dot
# Start dot position
dot_x1 = center[0] + dot_dist* math.cos(angle1)
dot_y1 = center[1] + dot_dist* math.sin(angle1)

# End dot position
dot_x2 = center[0] + dot_dist* math.cos(angle2)
dot_y2 = center[1] + dot_dist* math.sin(angle2)

draw.ellipse([dot_x1 - dot_radius, dot_y1 - dot_radius, dot_x1 + dot_radius, dot_y1 +
↪  dot_radius], fill='black')
draw.ellipse([dot_x2 - dot_radius, dot_y2 - dot_radius, dot_x2 + dot_radius, dot_y2 +
↪  dot_radius], fill='black')

######################

# Draw the arc (semicircle)
arc_radius = 90
arc_bbox = [center[0] - arc_radius, center[1] - arc_radius, center[0] + arc_radius,
↪  center[1] + arc_radius]
draw.arc(arc_bbox, start=0, end=180, fill='black', width=2)

# left part
draw.line([(center[0]-arc_radius,center[1]),(center[0]-10-arc_radius,center[1]-20)],
↪  fill='black', width=2)
draw.line([(center[0]-10-arc_radius,center[1]-20),(center[0]-30-arc_radius,center[1]-⌋
↪  40)], fill='black',
↪  width=2)
draw.line([(center[0]-arc_radius-30,center[1]-40),(center[0]-45-arc_radius,center[1]-⌋
↪  50)], fill='black',
↪  width=2)
draw.line([(center[0]-arc_radius-45,center[1]-50),(center[0]-arc_radius,center[1]-35)⌋
↪  ], fill='black',
↪  width=2)
draw.line([(center[0]-arc_radius,center[1]-35),(center[0]-arc_radius+30,center[1]-15)⌋
↪  ], fill='black',
↪  width=2)
draw.line([(center[0]-arc_radius+30,center[1]-15),(center[0]-40,center[1]+30)],
↪  fill='black', width=2)

# center part
```

```
draw.line([(center[0]-40,center[1]+30),(center[0]-20,center[1]+30)], fill='black',
↪  width=2)
draw.line([(center[0]-20,center[1]+30),(center[0]-40,center[1]-30)], fill='black',
↪  width=2)
draw.line([(center[0]-40,center[1]-30),(center[0]-40,center[1]-60)], fill='black',
↪  width=2)
draw.line([(center[0]-40,center[1]-60),(center[0]-30,center[1]-90)], fill='black',
↪  width=2)
draw.line([(center[0]-30,center[1]-90),(center[0],center[1]-120)], fill='black',
↪  width=2)
draw.line([(center[0],center[1]-120),(center[0]+30,center[1]-90)], fill='black',
↪  width=2)
draw.line([(center[0]+40,center[1]-60),(center[0]+30,center[1]-90)], fill='black',
↪  width=2)
draw.line([(center[0]+40,center[1]-30),(center[0]+40,center[1]-60)], fill='black',
↪  width=2)
draw.line([(center[0]+20,center[1]+30),(center[0]+40,center[1]-30)], fill='black',
↪  width=2)
draw.line([(center[0]+40,center[1]+30),(center[0]+20,center[1]+30)], fill='black',
↪  width=2)

#right part
draw.line([(center[0]+arc_radius,center[1]),(center[0]+10+arc_radius,center[1]-20)],
↪  fill='black', width=2)
draw.line([(center[0]+10+arc_radius,center[1]-20),(center[0]+30+arc_radius,center[1]-⌋
↪  40)], fill='black',
↪  width=2)
draw.line([(center[0]+arc_radius+30,center[1]-40),(center[0]+45+arc_radius,center[1]-⌋
↪  50)], fill='black',
↪  width=2)
draw.line([(center[0]+arc_radius+45,center[1]-50),(center[0]+arc_radius,center[1]-35)⌋
↪  ], fill='black',
↪  width=2)
draw.line([(center[0]+arc_radius,center[1]-35),(center[0]+arc_radius-30,center[1]-15)⌋
↪  ], fill='black',
↪  width=2)
draw.line([(center[0]+arc_radius-30,center[1]-15),(center[0]+40,center[1]+30)],
↪  fill='black', width=2)

#bottom rectange
draw.line([(center[0]-20,center[1]+arc_radius+2),(center[0]-20,center[1]+arc_radius+3⌋
↪  0)], fill='black',
↪  width=2)
draw.line([(center[0]-20,center[1]+arc_radius+30),(center[0]+20,center[1]+arc_radius+⌋
↪  30)], fill='black',
↪  width=2)
draw.line([(center[0]+20,center[1]+arc_radius+2),(center[0]+20,center[1]+arc_radius+3⌋
↪  0)], fill='black',
↪  width=2)

######################

## making ellipse between left, ceenter and right part of center figure, with
↪  parameters ->
# a, b (axis lengths of ellipse), angle (to rotate) and fill (whether to color the
↪  ellipse or not)
def draw_tilted_ellipse(draw, center, a, b, angle, fill=False):
ellipse_img = Image.new('RGBA', (2 * a, 2 * b), (255, 255, 255, 0))  # Transparent
↪   background
ellipse_draw = ImageDraw.Draw(ellipse_img)

bbox = [0, 0, 2 * a, 2 * b]
if fill:
    ellipse_draw.ellipse(bbox, outline='black', fill='black', width=2)  # Fill
    ↪   with black if fill is True
else:
    ellipse_draw.ellipse(bbox, outline='black', width=2)  # No fill if fill is
    ↪   False

# Rotate the ellipse
rotated_ellipse = ellipse_img.rotate(angle, expand=True)
paste_position = (int(center[0] - rotated_ellipse.width / 2), int(center[1] -
↪   rotated_ellipse.height / 2))

# Paste the rotated ellipse back onto the main image
img.paste(rotated_ellipse, paste_position, rotated_ellipse)


draw_tilted_ellipse(img,[center[0],center[1]-35],20,50,0,False)
draw_tilted_ellipse(img,[center[0],center[1]-35],20,15,0,True)

draw_tilted_ellipse(img,[center[0]-arc_radius+20,center[1]+5],10,25,25,False)
draw_tilted_ellipse(img,[center[0]-arc_radius+20,center[1]+5],10,10,0,True)

draw_tilted_ellipse(img,[center[0]+arc_radius-20,center[1]+5],10,25,-25,False)
draw_tilted_ellipse(img,[center[0]+arc_radius-20,center[1]+5],10,10,0,True)


#####################

# Function to draw a gear with specified parameters
def draw_gear(draw, center, num_teeth, pitch_radius, tooth_height, tooth_top_width):
center_x, center_y = center
pitch_angle = 2 * math.pi / num_teeth  # Calculate the angle between each tooth

# Loop through each tooth and draw it
for i in range(num_teeth):
    angle = i * pitch_angle
    # Calculate positions of the tooth's base and top corners
    x1, y1 = pitch_radius * math.cos(angle) + center_x, pitch_radius *
    ↪   math.sin(angle) + center_y
    x2, y2 = pitch_radius * math.cos(angle + pitch_angle) + center_x, pitch_radius
    ↪   * math.sin(angle + pitch_angle) + center_y
    x3, y3 = (pitch_radius + tooth_height) * math.cos(angle + (pitch_angle -
    ↪   tooth_top_width / pitch_radius) / 2) + center_x, \
             (pitch_radius + tooth_height) * math.sin(angle + (pitch_angle -
             ↪   tooth_top_width / pitch_radius) / 2) + center_y
    x4, y4 = (pitch_radius + tooth_height) * math.cos(angle + (pitch_angle +
    ↪   tooth_top_width / pitch_radius) / 2) + center_x, \
             (pitch_radius + tooth_height) * math.sin(angle + (pitch_angle +
             ↪   tooth_top_width / pitch_radius) / 2) + center_y

    # Draw the sides of each tooth
    draw.line([(int(x1), int(y1)), (int(x3), int(y3))], fill='black', width=2)
    draw.line([(int(x3), int(y3)), (int(x4), int(y4))], fill='black', width=2)
        draw.line([(int(x4), int(y4)), (int(x2), int(y2))], fill='black', width=2)

# Parameters for drawing the gear
num_teeth = 30  # Number of teeth on the gear
pitch_radius = 245  # Distance from the center to the base of the teeth
tooth_height = 25  # Height of each tooth
tooth_top_width = 20  # Width of each tooth at the top

# Draw the gear at the specified center
draw_gear(draw, center, num_teeth, pitch_radius, tooth_height, tooth_top_width)

######################

# Display the image
img.show()
```

Result

The resulting image of the recreated IITK logo is shown below.

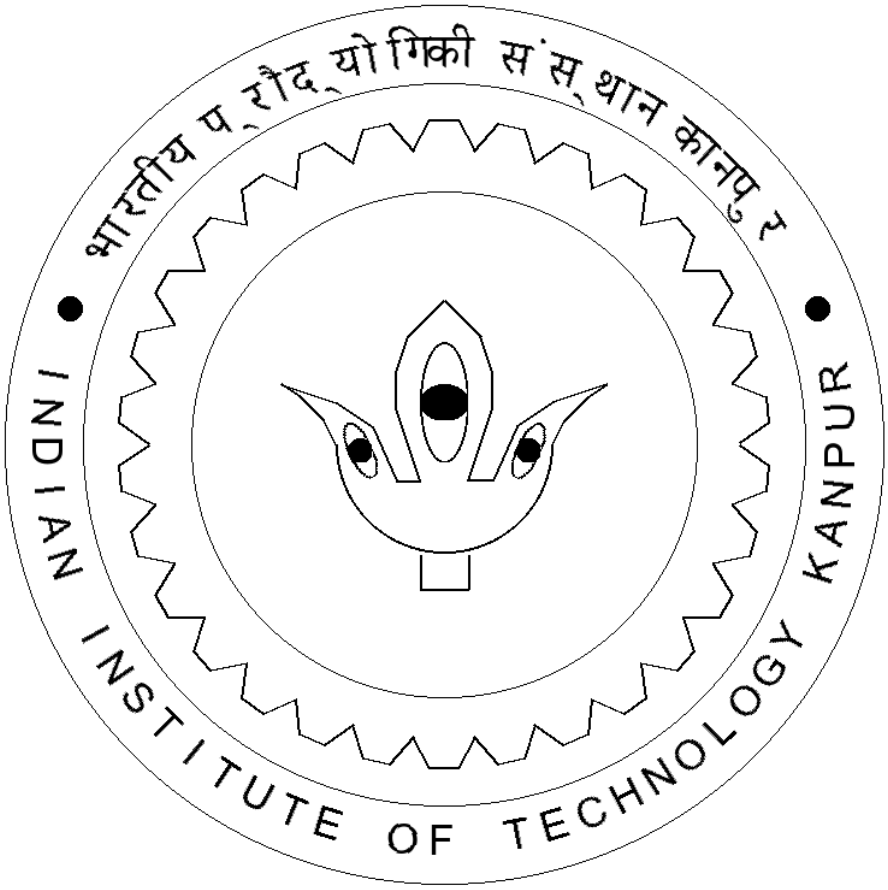
भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Figure 2: Recreated IITK Logo