Republic of the Philippines
**Laguna State Polytechnic University**
Province of Laguna

| Machine Problem No. 2 | | | |
|---|---|---|---|
| **Topic:** | **Topic 1.2: Image Processing Techniques** | **Week No.** | 3-5 |
| **Course Code:** | **CSST106** | **Term:** | 1st Semester |
| **Course Title:** | **Perception and Computer Vision** | **Academic Year:** | 2024-2025 |
| **Student Name** | Maxyne Nuela Ignacio | **Section** | |
| **Due date** | **September 21, 2024** | **Points** | |

## DOCUMENTATION REPORT

**Machine Problem No. 2: Applying Image Processing Techniques**

**Objective:**
Understand and apply various image processing techniques, including image transformations and filtering, using tools like OpenCV. Gain hands-on experience in implementing these techniques and solving common image processing tasks.

**Instructions:**
**Research and Comprehend:**

- **Lecture on Image Processing Concepts:**
  - Attend the lecture on image processing to gain a thorough understanding of the core concepts, including image transformations like scaling and rotation, as well as filtering techniques such as blurring and edge detection.

**Hands-On Exploration:**

- **Lab Session 1: Image Transformations**
  - **Scaling and Rotation:** Learn how to apply scaling and rotation transformations to images using OpenCV.
  - **Implementation:** Practice these transformations on sample images provided in the lab.
- **Lab Session 2: Filtering Techniques**
  - **Blurring and Edge Detection:** Explore how to apply blurring filters and edge detection algorithms to images using OpenCV.
  - **Implementation:** Apply these filters to sample images to understand their effects.

**Assignment:**

- **Implementing Image Transformations and Filtering:**
  - Choose a set of images and apply the techniques you've learned, including scaling, rotation, blurring, and edge detection.
  - **Documentation:** Document the steps taken, and the results achieved in a report.

**Problem-Solving Session:**

- **Common Image Processing Tasks:**
  - Engage in a problem-solving session focused on common challenges encountered in image processing tasks.
  - **Scenario-Based Problems:** Solve scenarios where you must choose and apply appropriate image processing techniques.

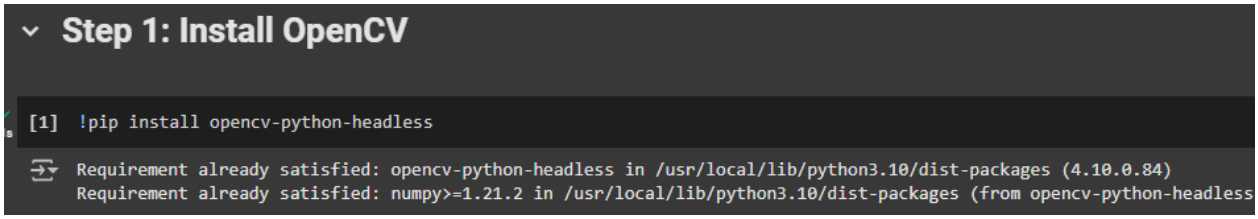
**Common Challenges Encountered in Image Processing Task that can be solved through Edge Detection and Filtering Techniques:**

- **Distinguishing Key Facial Features** - Sometimes, a face recognition system needs help finding essential features like the eyes, nose, or mouth. Edge detection can make these features stand out by outlining them more clearly, making it easier for the system to "see" and identify faces, especially in complex backgrounds.
- **Reducing Image Noise for Clarity -** Photos or video frames often contain "noise", random bits of blur or grain that make it harder to recognize faces accurately. Filtering techniques act like a softener for the image, reducing this noise without blurring important facial details. This helps make faces clearer and easier for the system to recognize.
- **Handling Different Lighting Conditions** - Lighting changes can make a face look different, with some areas too bright or too shadowed. Edge detection and filtering adjust the image's brightness and contrast, helping balance these lighting differences. This way, the system can still recognize a face even if it's in shadow or under a strong light.

**Hands-On Exploration**

**Lab Session 1: Image Transformations (SCALING AND ROTATION)**

**STEP 1: INSTALL REQUIRED LIBRARIES**




This package provides OpenCV functionality for image processing, minus the GUI features (useful for server environments or applications where display windows aren't needed).

**Step 2: Import Libraries**

## Step 2: Import Libraries

```
[2] import cv2
    import numpy as np
    import matplotlib.pyplot as plt
    from google.colab import files
    from io import BytesIO
    from PIL import Image
```

- cv2 enables image and video processing with OpenCV functions
- numpy provides tools for efficient numerical operations, ideal for handling arrays
- matplotlib.pyplot allows image and data visualization in various formats
- files from google.colab supports file uploads and downloads directly in Colab
- BytesIO and PIL Image manage image loading, saving, and in-memory file handling

## Step 3: Display Image

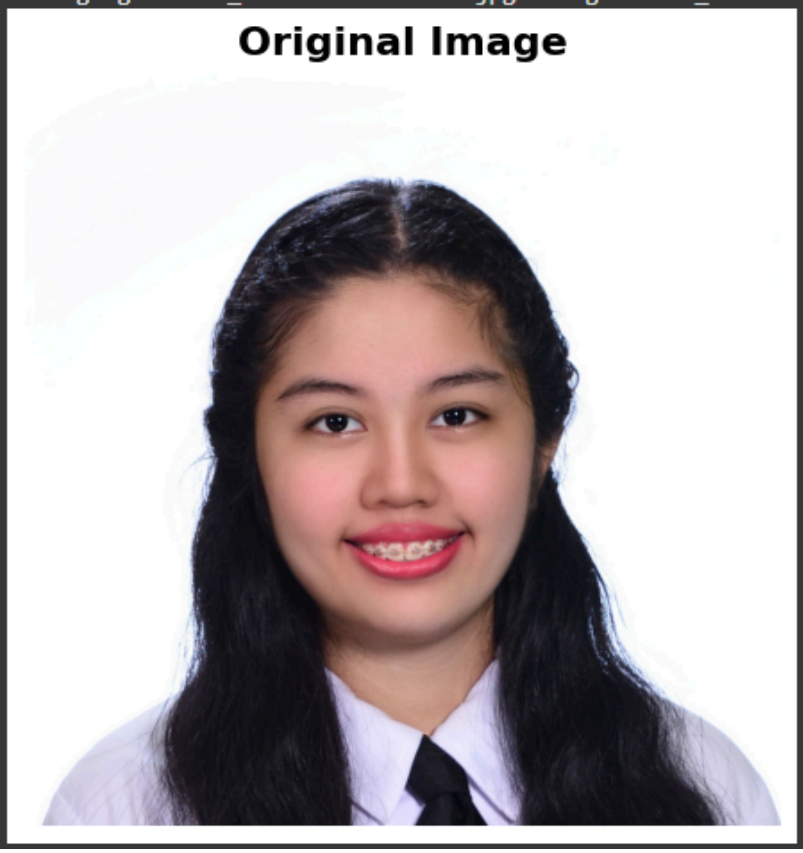
```
[3] def display_image(img,title="Image", figsize=(12,6)):
        plt.figure(figsize=figsize)
        plt.subplot(1,2,1)
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title(title, fontsize=16, fontweight='bold')
        plt.axis("off")
        plt.show()

    def display_image_gray(img1,img2, title1="Image 1",  title2="Image 2", figsize=(12,6)):
        plt.figure(figsize=figsize)
        plt.subplot(1,2,1)
        plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
        plt.title(title1, fontsize=16, fontweight='bold')
        plt.axis("off")


        plt.subplot(1,2,2)
        plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
        plt.title(title2, fontsize=16, fontweight='bold')
        plt.axis("off")

        plt.show()
```

- display_image takes a single image, converts it to RGB, and displays it with a title.
- display_image_gray takes two images, converts them to RGB, and displays them side-by-side with separate titles for each image.

## Step 4: Load Image

```
upload = files.upload()

image_path = next(iter(upload))
image = Image.open(BytesIO(upload[image_path]))
image = cv2.cvtColor(np.array(image),cv2.COLOR_RGB2BGR)

display_image(image, "Original Image")
```

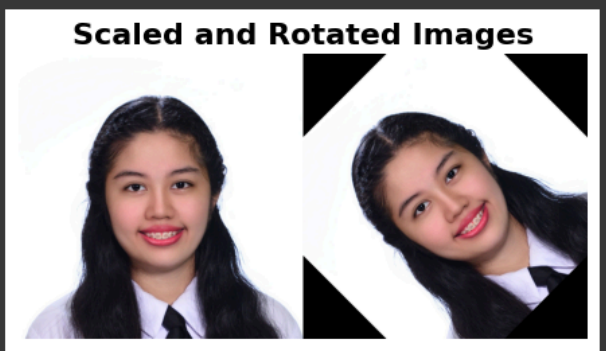
```
Choose Files IgnacioMN... - Auditor.jpg
• IgnacioMN_BSCS1B - Auditor.jpg(image/jpeg) - 504393 bytes, last modified: 6/9/2022 - 100% done
Saving IgnacioMN_BSCS1B - Auditor.jpg to IgnacioMN_BSCS1B - Auditor.jpg
```




- Start by uploading a file for processing
- Extract the filename of the uploaded image
- Open and load the image in RGB format, then convert it for OpenCV compatibility
- Display the uploaded image with a title for easy reference

**Step 5: Scaling and Rotation**

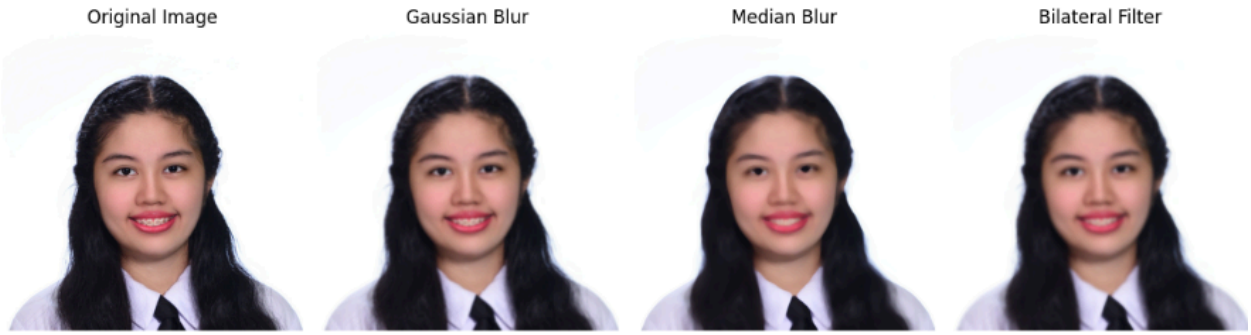
```
[5]  # Function to scale the image
     def scale_image(img, scale_factor):
         height, width = img.shape[:2]
         scale_img = cv2.resize(img, (int(width * scale_factor), int(height * scale_factor)), interpolation=cv2.INTER_LI
         return scale_img

     # Function to rotate the image
     def rotate_image(img, angle):
         height, width = img.shape[:2]
         center = (width / 2, height / 2)
         matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
         rotated_img = cv2.warpAffine(img, matrix, (width, height))
         return rotated_img

     # Apply scaling and rotation
     scaled_image = scale_image(image, 0.5)
     rotated_image = rotate_image(image, 45)

     # Resize images to have the same height
     scaled_image_resized = cv2.resize(scaled_image, (rotated_image.shape[1], rotated_image.shape[0]))

     # Combine the images side by side
     side_by_side_image = cv2.hconcat([scaled_image_resized, rotated_image])

     # Display the combined image
     display_image(side_by_side_image, "Scaled and Rotated Images")
```




- Defined a function to scale an image by a given factor
- Defined a function to rotate an image by a specified angle
- Applied both scaling and rotation transformations to the uploaded image
- Resized the scaled image to match the rotated image's height for alignment
- Displayed the scaled and rotated images side by side

**Step 6: Apply Filtering Techniques**

```
[6]  image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert from BGR to RGB for matplotlib

     # Split the image into R, G, B channels to apply blur separately
     r, g, b = cv2.split(image_rgb)

     # Apply stronger blurring techniques on each channel separately
     gaussian_blur_r = cv2.GaussianBlur(r, (35, 35), 0)
     gaussian_blur_g = cv2.GaussianBlur(g, (35, 35), 0)
     gaussian_blur_b = cv2.GaussianBlur(b, (35, 35), 0)
     gaussian_blur = cv2.merge([gaussian_blur_r, gaussian_blur_g, gaussian_blur_b])

     median_blur_r = cv2.medianBlur(r, 35)
     median_blur_g = cv2.medianBlur(g, 35)
     median_blur_b = cv2.medianBlur(b, 35)
     median_blur = cv2.merge([median_blur_r, median_blur_g, median_blur_b])

     bilateral_filter_r = cv2.bilateralFilter(r, 35, 150, 150)
     bilateral_filter_g = cv2.bilateralFilter(g, 35, 150, 150)
     bilateral_filter_b = cv2.bilateralFilter(b, 35, 150, 150)
     bilateral_filter = cv2.merge([bilateral_filter_r, bilateral_filter_g, bilateral_filter_b])

     # Prepare list of images and their labels
     images = [image_rgb, gaussian_blur, median_blur, bilateral_filter]
     labels = ['Original Image', 'Gaussian Blur', 'Median Blur', 'Bilateral Filter']

     # Set up matplotlib figure
     plt.figure(figsize=(12, 6))
     for i in range(len(images)):
         plt.subplot(1, len(images), i + 1)
         plt.imshow(images[i])
         plt.title(labels[i])
         plt.axis('off')

     # Show the combined image with labels
     plt.tight_layout()
     plt.show()
```




- Converted the original image from BGR to RGB for accurate color display in Matplotlib
- Split the image into red, green, and blue channels for applying blur to each separately
- Applied Gaussian blur, median blur, and bilateral filter to each color channel
- Merged the blurred channels back into single images for each type of blur
- Displayed the original and blurred images side by side with labels

**Step 6: Edge Detection**

```python
if image is None:
    print("Error loading image")
else:
    grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale

    # Adjusted edge detection parameters
    # Apply edge detection techniques with fine-tuned parameters
    canny_edges = cv2.Canny(grayscale_image, 50, 150)  # Lower thresholds for Canny

    # Sobel Edge Detection in both X and Y directions
    sobel_x = cv2.Sobel(grayscale_image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(grayscale_image, cv2.CV_64F, 0, 1, ksize=3)

    # Combining Sobel X and Y into a single output for better edge representation
    sobel_combined = cv2.magnitude(sobel_x, sobel_y)
    sobel_combined = cv2.convertScaleAbs(sobel_combined)  # Convert to 8-bit for display

    # Laplacian Edge Detection
    laplacian = cv2.Laplacian(grayscale_image, cv2.CV_64F, ksize=3)
    laplacian = cv2.convertScaleAbs(laplacian)  # Convert to 8-bit for display

    # Prepare list of images and their labels
    images = [grayscale_image, canny_edges, sobel_combined, laplacian]
    labels = ['Original Image (Grayscale)', 'Canny Edges', 'Sobel Combined', 'Laplacian']

    # Set up matplotlib figure
    plt.figure(figsize=(12, 6))
    for i in range(len(images)):
        plt.subplot(1, len(images), i + 1)
        plt.imshow(images[i], cmap='gray')
        plt.title(labels[i])
        plt.axis('off')

    # Show the combined image with labels
    plt.tight_layout()
    plt.show()
```


- Checked if the image loaded successfully to avoid errors
- Converted the image to grayscale for accurate edge detection
- Applied Canny edge detection with adjusted threshold values for finer edges
- Performed Sobel edge detection in X and Y directions and combined results for comprehensive edges
- Used Laplacian edge detection and converted to 8-bit for easier visualization
- Displayed all edge-detection results alongside the original grayscale image for comparison

**Lab Work Submission:**

- **Submitting Processed Images:**
    - Submit the processed images along with your code and documentation to your GitHub repository.
    - Ensure that your repository is well-organized and that all files are correctly labeled and documented.

**Submission Instruction:**

- Create Folder under the Github Repository of the subject.
- **Submission Format:**
    - Upload your processed images, code, and documentation to the GitHub repository.
    - Ensure all content is well-organized and clearly labeled.
- **Filename Format:** [SECTION-BERNARDINO-MP2] 4D-BERNARDINO-MP2
- **Penalties:** Failure to follow these instructions will result in a 5-point deduction for incorrect filename format and a 5-point deduction per day for late submission. Cheating and plagiarism will be penalized.

**Rubric for Machine Problem No. 2: Applying Image Processing Techniques**

| Criteria | Excellent (10 points) | Good (8 points) | Fair (5 points) | Poor (2 points) |
|---|---|---|---|---|
| **Understanding of Image Processing Concepts** | Comprehensive understanding of image processing techniques. | Good understanding with minor gaps. | Basic understanding with some inaccuracies. | Poor or incomplete understanding. |
| **Application of Transformations and Filters** | Correct and effective application of image transformations and filters. | Mostly correct with minor inaccuracies. | Basic application with significant errors. | Inaccurate or ineffective application. |
| **Problem-Solving Ability** | Effective solutions to common image processing tasks. | Adequate solutions with some errors. | Limited solutions with many errors. | Poor or incorrect solutions. |
| **Lab Work Submission** | Well-organized, clear, and complete submission. | Organized with minor issues in clarity. | Somewhat organized but lacks clarity. | Disorganized and unclear submission. |