**Exam: Computer Vision Project**

**Total Marks: 20  
 Time: 72 Hours (3 days)**

**Objective:**  
The aim of this project is to implement and analyze various edge detection techniques on images using OpenCV. Edge detection is a crucial step in many computer vision tasks like object detection, segmentation, and feature extraction. This project will explore different edge detection algorithms, including Sobel, Laplacian, and Canny, and evaluate their performance.

**Tools and Libraries:**

* **Programming Language:** Python
* **Core Library:** OpenCV
* **Supporting Libraries:** NumPy, Matplotlib (for visualizations)

**Project Outline:**

The project is divided into key stages:

1. **Image Acquisition and Preprocessing**
2. **Edge Detection Techniques**
3. **Performance Analysis and Optimization**
4. **Edge-Based Image Applications**

Each stage will explore different methods and how edge detection can be applied to enhance image understanding.

**Part A: Image Acquisition and Preprocessing**

**Task 1: Acquiring Sample Images**

* **Objective:** Obtain a set of images for testing edge detection methods. Images can be sourced from natural scenes, industrial objects, or geometric shapes to highlight various edge properties.
* **Steps:**
  + Load an image using OpenCV (cv2.imread()).
  + Display the image using OpenCV’s imshow() or Matplotlib.
* **Deliverable:** Present the original images, highlighting different types of edges (e.g., sharp, soft, complex).

**Task 2: Grayscale Conversion**

* **Objective:** Convert the color images into grayscale to simplify edge detection by focusing on intensity changes.
* **Steps:**
  + Convert the image to grayscale using OpenCV (cv2.cvtColor()).
  + Explain why grayscale is preferred for edge detection.
* **Deliverable:** Provide the code for grayscale conversion and display the grayscale image.

**Task 3: Noise Reduction Using Smoothing**

* **Objective:** Apply noise reduction techniques to smooth the image and improve edge detection accuracy.
* **Steps:**
  + Use a Gaussian blur or median filter to remove noise without significantly altering the edges.
  + Compare the original image with the smoothed image.
* **Deliverable:** Submit the code and display the images before and after applying the smoothing filters.

**Part B: Edge Detection Techniques**

**Task 4: Sobel Edge Detection**

* **Objective:** Detect edges using the Sobel operator, which calculates the gradient in both the x and y directions.
* **Steps:**
  + Apply Sobel filters for both horizontal and vertical edges.
  + Combine the gradients to get a complete edge map.
  + Experiment with different kernel sizes and compare the results.
* **Deliverable:** Provide the code, display the edge-detected image, and explain how the Sobel operator works.

**Task 5: Laplacian Edge Detection**

* **Objective:** Detect edges using the Laplacian operator, which calculates the second-order derivatives of the image.
* **Steps:**
  + Apply the Laplacian filter to the image to highlight regions of rapid intensity change.
  + Compare the Laplacian results with the gradient-based methods (Sobel).
* **Deliverable:** Provide the code and display the Laplacian-detected edges.

**Task 7: Canny Edge Detection**

* **Objective:** Implement the Canny edge detection algorithm, which is a multi-stage process for detecting strong and weak edges.
* **Steps:**
  + Apply the Canny edge detector using OpenCV (cv2.Canny()).
  + Experiment with different threshold values for edge detection and explore how they impact the results.
  + Compare Canny with Sobel, and Laplacian in terms of accuracy and performance.
* **Deliverable:** Submit the code and display the Canny edge-detected image, explaining why it is considered a robust edge detection method.

**Part C: Performance Analysis and Optimization**

**Task 8: Comparing Edge Detection Techniques**

* **Objective:** Compare and evaluate the performance of the different edge detection methods implemented earlier.
* **Steps:**
  + Compare the Sobel, Laplacian, and Canny algorithms on different images (e.g., high-contrast images, noisy images, and smooth edges).
  + Evaluate their performance based on edge sharpness, accuracy, noise sensitivity, and computational efficiency.
  + Create a table or chart summarizing the pros and cons of each method.
* **Deliverable:** Submit the comparison report, with visual examples and a summary of findings for each algorithm.

**Final Report:**

Your final report should include:

1. **Introduction** – An overview of the edge detection problem and its applications in computer vision.
2. **Methods** – A detailed explanation of each edge detection technique with code snippets.
3. **Results** – Examples of the edge-detected images for each method and analysis of results.
4. **Challenges** – Discussion of the difficulties faced and how you addressed them.
5. **Conclusion** – Summary of findings and suggestions for improving edge detection techniques in future work.

**Grading Criteria:**

* **Code Functionality (8 marks)**: Correctness and performance of the code.
* **Report Quality (6 marks)**: Clarity of explanation and documentation.
* **Creativity and Problem-Solving (4 marks)**: Innovative solutions and optimizations in edge detection.
* **Presentation (2 marks)**: Quality of visuals and overall report structure.

**End of Project**

Top of Form