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

Overview

This script performs lane detection in an image using Hough Transform, The steps involved are:

1. *Load Image*: Read the image from the file.

2. *Preprocess Image*: Convert it to grayscale, apply blurring, and detect edges.

3. *Region of Interest (ROI)*: Select only the road area to reduce noise.

4. *Hough Transform*: Detect lane lines by accumulating votes in Hough space.

5. *Overlay Lines*: Superimpose detected lanes on the original image.

6. *Display Results*: Show the outputs at different stages.

```
#Load image
image = cv2.imread("road2.jpeg")
```

Preprocess the image by converting to grayscale, applying a median blur, and detecting edges

1. Convert the image to grayscale to simplify processing

2. Apply a median blur to reduce noise while preserving edges

3. Use the Canny edge detector to extract edges from the image

```
def preprocess_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.medianBlur(gray, 5)
    edges = cv2.Canny(blurred, 50, 150)
    return edges
```

Masks the image to focus only on region of interest (the lane lines)

it creates a mask in the shape of a polygon covering the road area.

```
def ROI(edges):
    height, width = edges.shape
    mask = np.zeros_like(edges)
    polygon = np.array([[(100, height), (width//2 - 100, height//2 +
50),
```

```
                            (width//2 + 100, height//2 + 50), (width-100,
height)]], dtype=np.int32)
    cv2.fillPoly(mask, polygon, 255)
    masked_edges = cv2.bitwise_and(edges, mask)
    return masked_edges
```

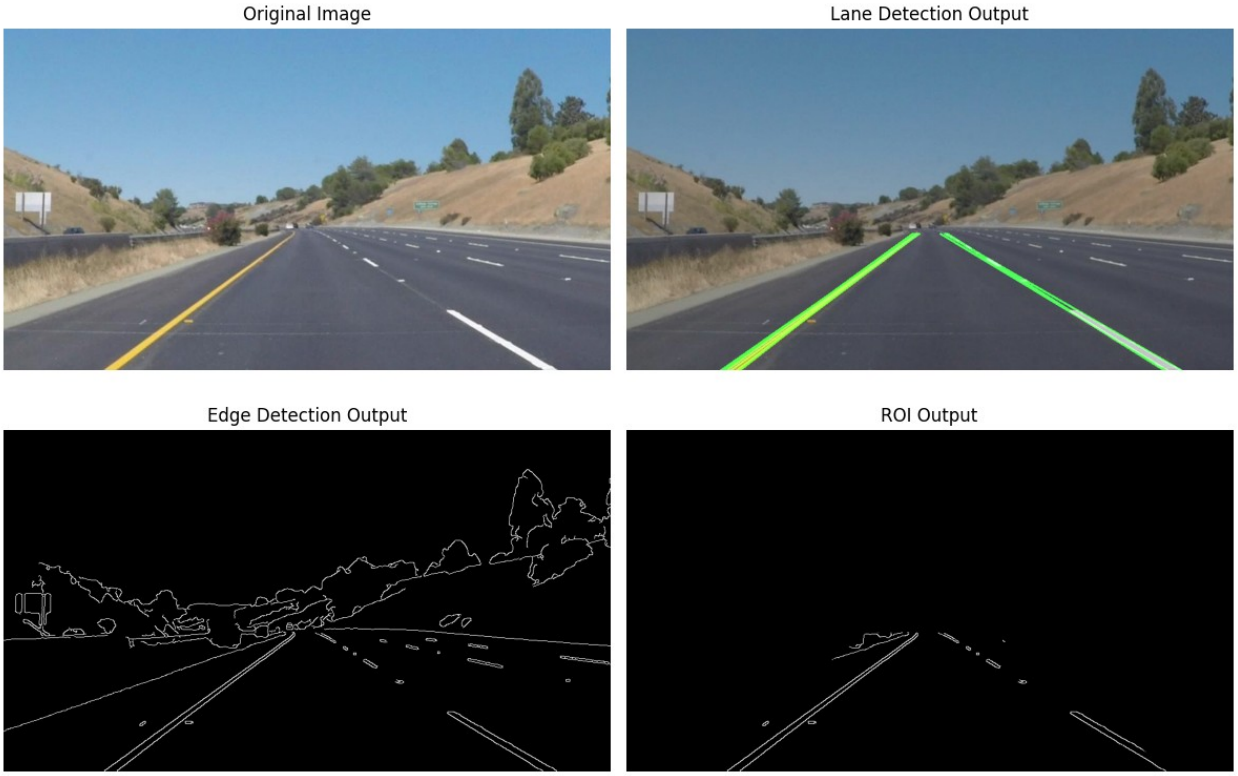
detects lines by accumulating votes in the Hough space in the edge-detected image.

```
def hough_transform(edges, image):
    height, width = edges.shape
    max_dist = int(math.sqrt(height**2 + width**2))  #Maximum possible
rho value
    theta_range = np.deg2rad(np.arange(-90, 90))  # -90 to 90 degrees
    accumulator = np.zeros((2 * max_dist + 1, len(theta_range)),
dtype=np.int32)
    edge_points = np.argwhere(edges)  # Get all nonzero edge pixels

    #Accumulate votes in Hough space
    for y, x in edge_points:
        for t_idx, theta in enumerate(theta_range):
            rho = int(x * np.cos(theta) + y * np.sin(theta)) +
max_dist  # Ensure non-negative index
            if 0 <= rho < accumulator.shape[0]: #Checking bounds
                accumulator[rho, t_idx] += 1

    #Identify the most significnt lines based on a voting threshold
    threshold = 100 #Minimum votes needed to consider a line
    lines = []
    for r_idx in range(accumulator.shape[0]):
        for t_idx in range(accumulator.shape[1]):
            if accumulator[r_idx, t_idx] > threshold:
                rho_val = r_idx - max_dist  # Convert back to actual
rho
                theta_val = theta_range[t_idx]
                lines.append((rho_val, theta_val))

    #draw the detected lines on a blank image
    line_image = np.zeros_like(image)

    for rho, theta in lines:
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho
        y0 = b * rho

        #Define lane region limits
        y1 = height #Bottom of image
        y2 = int(height * 0.6) #Top limit for lane line

        #calculate corresponding x values
```

```
        if a != 0:
            x1 = int((rho - y1 * b) / a)
            x2 = int((rho - y2 * b) / a)
        else:
            x1, x2 = x0, x0

        cv2.line(line_image, (x1, y1), (x2, y2), (0, 255, 0), 2)

    return line_image
```

Overlay detected lane lines into the original image

```
def overlay_lines(image, lines):
    return cv2.addWeighted(image, 0.8, lines, 1, 0)

#Process image
edges = preprocess_image(image)
roi_edges = ROI(edges)
line_image = hough_transform(roi_edges, image)
final_image = overlay_lines(image, line_image)

#Display
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
axes[0, 0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  #Convert
BGR to RGB for correct display
axes[0, 0].set_title("Original Image")
axes[0, 0].axis("off")

axes[0, 1].imshow(cv2.cvtColor(final_image, cv2.COLOR_BGR2RGB))
axes[0, 1].set_title("Lane Detection Output")
axes[0, 1].axis("off")

axes[1, 0].imshow(edges, cmap="gray")
axes[1, 0].set_title("Edge Detection Output")
axes[1, 0].axis("off")

axes[1, 1].imshow(roi_edges, cmap="gray")
axes[1, 1].set_title("ROI Output")
axes[1, 1].axis("off")

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




```
cv2.imwrite("edge_output.png", edges)
cv2.imwrite("roi_output.png", roi_edges)
cv2.imwrite("lane_detection.png", final_image)
```

```
True
```