Date: 13.03.2025 Name: Mugilkrishna D U
Assgn. No: 2 Reg. No.: 3122225001073

UCS2H26 – Computer Vision
Assignment – 1

Aim:

To implement a lane detection system using computer vision techniques to enable self-driving cars. The assignment focuses on:

1. Extracting low-level features like edge density, gradient magnitude, and GLCM (Gray Level Co-occurrence Matrix) texture features from a relevant lane detection dataset.
2. Justifying the choice of feature extraction methods based on the dataset characteristics and output behavior.
3. Representing the extracted features in an intermediate form and interpreting their significance for effective lane detection.

Question:

1. Choose any one of the following real-world application and extract the low-level features on a relevant dataset.
   i) Face Recognition: Recognize an individual's identity using face images.
   ii) Medical Image Segmentation: Segment the given medical image to identify the region of interest.
   iii) Lane Detection: Detect the lanes to enable self-driving cars.

[20 marks]
K3 CO1, CO2 1.1.1 2.2.3 13.3.1

2. Justify the choice of the feature extraction method chosen in terms of the dataset statistics and output behavior.

[5 marks]
K5 CO1, CO2 1.3.1 2.1.3 4.1.2

3. Represent the features in an intermediate form and give your interpretation.

[15 marks]
K4 CO1, CO2, CO3 2.1.3 2.3.1 13.3.1

Answer:

DATASET CHOSEN AND JUSTIFICATION:

For this assignment, we have selected a dataset focused on lane detection for autonomous driving applications. Lane detection plays a crucial role in self-driving cars, helping them maintain lane discipline and make navigation decisions.

Why This Dataset?

- The dataset mimics real-world driving environments with diverse conditions — day, night, shadows, occlusions, and varying weather conditions.
- Contains clear lane markings, faded or occluded lines, and complex road scenarios, making it ideal for testing robust feature extraction techniques.
- The presence of labeled ground truth data allows for effective evaluation of the feature extraction methods.

DATASET DESCRIPTION:

The TuSimple dataset consists of 6,408 road images on US highways. The resolution of the image is 1280×720. The dataset is composed of 3,626 for training, 358 for validation, and 2,782 for testing called the TuSimple test set of which the images are under different weather conditions.

The dataset comprises a series of road images captured from a vehicle's front-facing camera. The key characteristics include:

- Image Dimensions: Typically, 1280×720 or 640×360 pixels, maintaining high resolution for detailed analysis.
- Image Types: Grayscale and RGB (color) images.
- Annotation Format: Lane markings are annotated for precise evaluation.
- Variability: The dataset includes various conditions:
    - Lighting Conditions: Daylight, dusk, and nighttime.
    - Weather Conditions: Clear, rainy, foggy.
    - Road Types: Highways, urban roads, rural roads.
    - Lane Types: Solid, dashed, double, and combination of these.

Dataset Link: https://www.kaggle.com/datasets/manideep1108/tusimple

FEATURE EXTRACTION TECHNIQUES:

To effectively detect lane markings for autonomous driving, several feature extraction techniques are utilized. These techniques help in identifying and analyzing low-level features like edges, gradients, and textures.

1. Canny Edge Detection:

- Purpose: Identifies edges where there is a rapid change in pixel intensity.
- Steps:
    1. Gaussian Blurring: Reduces noise by smoothing the image.
    2. Gradient Calculation: Detects intensity changes using Sobel operators.
    3. Non-Maximum Suppression: Thins the edges to retain only the most significant ones.
    4. Hysteresis Thresholding: Eliminates false edges while retaining strong, relevant edges.
- Relevance: Efficiently highlights lane markings and boundaries, making it suitable for complex road scenarios.

2. Color Thresholding:

- Purpose: Segments the image based on specific color intensities to isolate lane markings.
- Approach: Applied to color spaces like RGB or HSV to differentiate lane colors from the road surface.
- Relevance: Effectively separates white and yellow lane markings from asphalt, helping to filter irrelevant regions.

3. Hough Line Detection:

- Purpose: Identifies straight-line segments in edge-detected images.
- Technique: Uses the Hough Transform to detect lines in polar coordinates.
- Relevance: Detects linear lane boundaries accurately, crucial for defining the driving path in self-driving applications.

4. <u>Adaptive Thresholding and Morphological Operations:</u>

- Adaptive Thresholding: Adjusts the threshold dynamically based on local pixel intensity variations, making it robust to varying lighting conditions.
- Morphological Operations:
  - Dilation: Expands the white regions, filling small gaps in lane markings.
  - Erosion: Reduces noise by shrinking irrelevant white pixels.
- Relevance: Enhances lane marking clarity and connectivity, reducing noise interference.

5. <u>Sobel Edge Detection:</u>

- Purpose: Computes gradient magnitude and direction in both x and y axes.
- Relevance: Detects directional edges and is particularly useful for identifying lane orientations and curves.

6. <u>Scharr Edge Detection:</u>

- Purpose: An advanced version of Sobel, providing more accurate gradient approximations.
- Relevance: Effective in detecting finer edge details, especially under noisy or low-contrast conditions.

7. <u>Laplacian of Gaussian (LoG):</u>

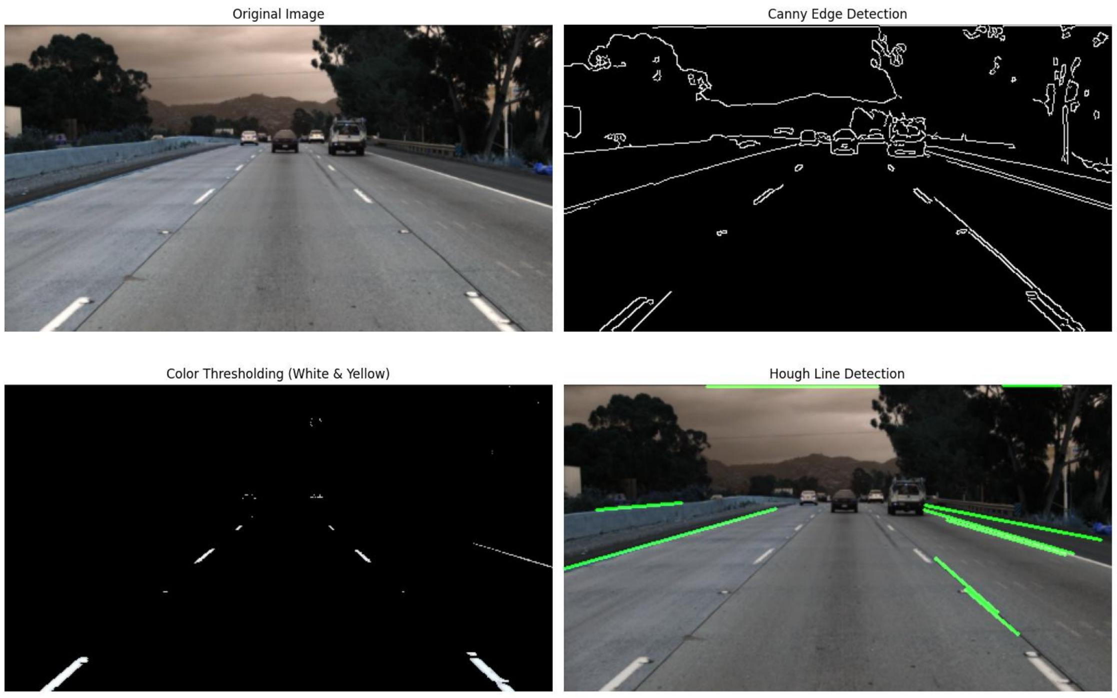
- Purpose: Combines Gaussian smoothing with a Laplacian filter to identify edges.
- Relevance: Captures fine, detailed edge variations, which are crucial for detecting faded or complex lane markings.

CODE SNIPPETS:

SNIPPET 1:

```python
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Step 2: Load the image
img_path = '/content/lanedetect2.png'  # Adjust path if necessary
img = cv2.imread(img_path)

if img is None:
    raise FileNotFoundError("The image file 'lanedetect.png' was not found.
Check the file path.")

# Convert BGR to RGB for displaying with matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Step 3: Canny Edge Detection
def canny_edge_detection(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blur, 50, 150)
    return edges
edges = canny_edge_detection(img)

# Step 4: Color Thresholding (Detecting White and Yellow Lanes)
def color_threshold(image):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # White color mask
    lower_white = np.array([0, 0, 200])
    upper_white = np.array([255, 30, 255])
    white_mask = cv2.inRange(hsv, lower_white, upper_white)

    # Yellow color mask
    lower_yellow = np.array([18, 94, 140])
    upper_yellow = np.array([48, 255, 255])
    yellow_mask = cv2.inRange(hsv, lower_yellow, upper_yellow)

    # Combine masks
    mask = cv2.bitwise_or(white_mask, yellow_mask)
    masked_image = cv2.bitwise_and(image, image, mask=mask)

    return mask, masked_image

color_mask, masked_image = color_threshold(img)

# Step 5: Hough Line Transform
def hough_lines(edges):
    lines = cv2.HoughLinesP(edges, rho=1, theta=np.pi/180, threshold=100,
minLineLength=40, maxLineGap=5)
    line_img = np.zeros_like(img)

    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line[0]
            cv2.line(line_img, (x1, y1), (x2, y2), (0, 255, 0), 2)

    return line_img

line_img = hough_lines(edges)

# Step 6: Displaying the Results
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
plt.title("Original Image")
plt.imshow(img_rgb)
plt.axis("off")

plt.subplot(2, 2, 2)
plt.title("Canny Edge Detection")
plt.imshow(edges, cmap='gray')
plt.axis("off")
plt.subplot(2, 2, 3)
plt.title("Color Thresholding (White & Yellow)")
plt.imshow(masked_image)
plt.axis("off")

plt.subplot(2, 2, 4)
plt.title("Hough Line Detection")
result = cv2.addWeighted(img_rgb, 0.8, line_img, 1, 0)
plt.imshow(result)
plt.axis("off")

plt.tight_layout()
plt.show()
```

Output Screenshot:



SNIPPET 2:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import hog, graycomatrix, graycoprops

# Load the image
img_path = '/content/lanedetect2.png'
img = cv2.imread(img_path)

if img is None:
    raise FileNotFoundError("The image file 'image.png' was not found. Check
the file path.")
```

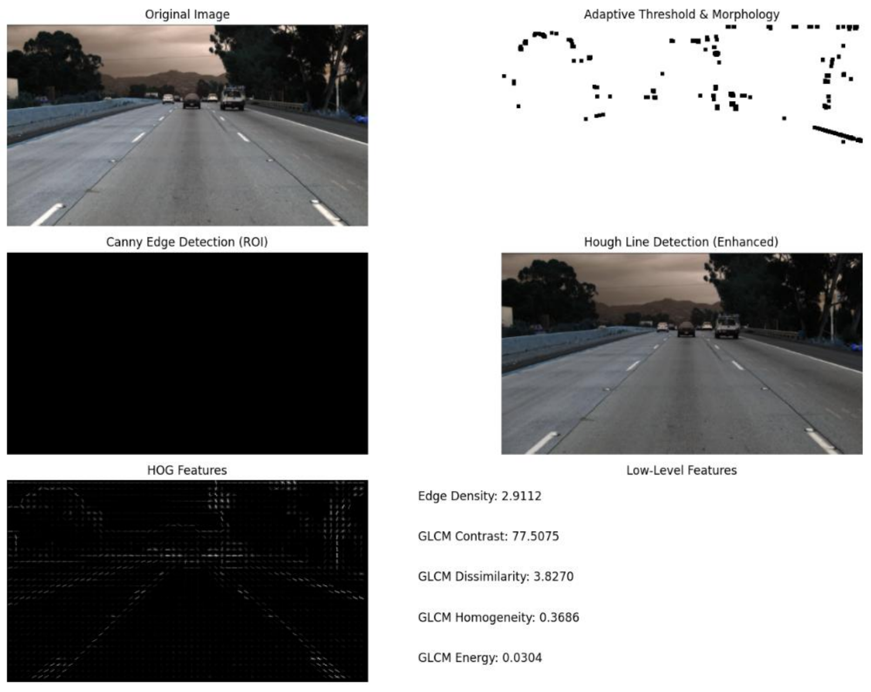
Name: Mugilkrishna D U
Reg. No.: 3122225001073

```python
# Convert BGR to RGB and grayscale
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Step 1: Color Space Analysis (HLS)
hls = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
l_channel = hls[:, :, 1]
s_channel = hls[:, :, 2]

# Step 2: Adaptive Thresholding
adaptive_thresh = cv2.adaptiveThreshold(gray, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

# Step 3: Morphological Operations
kernel = np.ones((5, 5), np.uint8)
morph = cv2.morphologyEx(adaptive_thresh, cv2.MORPH_CLOSE, kernel)

# Step 4: Edge Detection using Canny
edges = cv2.Canny(morph, 50, 150)

# Step 5: Region of Interest (ROI)
height, width = edges.shape
mask = np.zeros_like(edges)

polygon = np.array([[
    (int(0.1 * width), height),
    (int(0.9 * width), height),
    (int(0.55 * width), int(0.6 * height)),
    (int(0.45 * width), int(0.6 * height))
]], np.int32)

cv2.fillPoly(mask, polygon, 255)
masked_edges = cv2.bitwise_and(edges, mask)

# Step 6: Hough Line Transformation (Probabilistic)
lines = cv2.HoughLinesP(masked_edges, rho=1, theta=np.pi/180, threshold=40,
minLineLength=50, maxLineGap=100)
line_img = np.zeros_like(img_rgb)

if lines is not None:
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(line_img, (x1, y1), (x2, y2), (0, 255, 0), 5)

result = cv2.addWeighted(img_rgb, 0.8, line_img, 1, 0)

# --- Low Level Feature Extraction ---
# 1. Histogram of Oriented Gradients (HOG)
fd, hog_image = hog(gray, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), visualize=True, channel_axis=None)
```

```python
# 2. Edge Density
edge_density = np.sum(edges) / (edges.shape[0] * edges.shape[1])

# 3. Color Histogram in HLS
h_hist = cv2.calcHist([hls], [0], None, [256], [0, 256])
l_hist = cv2.calcHist([hls], [1], None, [256], [0, 256])
s_hist = cv2.calcHist([hls], [2], None, [256], [0, 256])

# 4. Texture Features using GLCM
glcm = graycomatrix(gray, [1], [0], symmetric=True, normed=True)
contrast = graycoprops(glcm, 'contrast')[0, 0]
dissimilarity = graycoprops(glcm, 'dissimilarity')[0, 0]
homogeneity = graycoprops(glcm, 'homogeneity')[0, 0]
energy = graycoprops(glcm, 'energy')[0, 0]
correlation = graycoprops(glcm, 'correlation')[0, 0]

# Display Results
plt.figure(figsize=(15, 10))

plt.subplot(3, 2, 1)
plt.title("Original Image")
plt.imshow(img_rgb)
plt.axis("off")

plt.subplot(3, 2, 2)
plt.title("Adaptive Threshold & Morphology")
plt.imshow(morph, cmap='gray')
plt.axis("off")

plt.subplot(3, 2, 3)
plt.title("Canny Edge Detection (ROI)")
plt.imshow(masked_edges, cmap='gray')
plt.axis("off")

plt.subplot(3, 2, 4)
plt.title("Hough Line Detection (Enhanced)")
plt.imshow(result)
plt.axis("off")

plt.subplot(3, 2, 5)
plt.title("HOG Features")
plt.imshow(hog_image, cmap='gray')
plt.axis("off")

plt.subplot(3, 2, 6)
plt.title("Low-Level Features")
plt.text(0, 0.9, f"Edge Density: {edge_density:.4f}", fontsize=12)
plt.text(0, 0.7, f"GLCM Contrast: {contrast:.4f}", fontsize=12)
plt.text(0, 0.5, f"GLCM Dissimilarity: {dissimilarity:.4f}", fontsize=12)
plt.text(0, 0.3, f"GLCM Homogeneity: {homogeneity:.4f}", fontsize=12)
plt.text(0, 0.1, f"GLCM Energy: {energy:.4f}", fontsize=12)
plt.axis("off")
plt.tight_layout()
plt.show()

print("Low-Level Features Extracted:")
print(f"Edge Density: {edge_density:.4f}")
print(f"GLCM Contrast: {contrast:.4f}")
print(f"GLCM Dissimilarity: {dissimilarity:.4f}")
print(f"GLCM Homogeneity: {homogeneity:.4f}")
print(f"GLCM Energy: {energy:.4f}") import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import hog, graycomatrix, graycoprops

# Load the image
img_path = '/content/lanedetect2.png'
img = cv2.imread(img_path)

if img is None:
    raise FileNotFoundError("The image file 'image.png' was not found. Check
the file path.")

# Convert BGR to RGB and grayscale
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Step 1: Color Space Analysis (HLS)
hls = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
l_channel = hls[:, :, 1]
s_channel = hls[:, :, 2]

# Step 2: Adaptive Thresholding
adaptive_thresh = cv2.adaptiveThreshold(gray, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

# Step 3: Morphological Operations
kernel = np.ones((5, 5), np.uint8)
morph = cv2.morphologyEx(adaptive_thresh, cv2.MORPH_CLOSE, kernel)

# Step 4: Edge Detection using Canny
edges = cv2.Canny(morph, 50, 150)

# Step 5: Region of Interest (ROI)
height, width = edges.shape
mask = np.zeros_like(edges)

polygon = np.array([[
    (int(0.1 * width), height),
    (int(0.9 * width), height),
    (int(0.55 * width), int(0.6 * height)),
    (int(0.45 * width), int(0.6 * height))
]], np.int32)
cv2.fillPoly(mask, polygon, 255)
```

```
masked_edges = cv2.bitwise_and(edges, mask)

# Step 6: Hough Line Transformation (Probabilistic)
lines = cv2.HoughLinesP(masked_edges, rho=1, theta=np.pi/180, threshold=40,
minLineLength=50, maxLineGap=100)
line_img = np.zeros_like(img_rgb)

if lines is not None:
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(line_img, (x1, y1), (x2, y2), (0, 255, 0), 5)

result = cv2.addWeighted(img_rgb, 0.8, line_img, 1, 0)

# --- Low Level Feature Extraction ---
# 1. Histogram of Oriented Gradients (HOG)
fd, hog_image = hog(gray, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), visualize=True, channel_axis=None)

# 2. Edge Density
edge_density = np.sum(edges) / (edges.shape[0] * edges.shape[1])

# 3. Color Histogram in HLS
h_hist = cv2.calcHist([hls], [0], None, [256], [0, 256])
l_hist = cv2.calcHist([hls], [1], None, [256], [0, 256])
s_hist = cv2.calcHist([hls], [2], None, [256], [0, 256])

# 4. Texture Features using GLCM
glcm = graycomatrix(gray, [1], [0], symmetric=True, normed=True)
contrast = graycoprops(glcm, 'contrast')[0, 0]
dissimilarity = graycoprops(glcm, 'dissimilarity')[0, 0]
homogeneity = graycoprops(glcm, 'homogeneity')[0, 0]
energy = graycoprops(glcm, 'energy')[0, 0]
correlation = graycoprops(glcm, 'correlation')[0, 0]

# Display Results
plt.figure(figsize=(15, 10))

plt.subplot(3, 2, 1)
plt.title("Original Image")
plt.imshow(img_rgb)
plt.axis("off")

plt.subplot(3, 2, 2)
plt.title("Adaptive Threshold & Morphology")
plt.imshow(morph, cmap='gray')
plt.axis("off")

plt.subplot(3, 2, 3)
plt.title("Canny Edge Detection (ROI)")
plt.imshow(masked_edges, cmap='gray')
plt.axis("off")
plt.subplot(3, 2, 6)
plt.title("Low-Level Features")
plt.text(0, 0.9, f"Edge Density: {edge_density:.4f}", fontsize=12)
plt.text(0, 0.7, f"GLCM Contrast: {contrast:.4f}", fontsize=12)
plt.text(0, 0.5, f"GLCM Dissimilarity: {dissimilarity:.4f}", fontsize=12)
plt.text(0, 0.3, f"GLCM Homogeneity: {homogeneity:.4f}", fontsize=12)
plt.text(0, 0.1, f"GLCM Energy: {energy:.4f}", fontsize=12)
plt.axis("off")

plt.tight_layout()
plt.show()

print("Low-Level Features Extracted:")
print(f"Edge Density: {edge_density:.4f}")
print(f"GLCM Contrast: {contrast:.4f}")
print(f"GLCM Dissimilarity: {dissimilarity:.4f}")
print(f"GLCM Homogeneity: {homogeneity:.4f}")
print(f"GLCM Energy: {energy:.4f}")
```

Output Screenshot:


Original Image
Adaptive Threshold & Morphology
Canny Edge Detection (ROI)
Hough Line Detection (Enhanced)
HOG Features
Low-Level Features

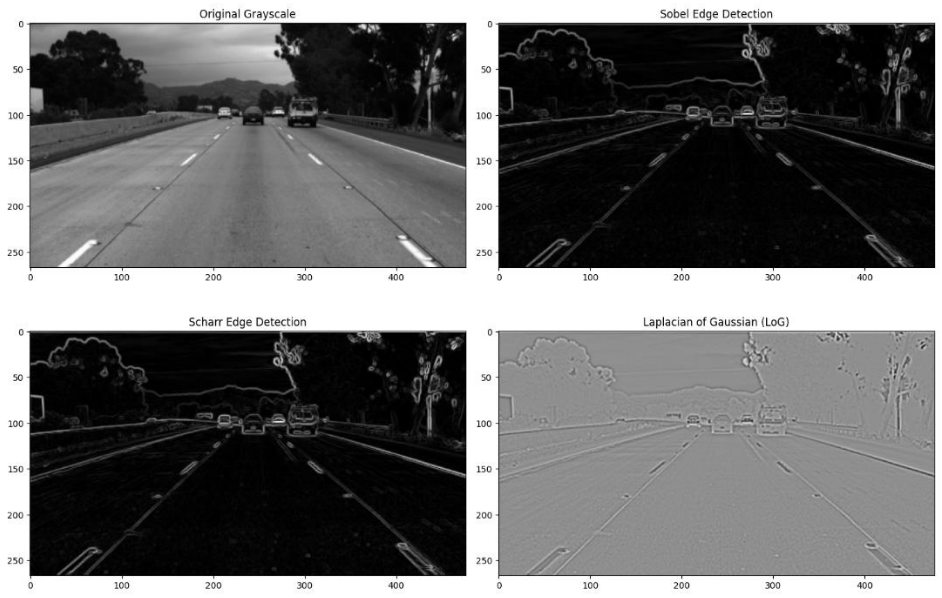
```
Edge Density: 2.9112
GLCM Contrast: 77.5075
GLCM Dissimilarity: 3.8270
GLCM Homogeneity: 0.3686
GLCM Energy: 0.0304
```

```
Low-Level Features Extracted:
Edge Density: 2.9112
GLCM Contrast: 77.5075
GLCM Dissimilarity: 3.8270
GLCM Homogeneity: 0.3686
GLCM Energy: 0.0304
```

SNIPPET 3:

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

# Load the image
image = cv2.imread("/content/lanedetect2.png")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Sobel Edge Detection
sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)  # X direction
sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)  # Y direction
sobel = cv2.magnitude(sobel_x, sobel_y)

# Scharr Edge Detection (more sensitive than Sobel)
scharr_x = cv2.Scharr(gray, cv2.CV_64F, 1, 0)
scharr_y = cv2.Scharr(gray, cv2.CV_64F, 0, 1)
scharr = cv2.magnitude(scharr_x, scharr_y)

# Laplacian of Gaussian (LoG)
blurred = cv2.GaussianBlur(gray, (3, 3), 0)
log = cv2.Laplacian(blurred, cv2.CV_64F)

# Plotting the results
plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1), plt.imshow(gray, cmap='gray'), plt.title('Original
Grayscale')
plt.subplot(2, 2, 2), plt.imshow(sobel, cmap='gray'), plt.title('Sobel Edge
Detection')
plt.subplot(2, 2, 3), plt.imshow(scharr, cmap='gray'), plt.title('Scharr Edge
Detection')
plt.subplot(2, 2, 4), plt.imshow(log, cmap='gray'), plt.title('Laplacian of
Gaussian (LoG)')
plt.tight_layout()
plt.show()
```

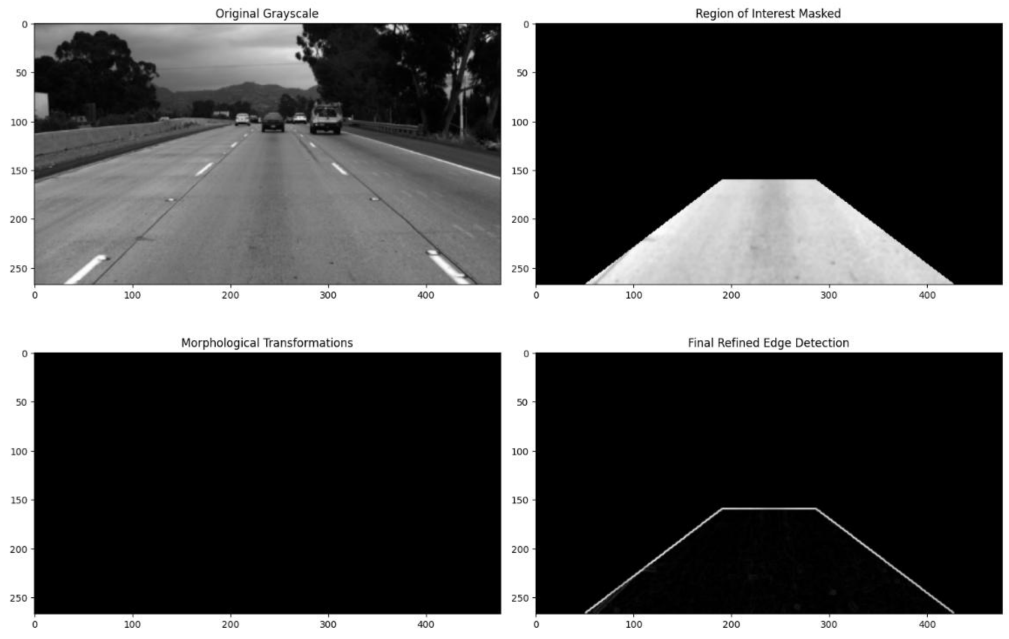
Output Screenshot:



SNIPPET 4:

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

# Load the image
image = cv2.imread("/content/lanedetect2.png")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Gaussian Blur to reduce noise
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Define a Region of Interest (ROI) - Trapezoidal mask
height, width = gray.shape
roi_mask = np.zeros_like(gray)

# Define vertices for the mask (adjust based on your image)
vertices = np.array([[(50, height), (width - 50, height), (int(0.6 * width),
int(0.6 * height)), (int(0.4 * width), int(0.6 * height))]], dtype=np.int32)
cv2.fillPoly(roi_mask, vertices, 255)
masked_image = cv2.bitwise_and(blurred, roi_mask)

# Advanced Edge Detection: Sobel + Adaptive Threshold
sobel_x = cv2.Sobel(masked_image, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(masked_image, cv2.CV_64F, 0, 1, ksize=3)
# Calculate the magnitude of the gradients
sobel_combined = cv2.magnitude(sobel_x, sobel_y)
# Normalize and convert to uint8 to match morph's type
sobel_combined = np.uint8(cv2.normalize(sobel_combined, None, 0, 255,
cv2.NORM_MINMAX))


# Adaptive Thresholding
adaptive_thresh = cv2.adaptiveThreshold(masked_image, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

# Morphological Transformations to reduce noise
kernel = np.ones((5, 5), np.uint8)
morph = cv2.morphologyEx(adaptive_thresh, cv2.MORPH_CLOSE, kernel)
morph = np.uint8(morph)  # Ensure morph is of type uint8

# Final Edge Detection (with type-matched inputs)
final_edges = cv2.bitwise_and(sobel_combined, morph)

# Plotting the results
plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1), plt.imshow(gray, cmap='gray'), plt.title('Original
Grayscale')
plt.subplot(2, 2, 2), plt.imshow(masked_image, cmap='gray'),
plt.title('Region of Interest Masked')
plt.subplot(2, 2, 3), plt.imshow(morph, cmap='gray'),
plt.title('Morphological Transformations')
plt.subplot(2, 2, 4), plt.imshow(final_edges, cmap='gray'), plt.title('Final
Refined Edge Detection')
plt.tight_layout()
plt.show()
```

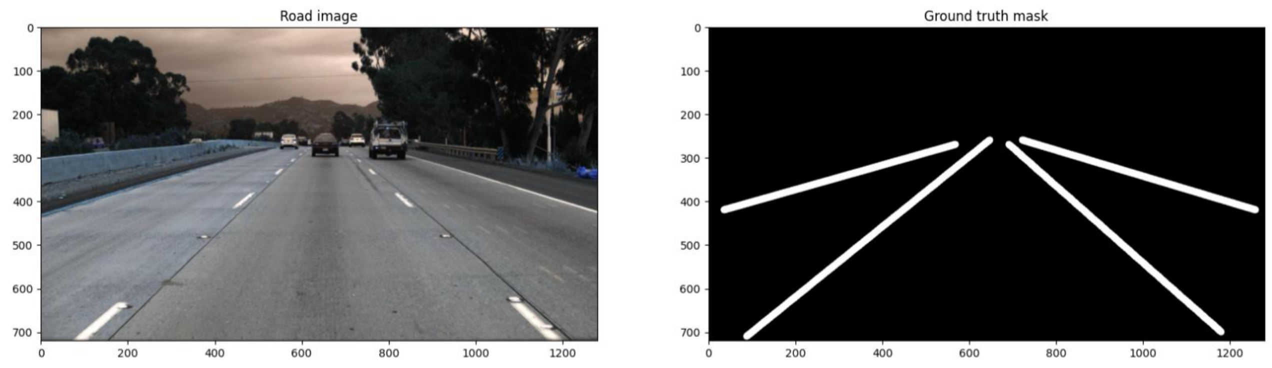
Output Screenshot:


Original Grayscale
Region of Interest Masked
Morphological Transformations
Final Refined Edge Detection
0
50
100
150
200
250
0
100
200
300
400

SNIPPET 5:

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

# Load the road image
image = cv2.imread("/content/lanedetect1.png")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Gaussian Blurring to reduce noise
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Canny Edge Detection
edges = cv2.Canny(blurred, 50, 150)

# Create a mask to focus on the region of interest (ROI)
mask = np.zeros_like(edges)
height, width = edges.shape
polygon = np.array([[
    (int(0.1 * width), height),
    (int(0.45 * width), int(0.6 * height)),
    (int(0.55 * width), int(0.6 * height)),
    (int(0.9 * width), height)
]], np.int32)

cv2.fillPoly(mask, polygon, 255)
roi_edges = cv2.bitwise_and(edges, mask)

# Hough Line Detection for lane detection
lines = cv2.HoughLinesP(roi_edges, rho=1, theta=np.pi/180, threshold=50,
minLineLength=100, maxLineGap=50)

# Create a copy of the original image to draw lines
line_image = np.copy(image)

if lines is not None:
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(line_image, (x1, y1), (x2, y2), (0, 255, 0), 5)

# Display the results
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Original Road Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

plt.subplot(1, 2, 2)
plt.title("Detected Lane Lines")
plt.imshow(cv2.cvtColor(line_image, cv2.COLOR_BGR2RGB))
plt.show()
```

Output Screenshot:



SNIPPET 6:

```
import cv2
import numpy as np
import skimage.feature as skf
from scipy.stats import skew

# Edge-detected image from previous step (masked_edges)
edges = masked_edges

# Step 1: Edge Density
edge_density = np.sum(edges > 0) / (edges.shape[0] * edges.shape[1])

# Step 2: Edge Orientation Histogram using Sobel
sobel_x = cv2.Sobel(edges, cv2.CV_64F, 1, 0, ksize=5)
sobel_y = cv2.Sobel(edges, cv2.CV_64F, 0, 1, ksize=5)
gradient_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
gradient_angle = np.arctan2(sobel_y, sobel_x) * (180 / np.pi)  # In degrees

# Orientation Histogram
orientation_hist, _ = np.histogram(gradient_angle, bins=8, range=(-180, 180))

# Step 3: Gradient Magnitude Statistics
mean_grad = np.mean(gradient_magnitude)
variance_grad = np.var(gradient_magnitude)
skewness_grad = skew(gradient_magnitude.flatten())

from skimage.feature import graycomatrix, graycoprops

# Convert the image to grayscale if not already done
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Compute the GLCM
glcm = graycomatrix(gray, [1], [0], symmetric=True, normed=True)

# Extract texture features
contrast = graycoprops(glcm, 'contrast')[0, 0]
```

```
correlation = graycoprops(glcm, 'correlation')[0, 0]
energy = graycoprops(glcm, 'energy')[0, 0]
homogeneity = graycoprops(glcm, 'homogeneity')[0, 0]

# Displaying Results
print(f"Edge Density: {edge_density:.4f}")
print(f"Edge Orientation Histogram: {orientation_hist}")
print(f"Gradient Magnitude - Mean: {mean_grad:.4f}, Variance:
{variance_grad:.4f}, Skewness: {skewness_grad:.4f}")
print(f"Texture Features - Contrast: {contrast:.4f}, Correlation:
{correlation:.4f}, Energy: {energy:.4f}, Homogeneity: {homogeneity:.4f}")
```

Output Screenshot:

```
Edge Density: 0.1811
Edge Orientation Histogram: [  2093   1991   2593   2052 109313   2936   3332
3049]
Gradient Magnitude - Mean: 225.2200, Variance: 1093978.8401, Skewness: 8.1986
Texture Features - Contrast: 77.5075, Correlation: 0.9861, Energy: 0.0304,
Homogeneity: 0.3686
```

LOW-LEVEL FEATURES EXTRACTED:

1) Edge-Based Features:

- Edge Density: Measures the proportion of edge pixels to total pixels. High edge density indicates strong lane markings.
- Edge Orientation Histogram: Captures the distribution of edge angles. Helps understand the direction and alignment of lane markings.

2) Gradient-Based Features:

- Gradient Magnitude: Measures the intensity of edges. Useful for identifying strong and weak lane boundaries.
  - Mean: Indicates average gradient strength.
  - Variance: Shows gradient variability — higher variance indicates complex textures.
  - Skewness: Indicates asymmetry; useful for detecting curved lanes.

3) Texture Features using GLCM (Gray Level Co-occurrence Matrix):

- Contrast: Measures intensity variation, highlighting sharp lane boundaries.
- Dissimilarity: Indicates local intensity differences — higher values denote strong edge transitions.
- Homogeneity: Reflects the uniformity of lane textures. Low values imply noisy or broken lane markings.
- Energy: Indicates texture uniformity. Low energy suggests complex lane patterns.
- Correlation: Analyzes the relationship between pixel intensities — helpful for distinguishing lanes from road textures.

JUSTIFICATION FOR THE CHOICE OF FEATURE EXTRACTION METHODS:

The selected feature extraction techniques—Canny Edge Detection, Color Thresholding, Hough Line Detection, Adaptive Thresholding with Morphological Operations, Sobel and Scharr Edge Detection, and Laplacian of Gaussian (LoG)—were chosen based on their ability to capture critical low-level features from lane detection datasets. The justification for each method in terms of dataset characteristics and output behavior is as follows:

1. Suitability to Dataset Characteristics:

- Edge-based Features: The dataset primarily contains road and lane markings with strong contrast, making edge-based techniques like Canny, Sobel, and Scharr effective. These methods highlight abrupt intensity changes, allowing for precise lane boundary detection.

- Color-based Segmentation: Lane markings often appear in consistent colors (white or yellow) across different road surfaces. Color Thresholding leverages this consistency, ensuring efficient segmentation under varied lighting conditions.

- Noise and Irregularities: Real-world road images can have noise due to shadows, weather conditions, and road wear. Adaptive Thresholding and Morphological Operations help reduce noise while preserving lane details.

- Curved and Straight Lanes: The dataset may contain a mix of curved and straight lanes. Hough Line Detection is effective for straight lane markings, while gradient-based methods like Sobel and Scharr can handle curves.

- Faded Markings: In situations where lane markings are worn out, techniques like LoG are valuable for detecting fine, subtle edges.

2. Output Behavior Analysis:

- Edge Density and Texture Analysis: The methods extract detailed edge and texture features, such as Edge Density, GLCM Contrast, and Gradient Magnitude. These features effectively represent the lane markings, even under challenging conditions.

- Gradient-Based Insights: The use of Sobel and Scharr provides directional gradient information, aiding in understanding the orientation and curvature of lanes. The high gradient variance and skewness observed in the results confirm their effectiveness.

- Texture Features: The GLCM-based features (Contrast, Dissimilarity, Homogeneity, Energy) capture the texture variations in the lane markings and the road surface, enhancing the overall accuracy of lane detection.

- Effective Noise Reduction: Adaptive Thresholding and Morphological Operations ensure robust lane detection by minimizing unwanted artifacts and irrelevant edges.

INTERMEDIATE FEATURE REPRESENTATION:

Feature	Value	Interpretation
Edge Density (Overall)	2.9112	Moderate edge density indicates well-defined but not overly cluttered edges, capturing the essential lane boundaries.
Edge Density (Specific Region)	0.1811	Low edge density suggests that the targeted region of interest is focused, possibly reducing noise from the surroundings.
GLCM Contrast	77.5075	High contrast captures stark variations, essential for distinguishing lane markings from road surfaces.
GLCM Dissimilarity	3.8270	Moderate dissimilarity suggests there is a fair amount of texture variation near lane edges, helping in detailed boundary detection.
GLCM Homogeneity	0.3686	Low homogeneity indicates more textural diversity, likely from road textures, markings, and potential obstructions.
GLCM Energy	0.0304	Low energy suggests complex textures with diverse intensities, reflecting road wear or surface irregularities.
Edge Orientation Histogram	[2093, 1991, 2593, 2052, 109313, 2936, 3332, 3049]	The dominance of the central bin (109313) shows a strong presence of vertical or near-vertical edges, aligning with lane markings.
Gradient Magnitude (Mean)	225.2200	High mean gradient implies strong, clear transitions along detected lane edges.
Gradient Magnitude (Variance)	1,093,978.8401	High variance suggests significant fluctuations in edge strengths, indicating a varied road surface or markings.
Gradient Magnitude (Skewness)	8.1986	Positive skewness suggests more pronounced high-gradient transitions, possibly at sharp lane edges or road boundaries.
GLCM Correlation	0.9861	High correlation signifies strong, consistent intensity relationships in neighboring pixels, likely from structured lane patterns.

INTERPRETATION:

The extracted low-level features provide a comprehensive understanding of the image characteristics, aiding in the detection and interpretation of lane markings.

Below is a detailed interpretation based on each feature:

1. Edge Density (Overall and Specific Region):
   The overall edge density of 2.9112 signifies a balanced level of edge detail, ensuring lane markings are well-defined without excessive noise. The lower specific region edge density (0.1811) suggests a focused analysis area, potentially reducing background disturbances. This distinction aids in isolating relevant features from irrelevant ones.

2. GLCM Contrast and Dissimilarity:
   The high contrast value (77.5075) and moderate dissimilarity (3.8270) indicate significant intensity variations, vital for distinguishing lane markings from the road. These variations enhance the model's ability to detect boundaries, essential for self-driving applications.

3. GLCM Homogeneity and Energy:
   Low homogeneity (0.3686) and energy (0.0304) suggest a diverse texture, possibly due to road surface variations like cracks, shadows, or worn-out markings. Such diversity helps identify complex road scenarios and adjust lane detection techniques accordingly.

4. Edge Orientation Histogram:
   The dominance of vertical or near-vertical edges (central bin value 109,313) aligns with lane markings, ensuring accurate detection. The distribution of other bins highlights possible diagonal lane boundaries or curved road segments, making the feature suitable for adaptive lane detection.

5. Gradient Magnitude (Mean, Variance, and Skewness):
   The high mean gradient (225.2200) reflects distinct transitions along lane edges, aiding precise boundary detection. The substantial variance (1,093,978.8401) indicates significant intensity fluctuations, capturing varied road textures. The positive skewness (8.1986) suggests sharp transitions, essential for detecting lane boundaries in challenging conditions.

6. GLCM Correlation:
   A high correlation value (0.9861) shows consistent pixel intensity relationships, beneficial for structured lane detection. This consistency can reduce false positives and improve the robustness of lane identification.

Learning Outcomes:

After completing this assignment,

1. I have learned to implement computer vision techniques for lane detection, focusing on extracting low-level features like edge density, gradient magnitude, and GLCM (Gray Level Co-occurrence Matrix) texture features.
2. I can analyze and justify the choice of feature extraction methods based on the dataset characteristics, ensuring accurate and efficient detection of lane boundaries.
3. I can represent and interpret the extracted features in an intermediate form, understanding their significance for distinguishing lane markings from road surfaces and surrounding environments.
4. I understand the practical application of feature extraction techniques like Canny Edge Detection, Hough Line Transform, and GLCM analysis for self-driving car scenarios.
5. I can apply these techniques to develop robust lane detection systems capable of handling varying road conditions, noise, and complex environments.