Road Lane Segmentation

IIT2022129 - ATHARAV YADAV
IIT2022156 - NIRBHAY PALIWAL
IIT2022157 - ADITYA BHARDWAJ
IIT2022158 - SHIVAM KUMAR
IIT2022205 - PRATIK SARVAIYA

Approach 1 : Canny Edge Detection and Hough Transform

1. Image Preprocessing
2. Canny Edge Detection
3. Region of Interest (ROI) Masking
4. Hough Line Transform

STEP-1 IMAGE PREPROCESSING

- **CONVERT IMAGE TO GRAYSCALE**
  - Simplifies the image by reducing it to intensity values, removing unnecessary color data

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

$$\text{Grayscale} = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B$$

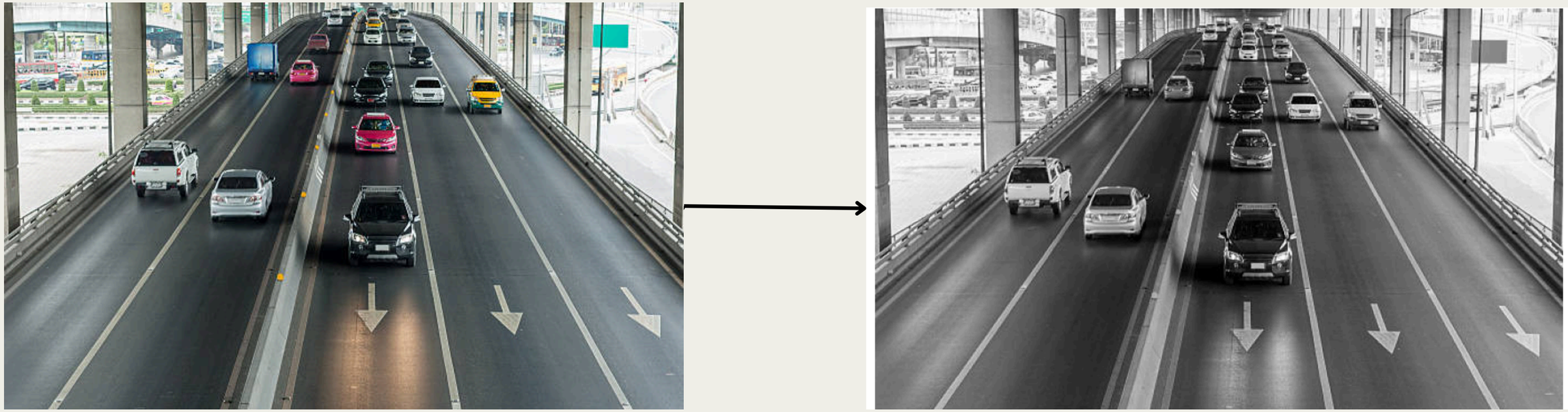
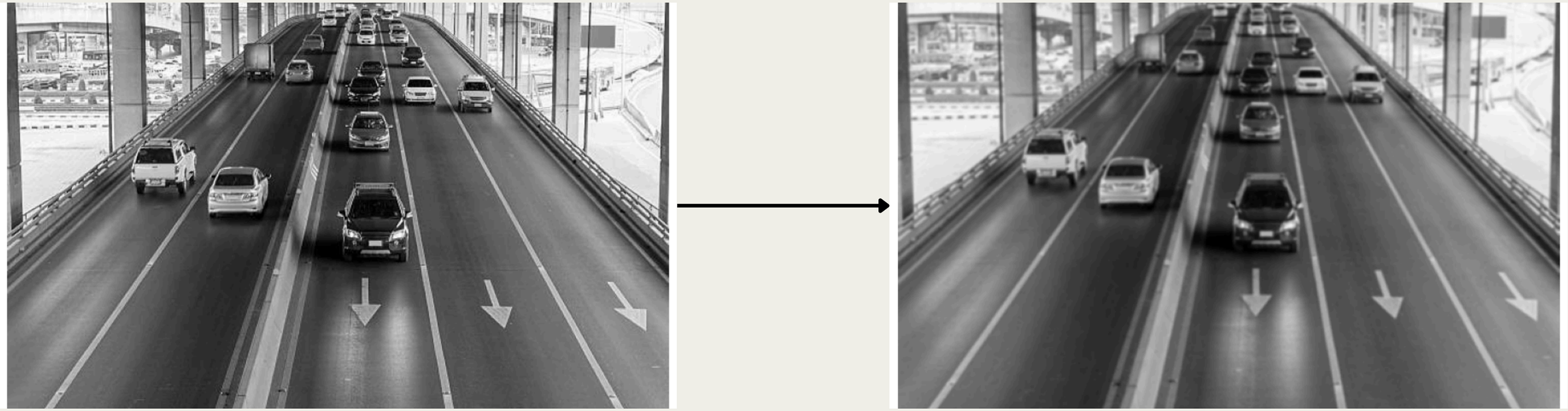


IMAGE PREPROCESSING

- **APPLY GAUSSIAN FILTER**
  - Reduces noise to prevent false edge detection.
  - Enhances edge clarity for cleaner results and it reduces noise that might create false edges.

```
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
```



Applying guassian blur using a 5x5 size kernel

STEP-2 CANNY EDGE DETECTION

• GRADIENT CALCULATION USING SOBEL OPERATOR

- The gradient represents the change in pixel intensities, and its magnitude gives the strength of the edge.

```
edges = cv2.Canny(blurred, 30, 100, apertureSize=3)
```

- **Gradient in the x-direction (G_x):**

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

- **Gradient in the y-direction (G_y):**

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

CANNY EDGE DETECTION

Gradient Magnitude: The magnitude of the gradient at each pixel is then calculated using the formula:

$$G = \sqrt{G_x^2 + G_y^2}$$

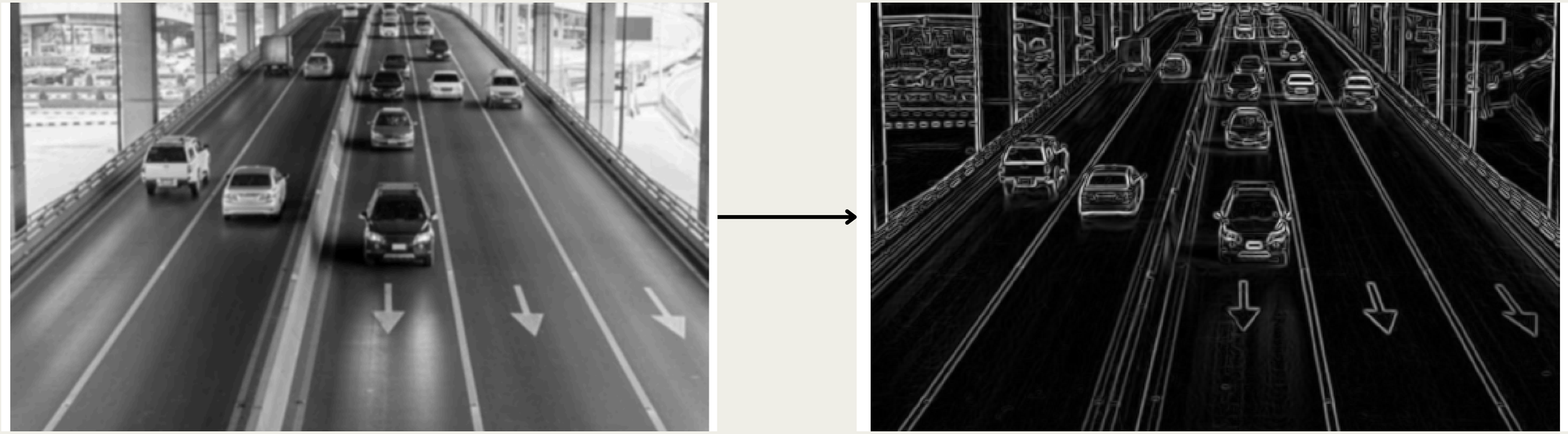
Gradient Direction: The direction of the edge (angle) can be calculated using:

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

CANNY EDGE DETECTION

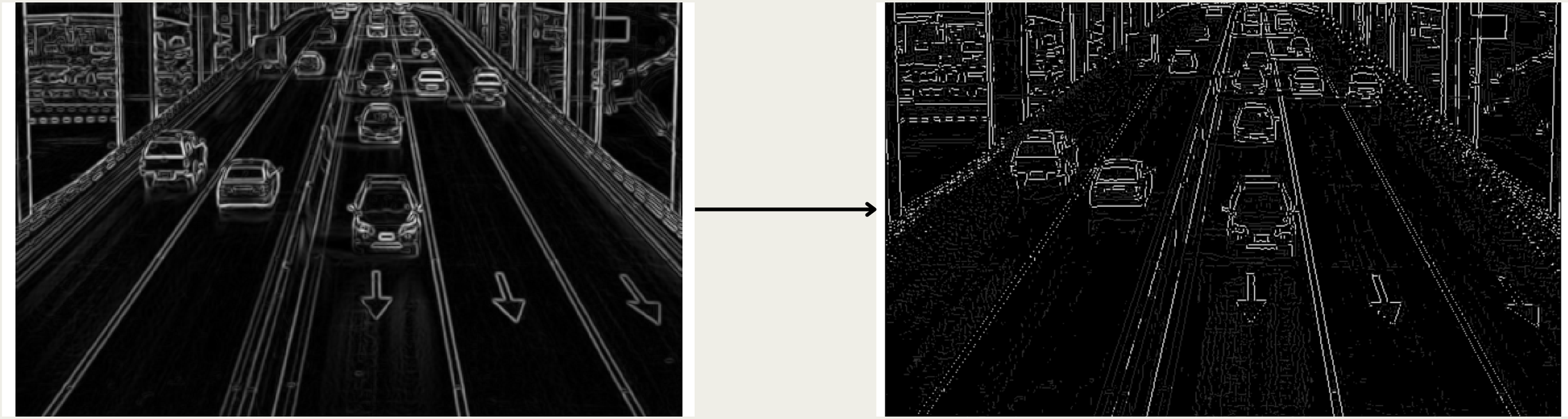
- **GRADIENT CALCULATION USING SOBEL OPERATOR**

The final result looks like this:



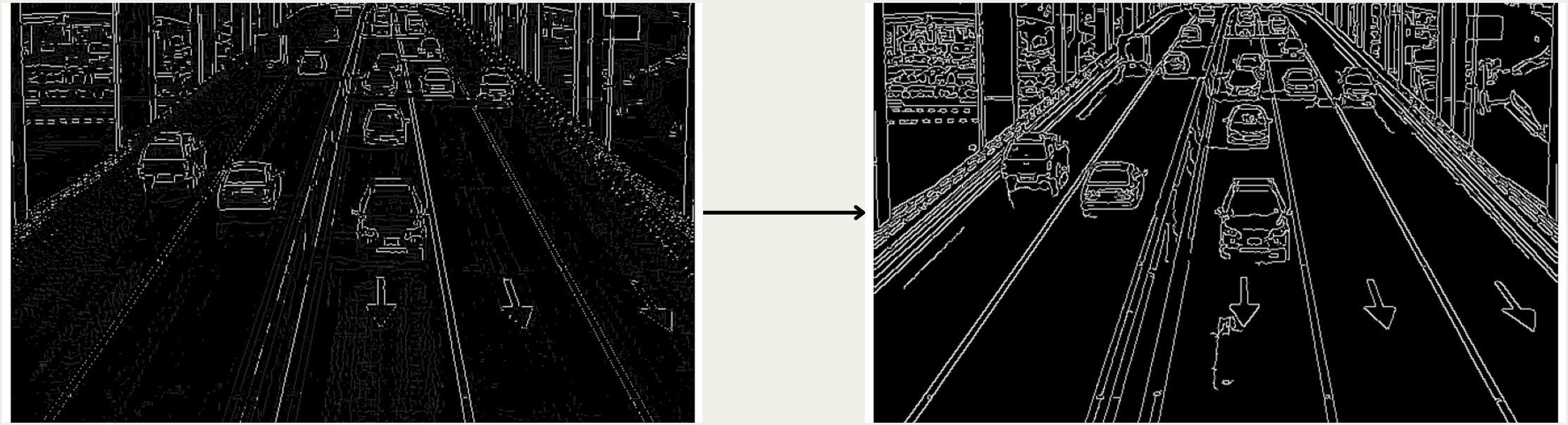
CANNY EDGE DETECTION

- **DOUBLE THRESHOLDING**
  - Two threshold values—low_threshold and high_threshold
  - Strong Edges: If a pixel's gradient intensity is greater than the high_threshold,
  - Weak Edges: If a pixel's intensity is between the low_threshold and the high_threshold.



CANNY EDGE DETECTION

- **EDGE TRACKING BY HYSTERESIS**
  - Retain weak edges that are connected to strong ones, while discarding isolated weak edges.

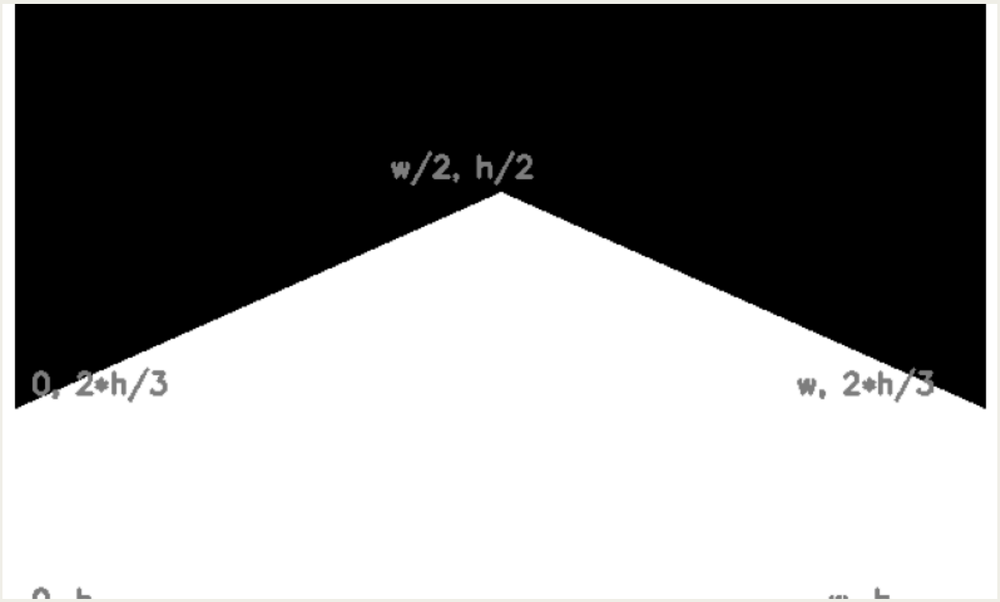


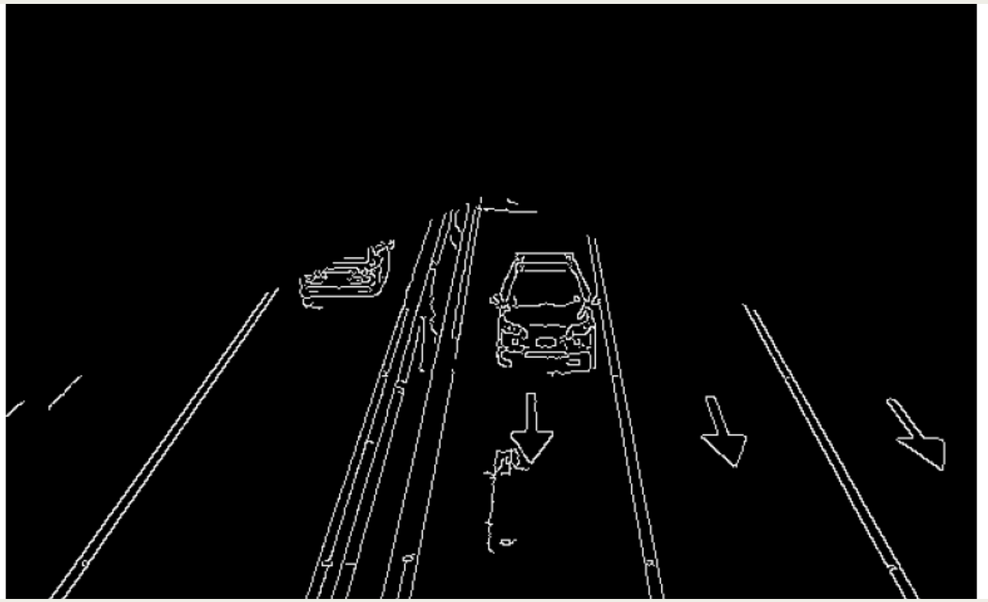
STEP-3 REGION OF INTEREST(ROI)

USED A PENTAGONAL ROI TO FOCUS ON THE AREA OF THE ROAD WHERE LANE MARKINGS ARE MOST LIKELY TO APPEAR.

```
cv2.fillPoly(mask, [points], 255)
masked_edges = cv2.bitwise_and(edges, edges, mask=mask)
```

```
points = np.array([
    [0, height],
    [0, 2*height // 3],
    [width // 2, height // 2],
    [width, 2*height // 3],
    [width, height]
], np.int32)
```





STEP-4 HOUGH LINES

- Hough Transform is used to detect the geometrical shapes in boundary, In this case we are using to detect the Straight lines.

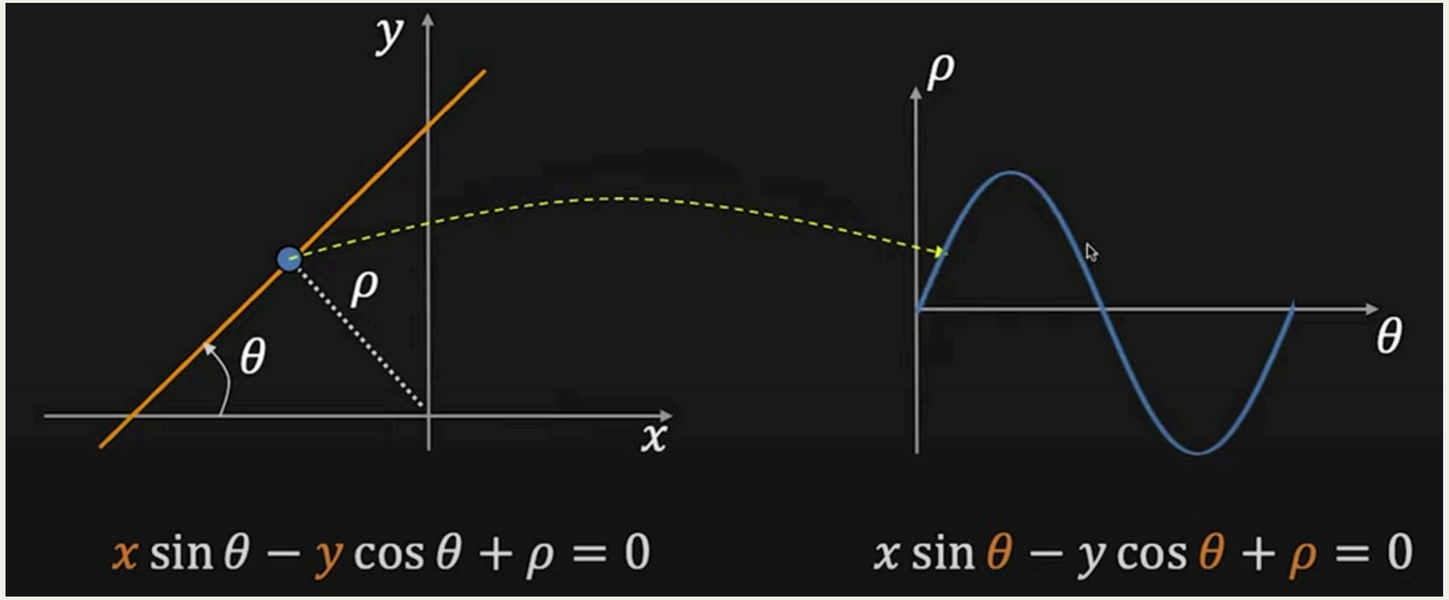
minLineLength -It defines the minimum pixels a line should to be considered

maxLineGap -This is the maximum allowed gap between line segments to treat them as a single line.

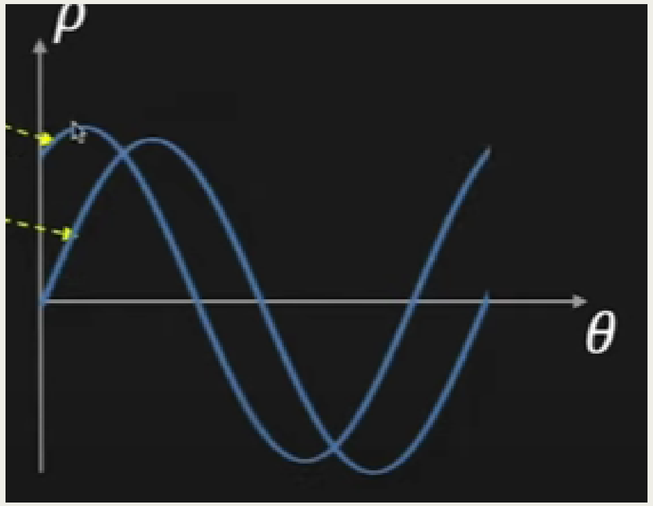
```
lines = cv2.HoughLinesP(
    masked_edges,
    rho=1,
    theta=np.pi / 180,
    threshold=80,
    minLineLength=75,
    maxLineGap=20
)
```

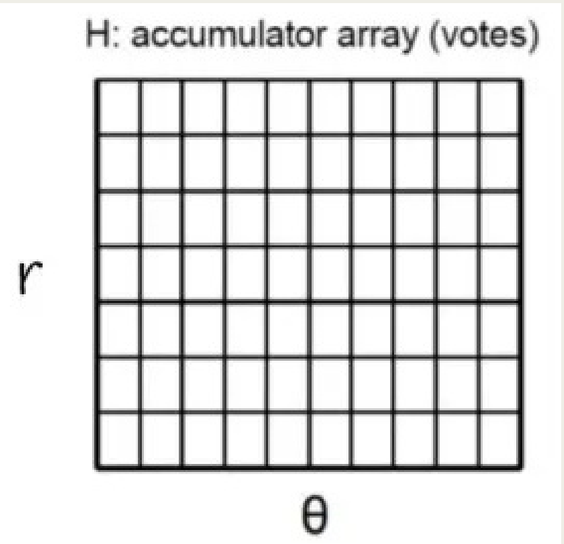
HOUGH LINES

- Each point in the image space is plotted to corresponding point in parameterized space.



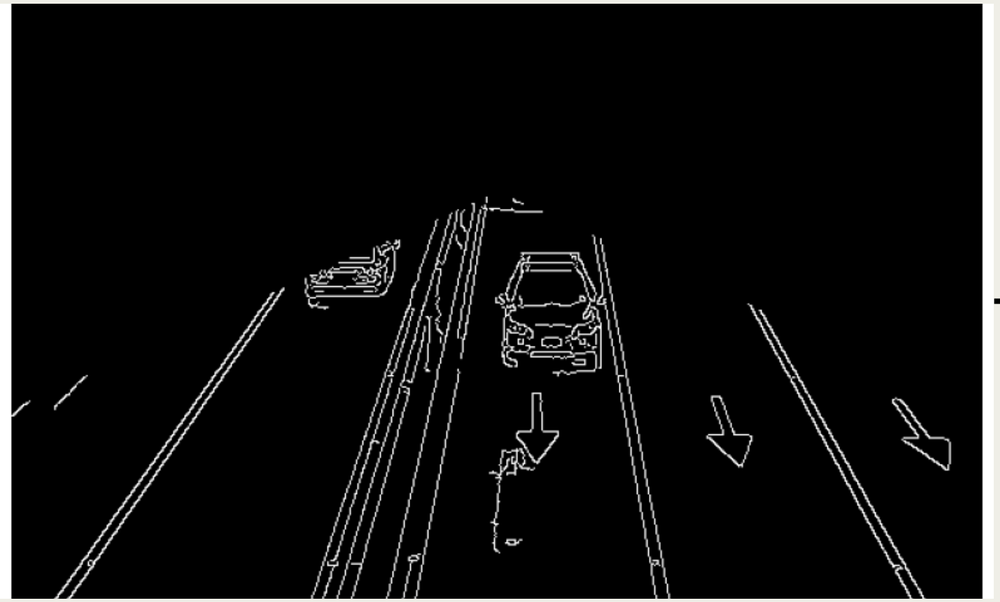
- Each points casts its vote in accumulator matrix points.

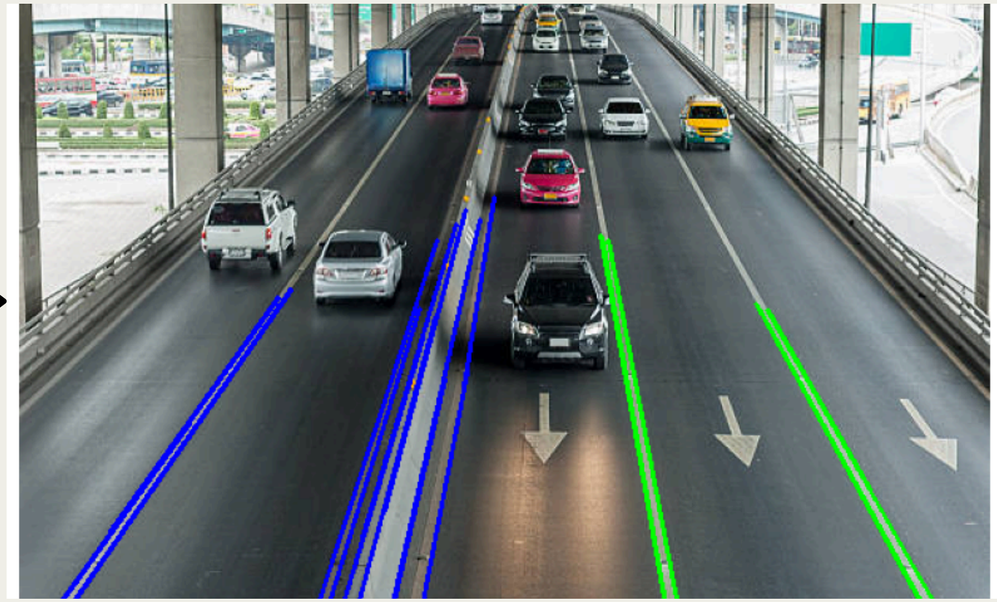




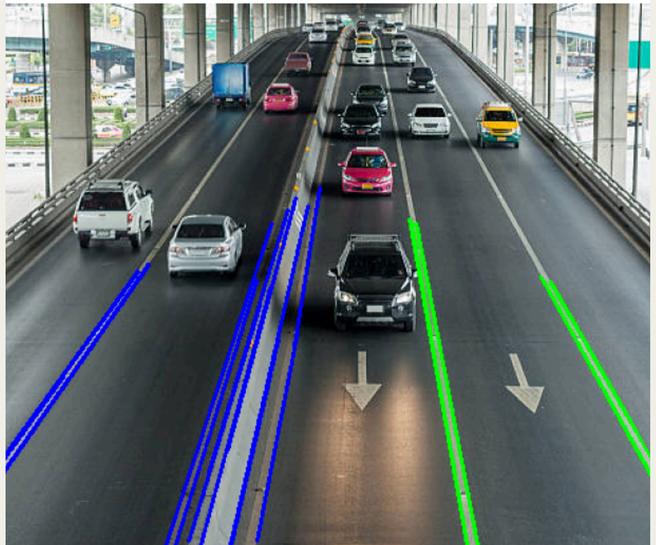
HOUGH LINES

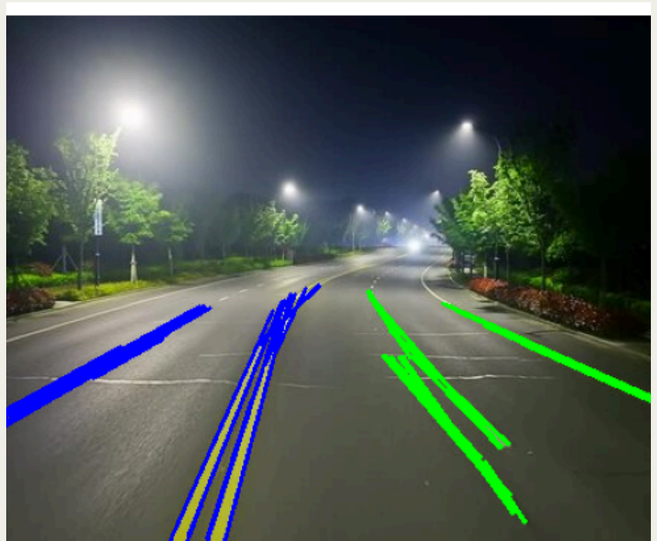
The final result looks like this:





SAMPLES





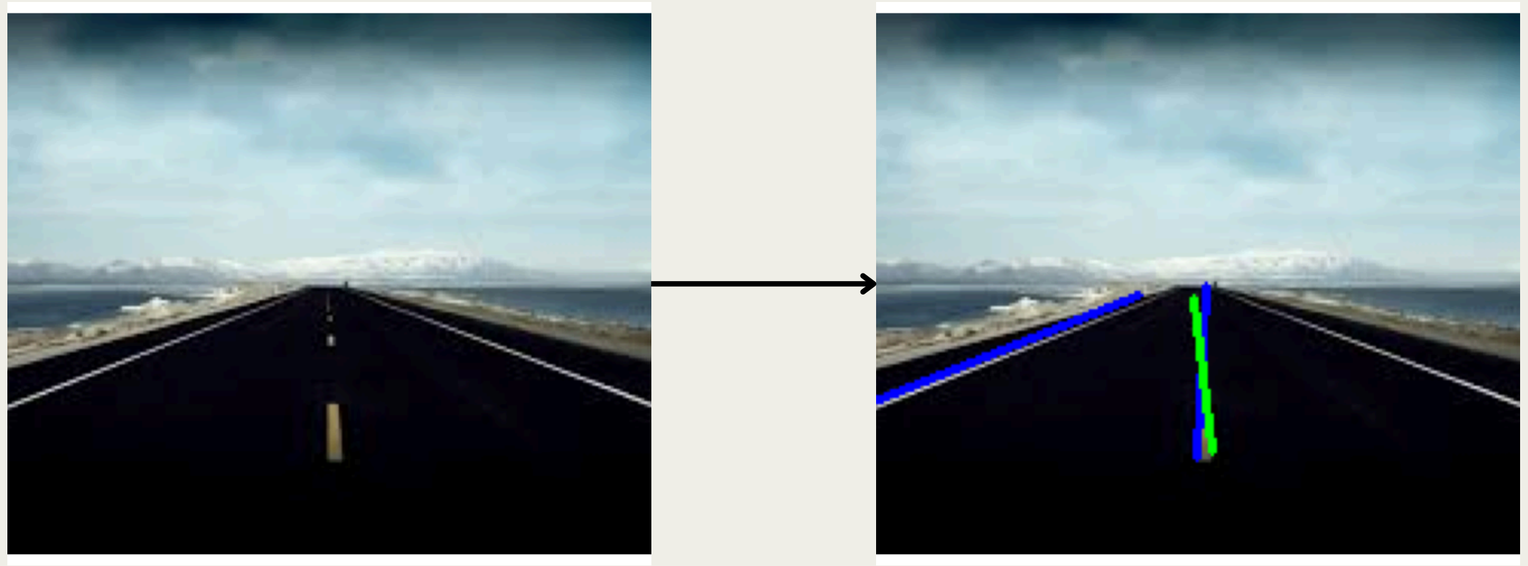
Limitations

(1) Non-Lane Objects in ROI

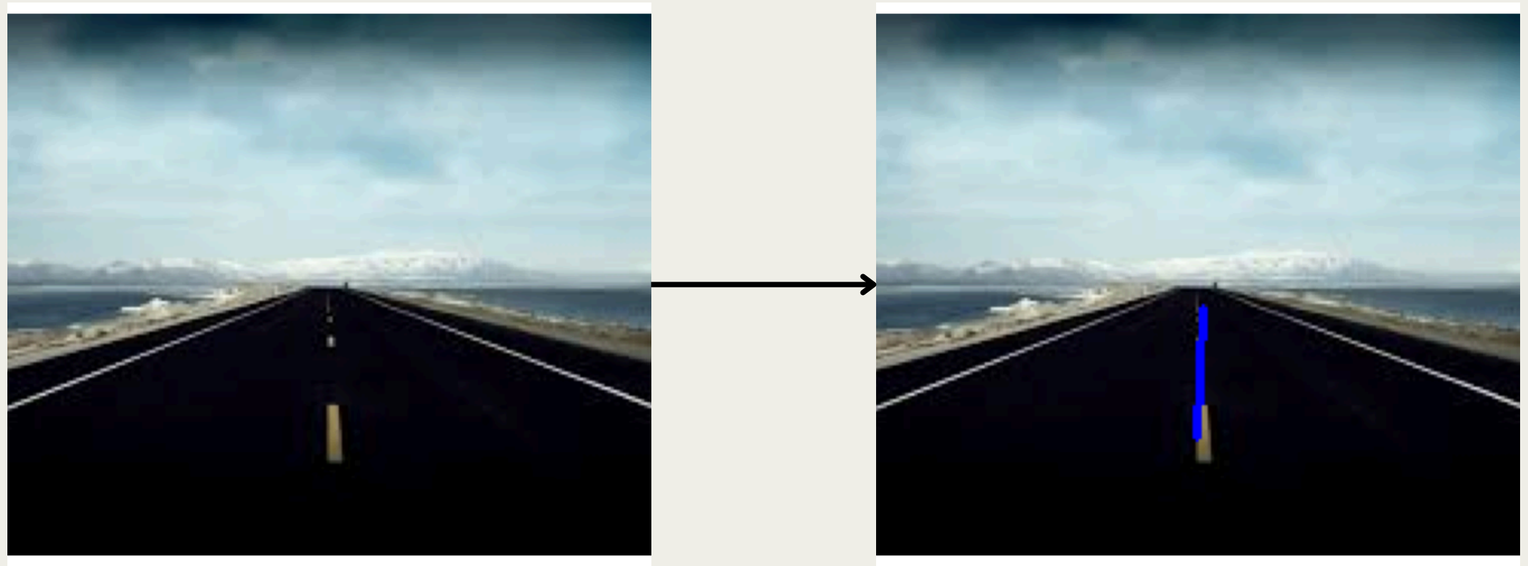
LIMITATIONS

(2) Hough Line Detection Parameters

Threshold Value : 20



Threshold Value : 25



APPROACH - 2 : USING THRESHOLDING & MORPHOLOGICAL OPERATION

Steps :

1.GrayScale Conversion

2.Power law Transformation

3.Gaussian blurring

4.Binary Thresholding

5.Selecting Region of Interest

6.Dilation

7.Contour Coloring

STEP - 1 : GRAYSCALE CONVERSION

The image is converted to grayscale which helps to focus on only the intensity changes in the image, making lane detection easier by removing color distractions.

```
gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
```

STEP-2 : POWER LAW TRANSFORMATION

- In many road scenes, bright regions like sunlight reflections, or highlights may cause distractions. A gamma value greater than 1 darkens these regions, making them less dominant in the image.
- If the image is too bright, it can make thresholding difficult because the bright areas might obscure lane markings or introduce unwanted noise.

```
c=1
gamma=2.0
power_law_transformed=c*(gray/255)**gamma
power_law_transformed=np.clip(power_law_transformed*255,0,255).astype(np.uint8)
```

STEP - 3 : GAUSSIAN BLURRING

- Reduces Noise: Smooths the image to remove small, irrelevant details like texture or minor imperfections on the road.
- Preserves Important Edges: The blur reduces noise while keeping lane edges moderately sharp for further processing steps.

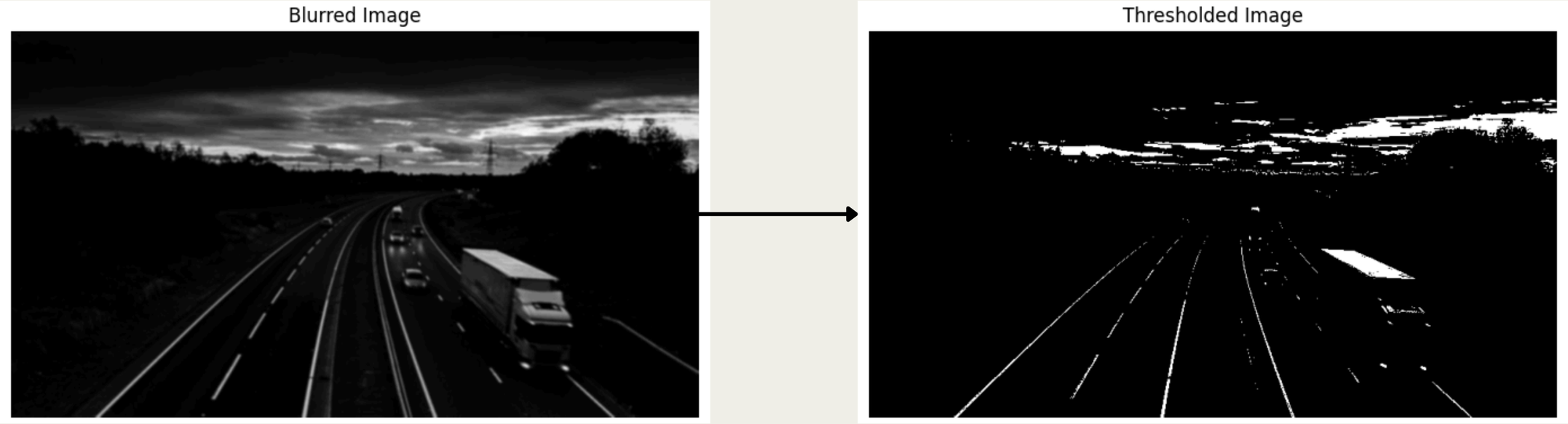
```
blurred=cv2.GaussianBlur(power_law_transformed,(5,5),0)
```

STEP - 4 : BINARY THRESHOLDING

Simplifies Lane Detection: Converts the image into a high-contrast black-and-white format, making it easier to isolate lane markings by separating the background and the road lines.

Reduces Complexity: After thresholding, only two intensity levels (black and white) remain, simplifying the task of identifying lane contours.

```
thresholded=cv2.threshold(power_law_transformed,125,255,cv2.THRESH_BINARY)
```

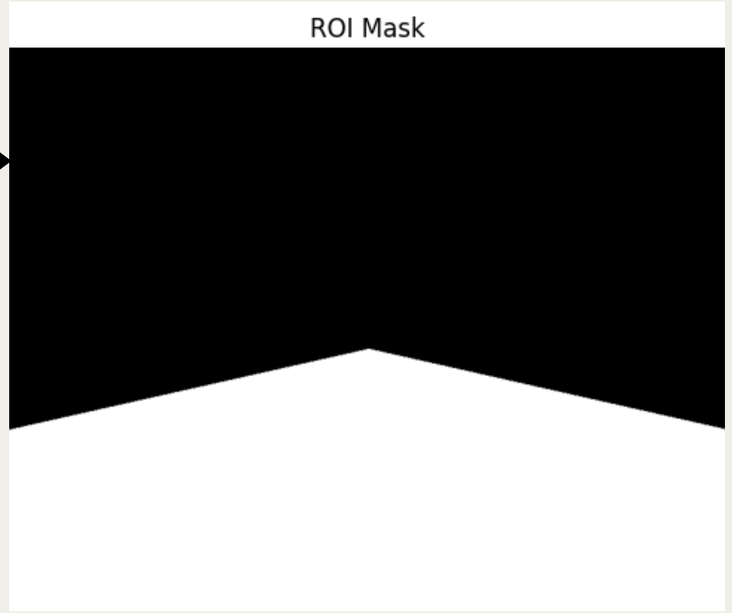


STEP - 5 : SELECTING REGION OF INTEREST

This reduces the area being processed to only the part of the image likely to contain lanes (typically the lower half), speeding up processing and reducing false detections.

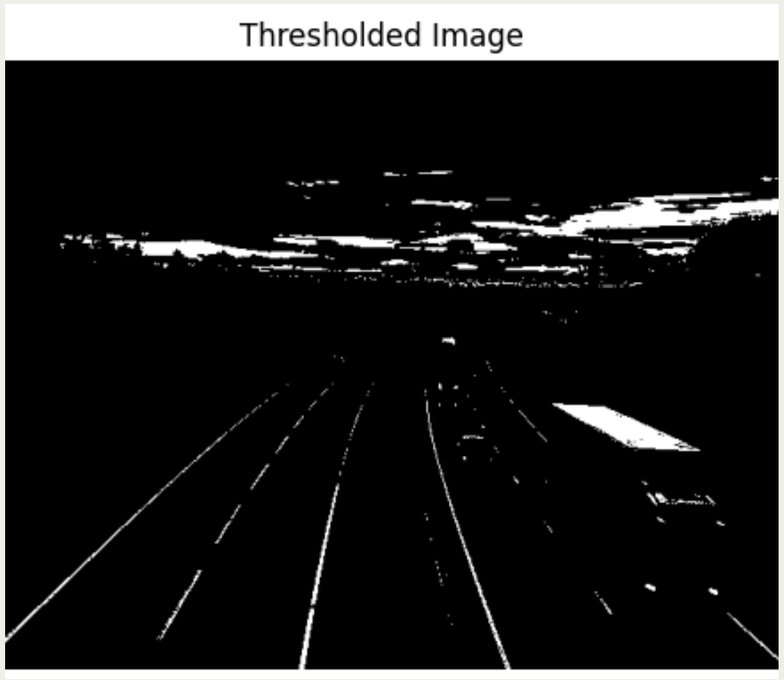
```
f region_of_interest(img):
  height, width=img.shape[:2]
  #Define the "home" shape points
  bottom_left=(0, height)  #bottom left
  bottom_right=(width, height)  #bottom right
  peak=(int(width/2),int(height*0.55))  #peak
  upper_left=(0,int(height*0.7))  #upper left
  upper_right=(width,int(height*0.7))  #upper right

  polygons=np.array([[bottom_left,upper_left,peak,upper_right,bottom_right]],dtype=np.int
  #Create a mask initialized to 0 with the same dimensions as the image
  mask=np.zeros((height, width),dtype=np.uint8)
  cv2.fillPoly(mask,polygons,255)
  #do bitwise AND
  masked_image=cv2.bitwise_and(img,img,mask=mask)
  return masked_image
```



STEP - 5 : SELECTING REGION OF INTEREST

```
roi=region_of_interest(thresholded)
```



STEP - 6 : APPLYING DILATION

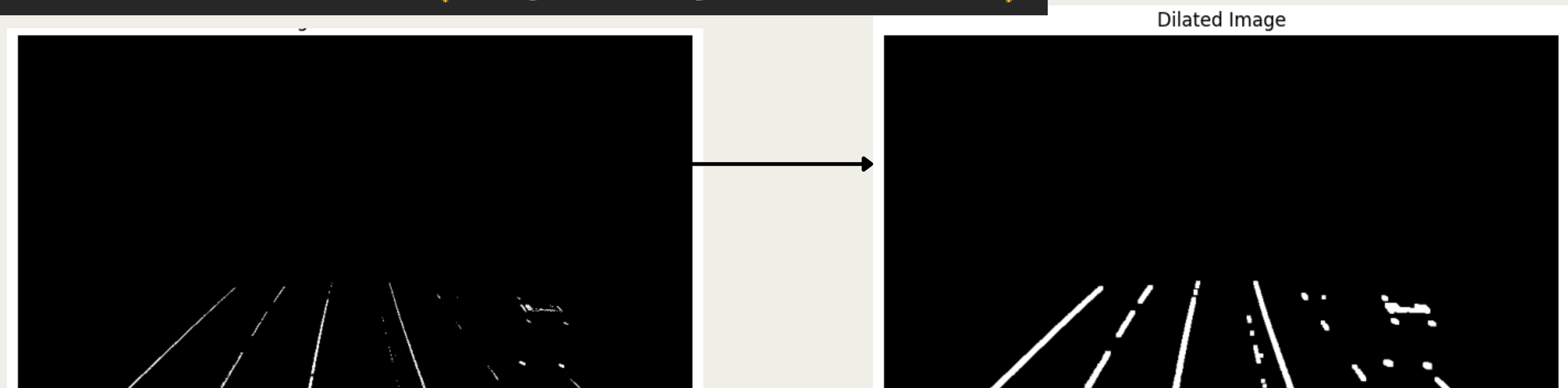
Dilation takes each pixel in the image and checks the region around it using this kernel. If any of the pixels in the kernel area are white (value 1), the pixel in the output image is set to white (1).

This helps in bridging small gaps and thickening thin lane lines, making them more continuous and easier to detect.

*5*5 size Kernel*

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

```
kernel=np.ones((5, 5),np.uint8)
dilated=cv2.dilate(roi,kernel,iterations=1)
```



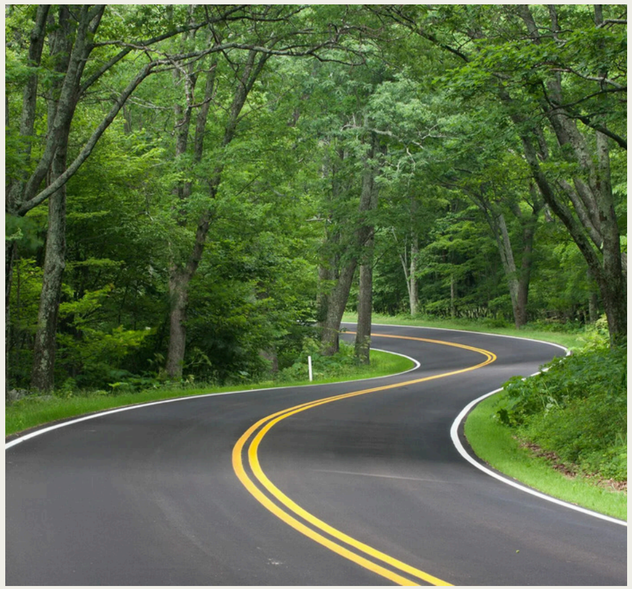
STEP - 7 : CONTOUR COLORING

A contour is a curve or boundary that joins all the continuous points along a particular boundary with the same intensity.

We first find the contours which are the boundaries of the lanes and then we fill these contours

```
contours,_=cv2.findContours(dilated,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
for contour in contours:
    color=(0,0,255)  #fix red color
    cv2.fillPoly(colored_output,[contour],color)
```

SAMPLES



Filled Lane Markings

Filled Lane Markings

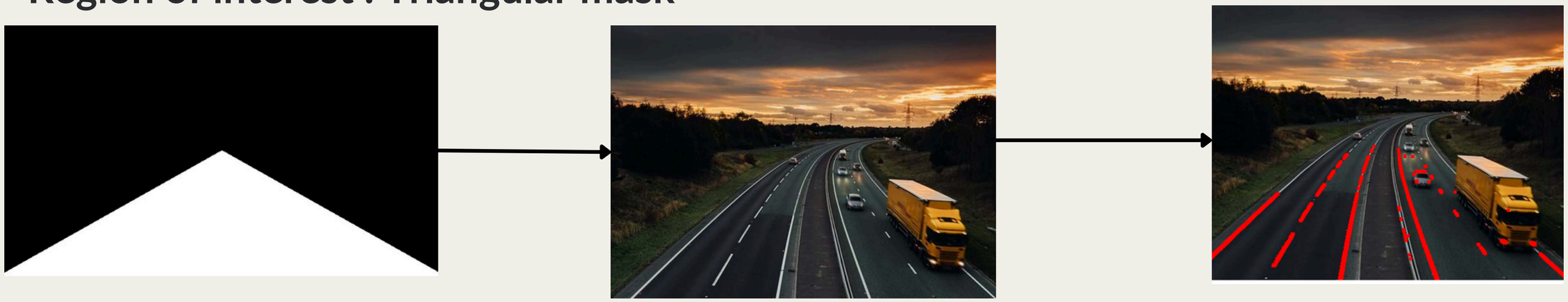
Filled Lane Markings

LIMITATIONS

(1) Depends on the region of interest .

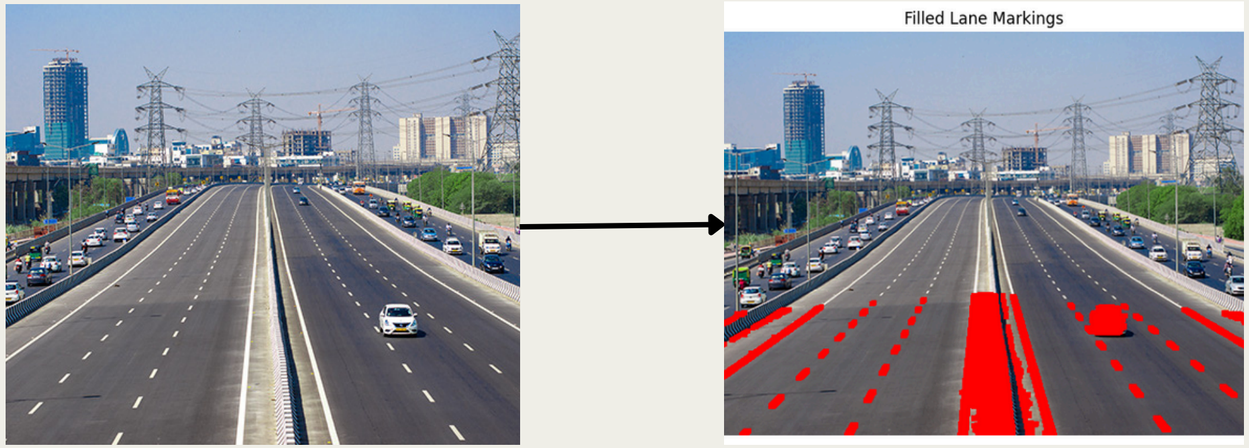
Region of interest : Polygon mask

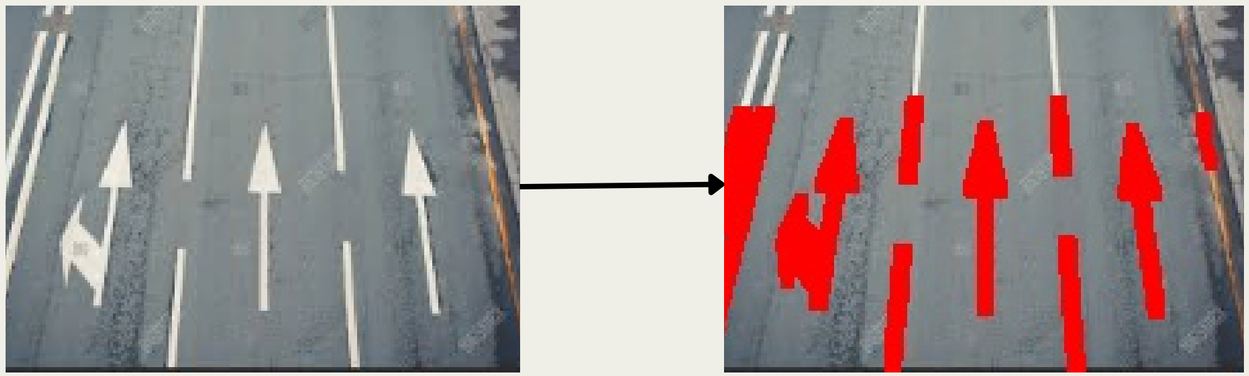
Region of interest : Triangular mask



LIMITATIONS

(2) If region of interest contains brighter objects other than lanes then also it identifies as lanes.

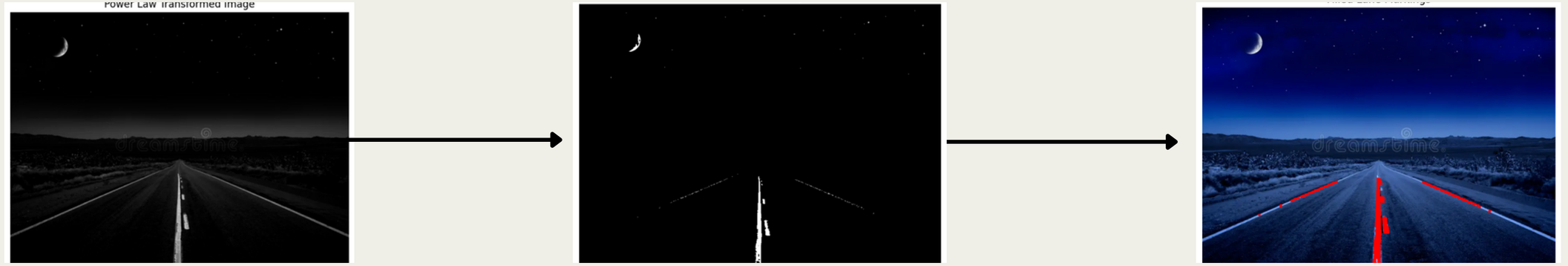




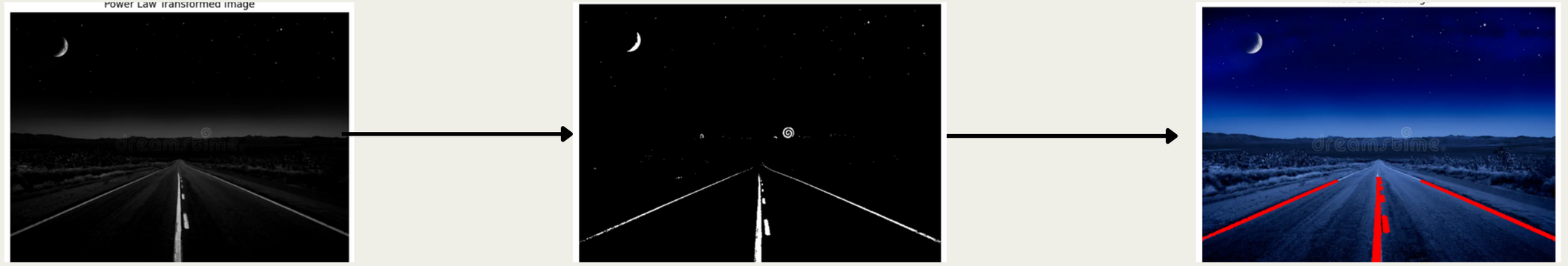
LIMITATIONS

(3) Depends on threshold value.

Threshold Value : 125



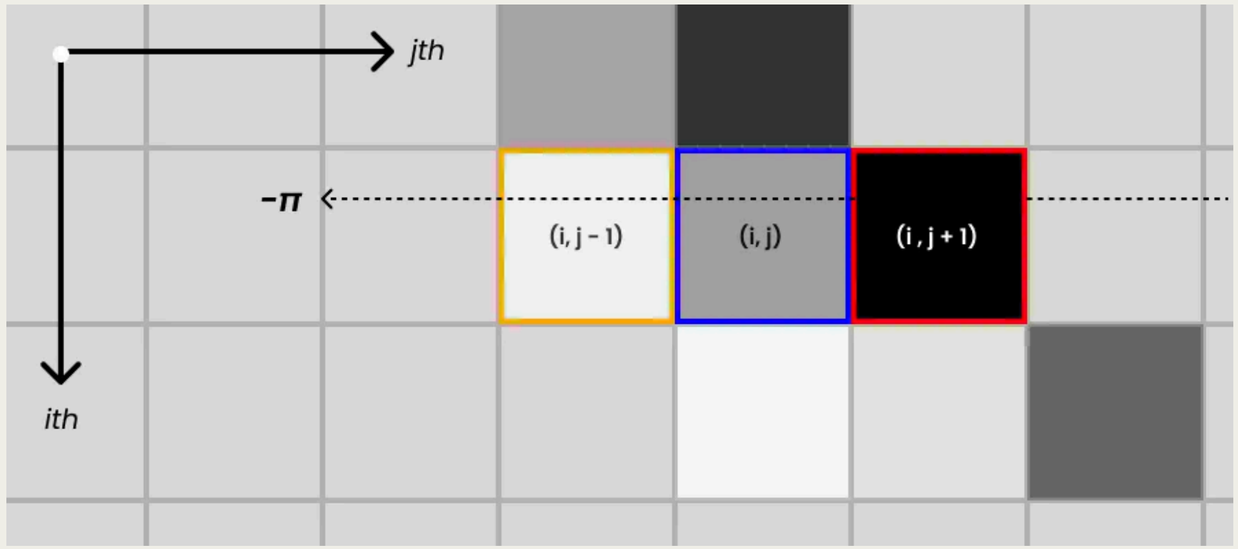
Threshold Value : 70

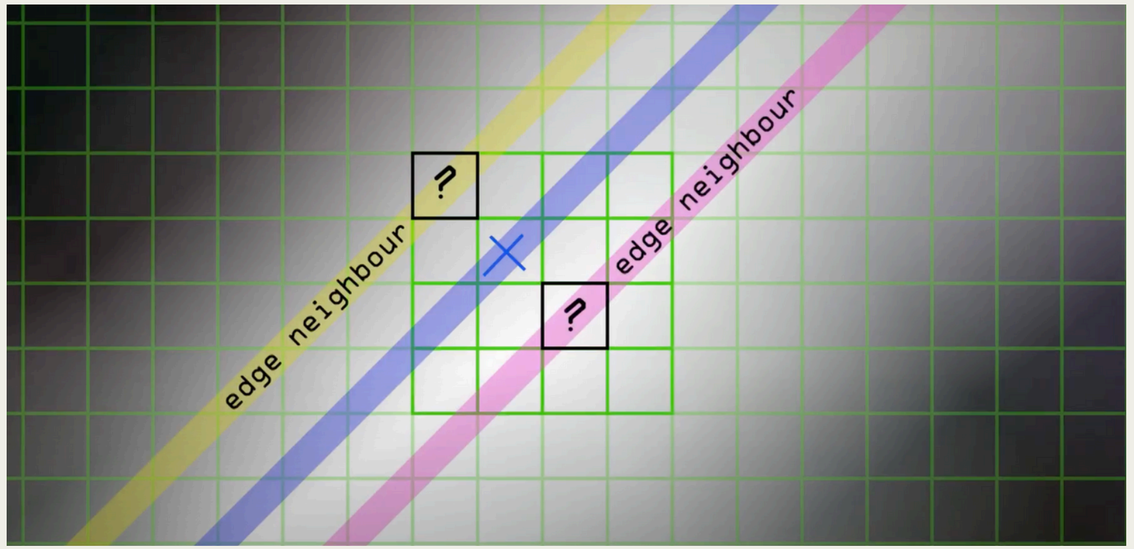


Thank you!

CANNY EDGE DETECTION

- **NON-MAX SUPPRESSION**
  - Edges may still be thick and blurry.
  - It helps in thinning the edges to create more accurate boundaries.
  - Each pixel is checked to see if it is a local maximum in the direction of the gradient.





CANNY EDGE DETECTION

- **NON-MAX SUPPRESSION**

  The final result looks like this: