Mini Project

Project X
TECO

Mohd Yusuf
Altamash Khan
Talha Fakih
Aaris Kazi

Lane Detection and Pedestrian Tracking System

Under the guidance of
Prof. Mubashir Khan
(Lecturer, Computer Engineering. Dept)
Department of Computer Engineering

Problem Statement:

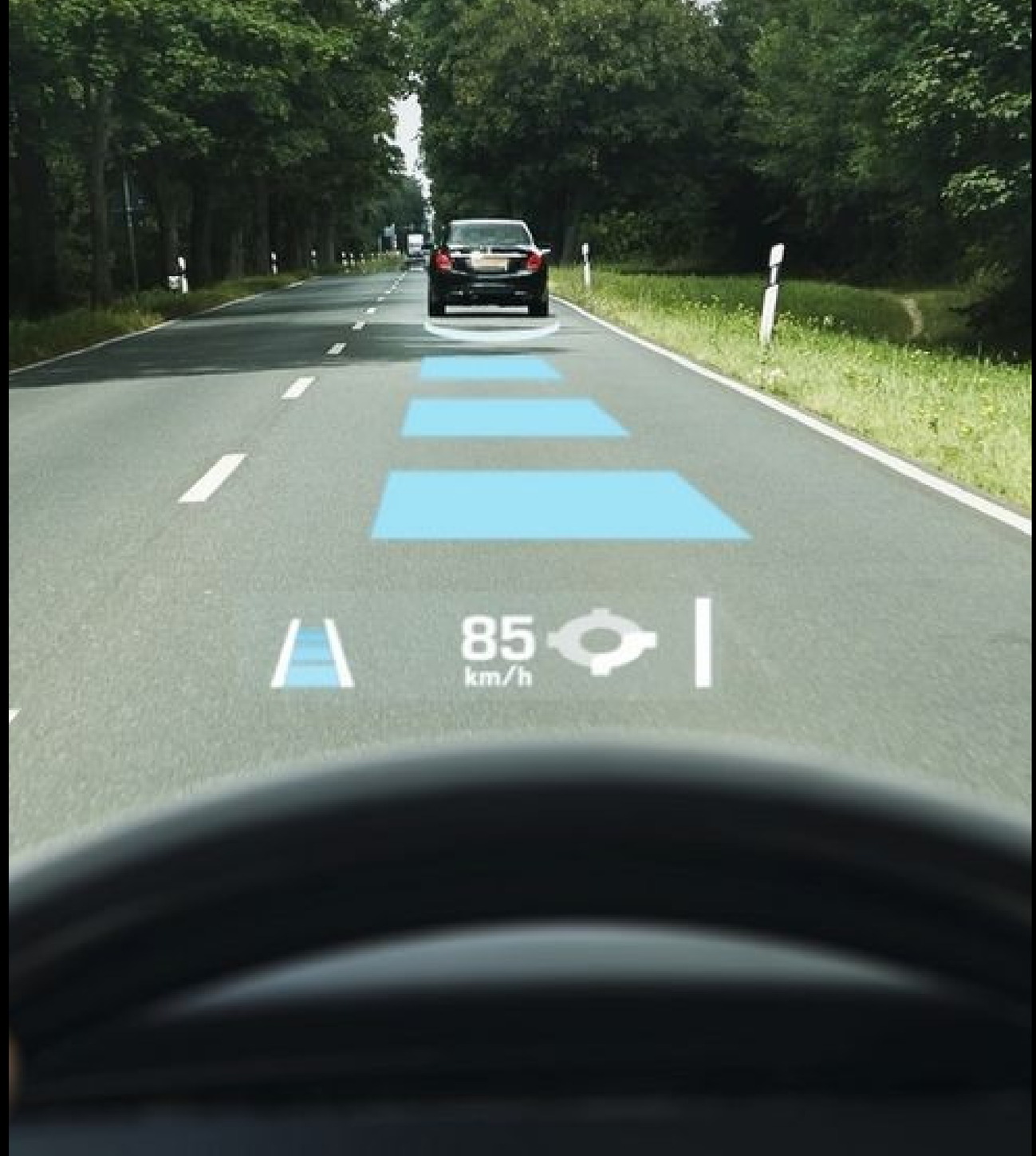
Lane Detection is a multifeature detection problem that has become a real challenge for computer vision and machine learning techniques. Although many machine learning methods are used for lane detection, they are mainly used for classification rather than feature design.

Introduction

- Traffic accidents are a serious growing problem
- Goals:
    a. Safety
    b. User-friendly
    c. Cost Saving
- Challenges:
        Real time dynamic complex environment
        Process large amount of data
        Robust vehicle motion control

Motivation

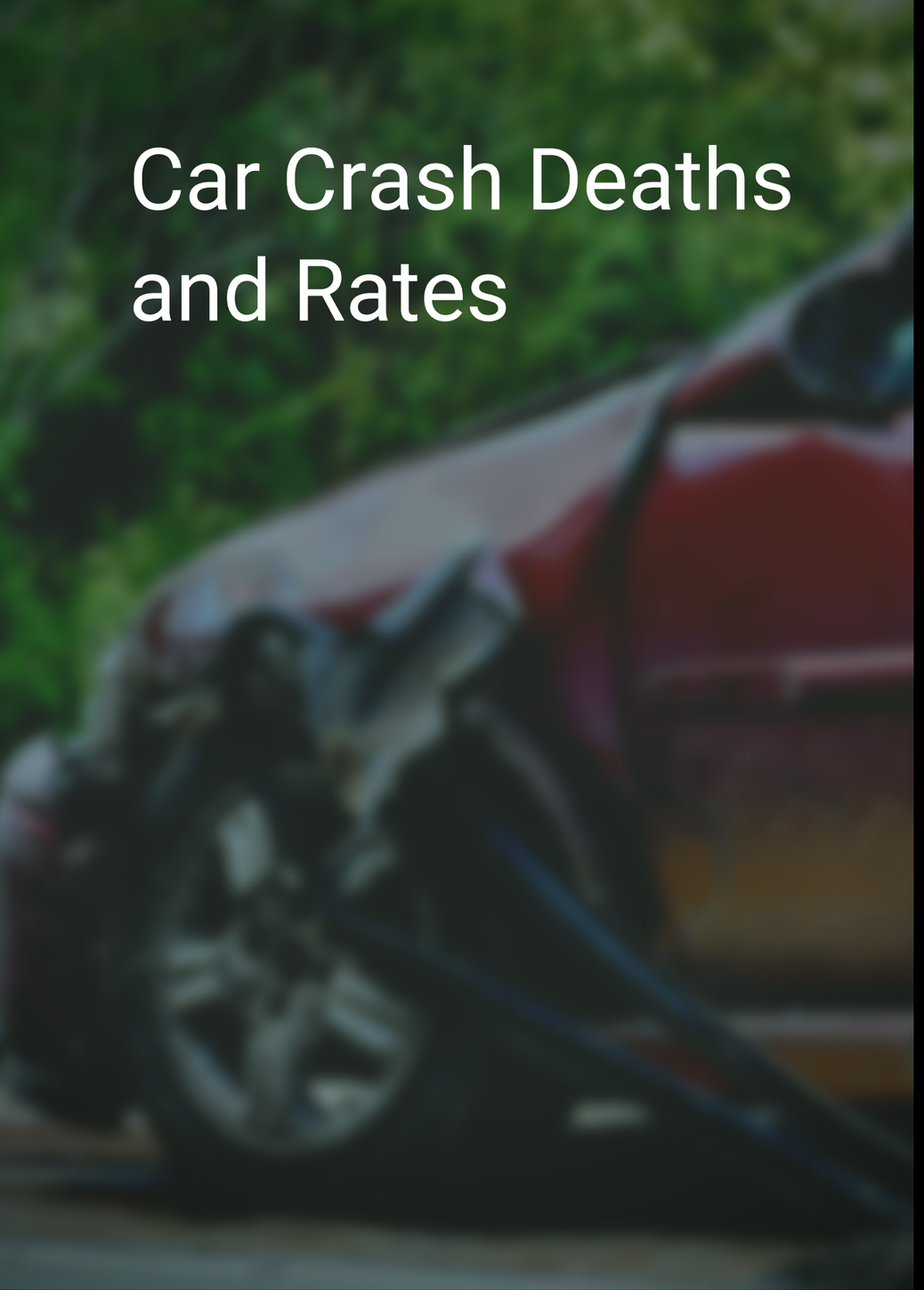
- Autonomous driving
- Driver assistance (collision avoidance, more precise driving directions)

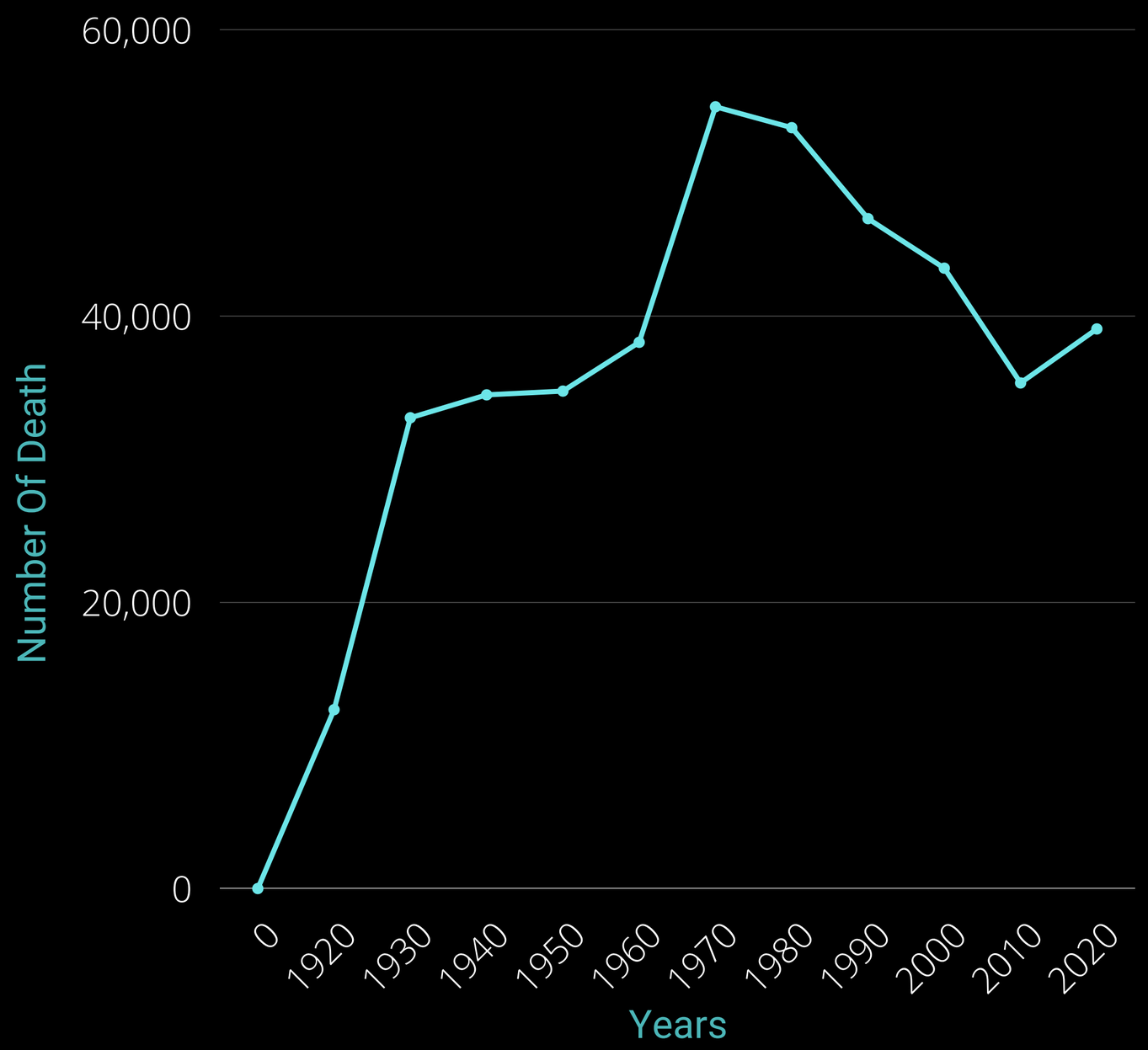


Goals of our lane tracker

- Recover lane boundary for straight or curved lanes in suburban environment
- Recover orientation and position of vehicle in detected lane boundaries
- Use temporal coherence for robustness

Car Crash Deaths and Rates



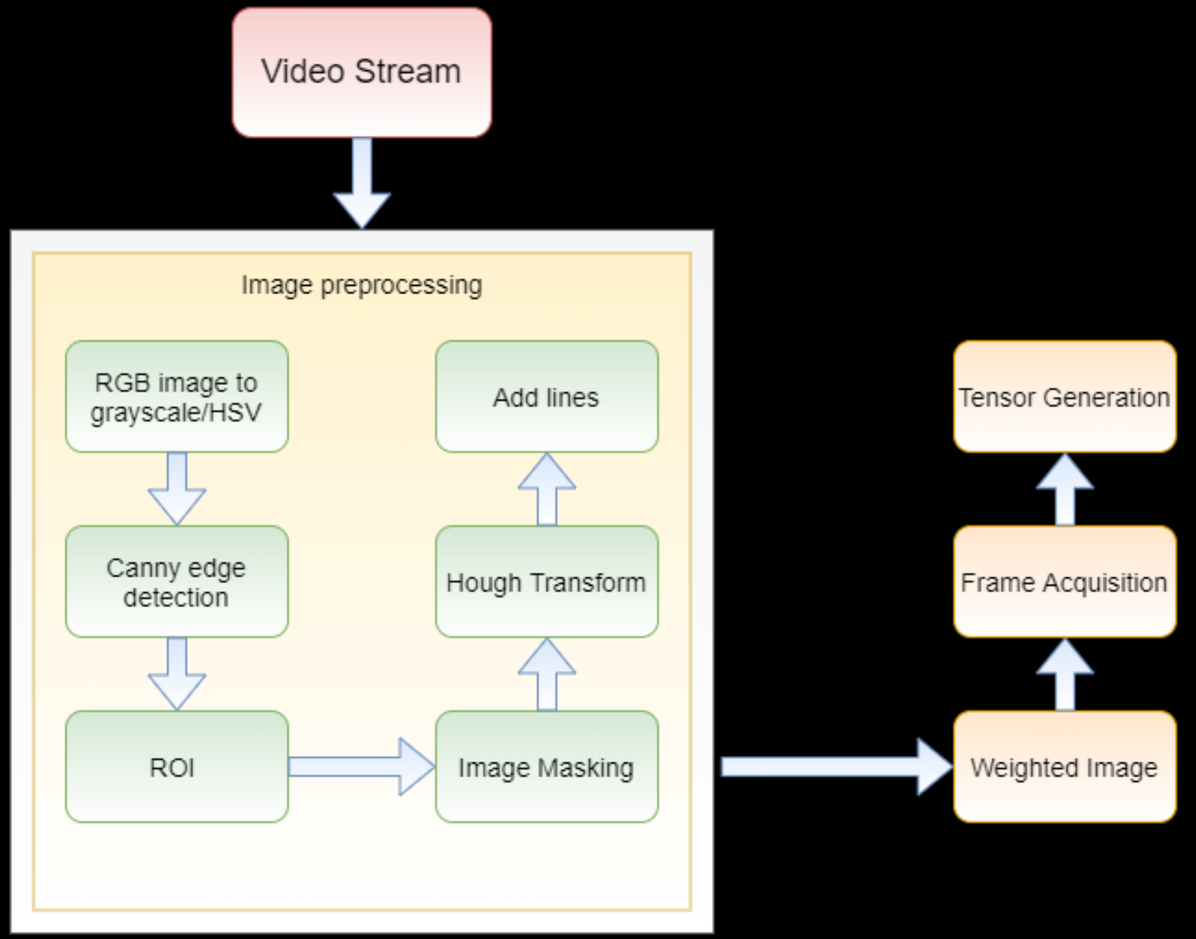


Processing of Model:

Before

After

System Architecture:

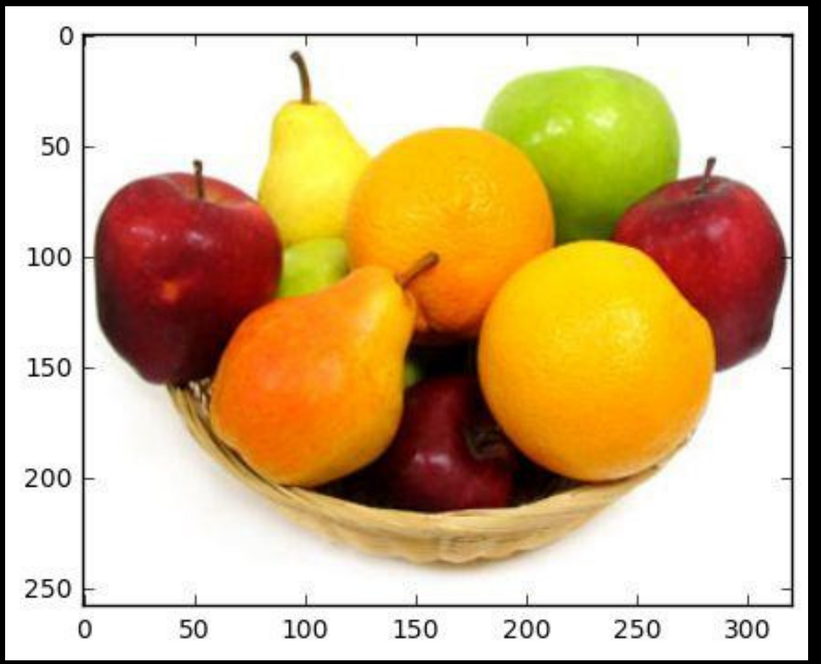


Functions Used:

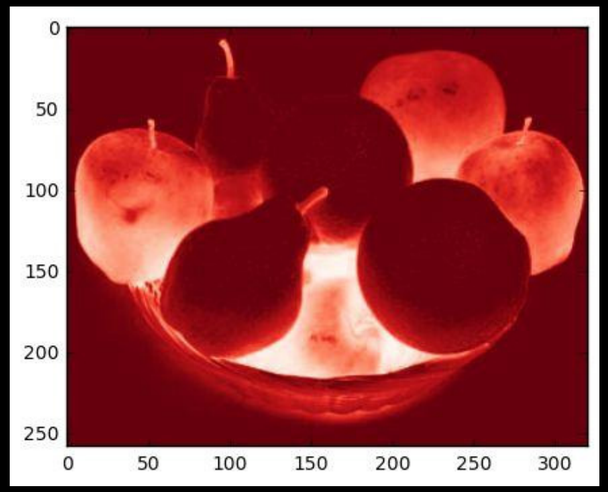
A) Image as tensors
B) Grayscale
C) Gaussian Blur
D) Canny Edge Detection
E) Masked Image
F) Hough Transform
G) Overlay Hough Image over Original

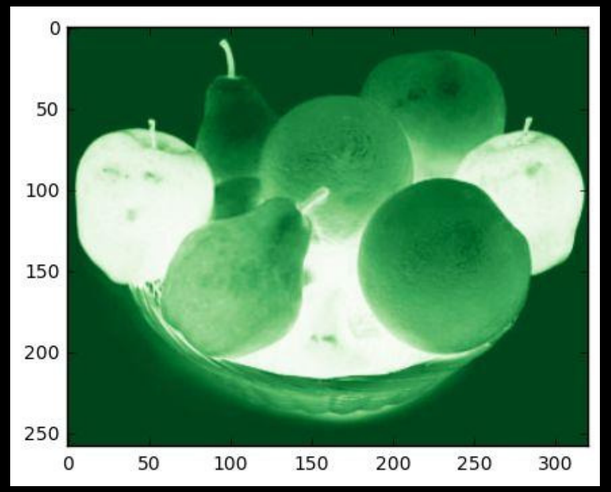
A) Visualizing Images as Tensors

Image.shape => (258, 320, 3)



We can isolate the Red, Green, and Blue Channels





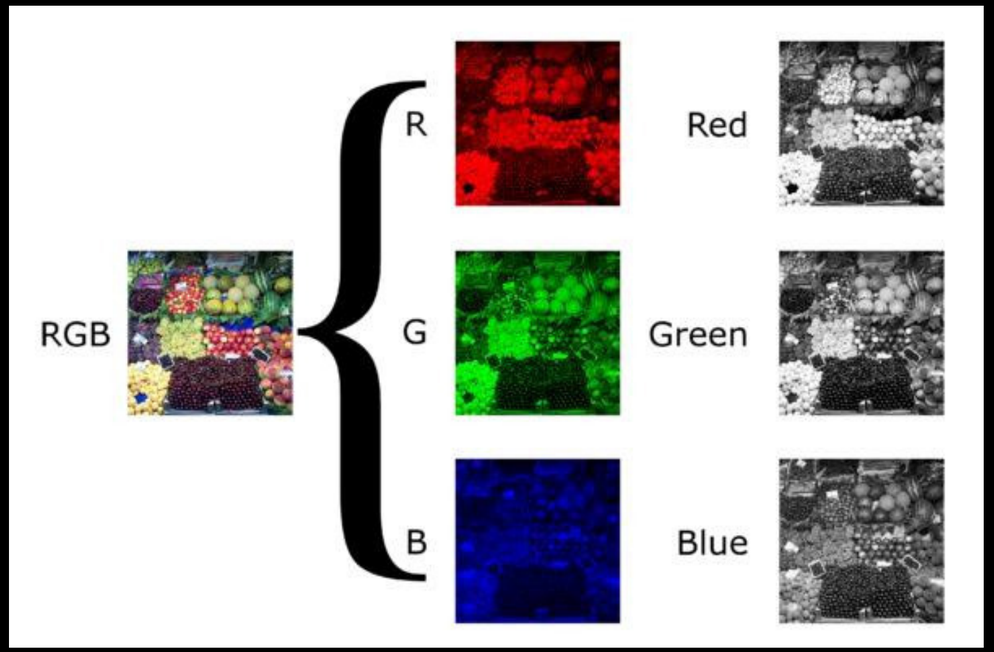
Image[:,:,0] => red_channel

Image[:,:,1] => green_channel

B) Grayscale

- We convert our image to grayscale so that we can abstract away from a Tensor to a Matrix and deal with raw pixel values. (960,540,3) -> (960,540)
- Treat yellow or white lane lines the same
- Pixel list = color channel
- Average color channel values within the pixel list (R + G + B / 3)

gray = cv2.grayscale(image)



output = (960,540) MATRIX

Input = (960,540,3) TENSOR

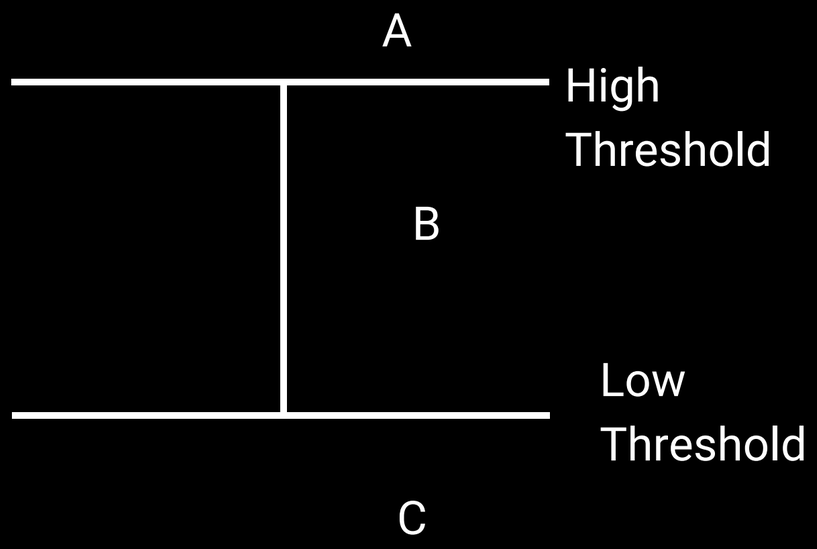
C) Gaussian Blur

- Apply Gaussian Blur to reduce image noise and detail.
- Smooth out the image before applying Canny edge detection so we do not detect faint edges
- Blur_gray = cv2.GaussianBlur(src, ksize, std_dev)
- Src: image to modify
- Ksize = kernel size. Larger kernel size implies averaging/ smoothing over a larger area. (sample size for convolution)
- Return a new matrix Blur_gray

If std_dev = 10

D) Canny Edge Detection

- Used to detect boundaries of an image
- Uses differential of pixel values
- Three regions:
- A: Pixel higher than upper threshold = edge (kept)
- B: Pixel between High and Low is accepted if connected to edge
- C: Pixel below Low rejected
- Recommend 1:2 or 1:3 Low:High threshold ratio



Process so far:

A

Original Color Image (960,540,3)

B

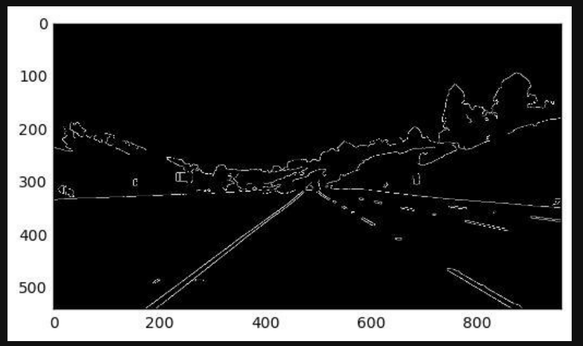
Grayscale (960,540)

C

Gaussian Blur

D

Canny Edges

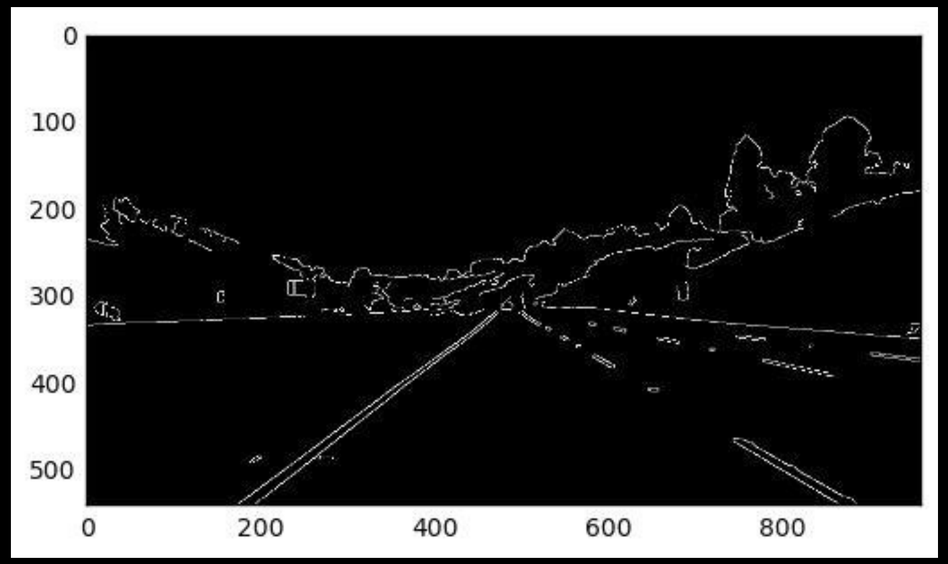


We’re about halfway

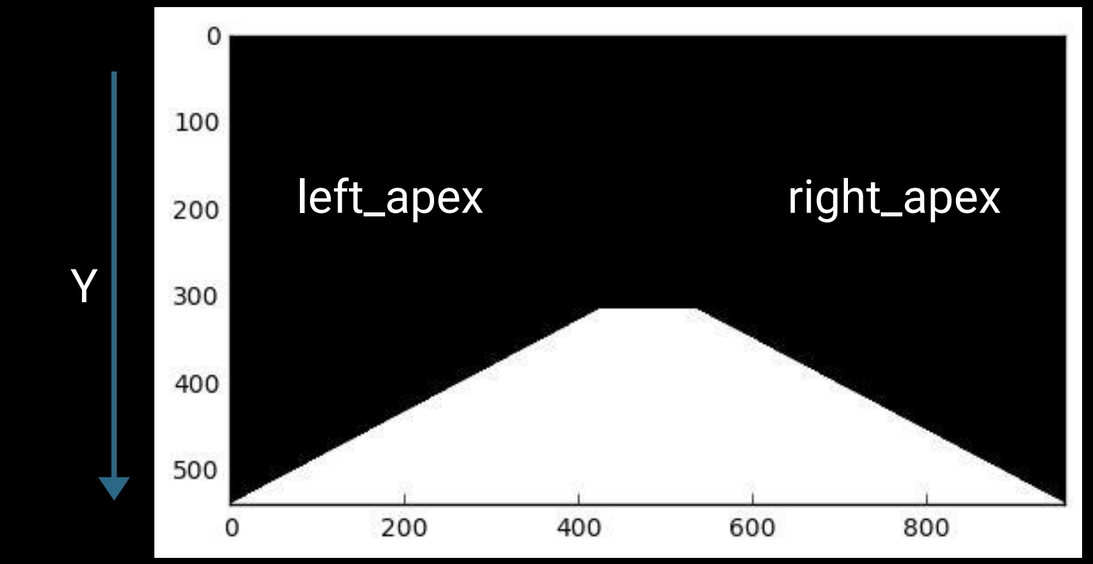
Remember to display gray color map when plotting gray images plt.imshow(gray, cmap=”gray)

E) Masked Image

- We want to narrow down our analysis to a section of the image.
- Apply a trapezoidal mask over edges
- Use bitwise AND to return image only where mask pixels are non-zero.



Apply over edges



Y

left_bottom

X

right_bottom

A

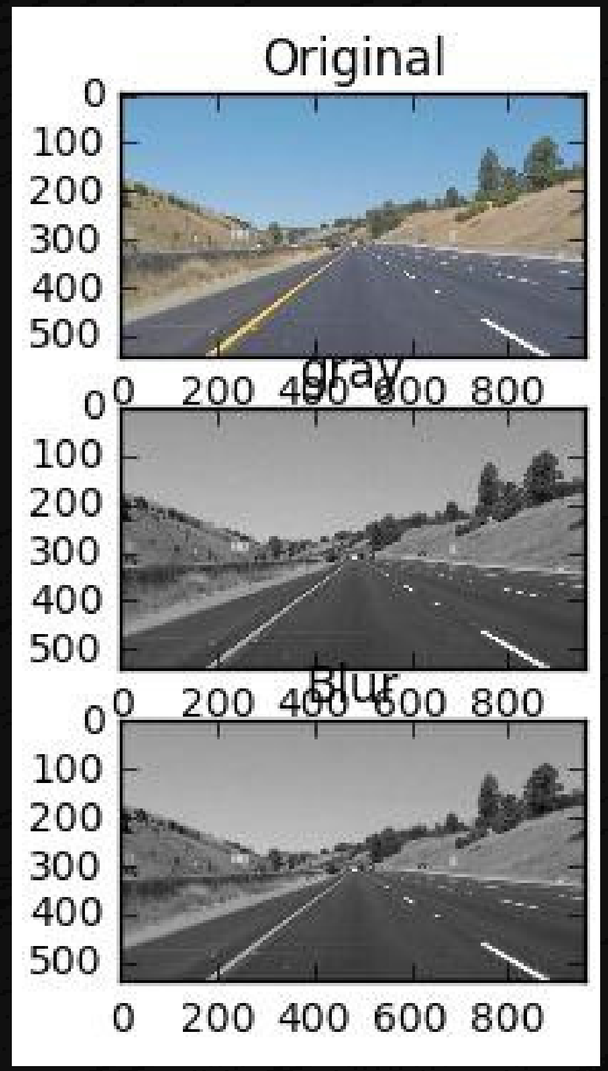
Original Color Image (960,540,3)

B

Grayscale (960,540)

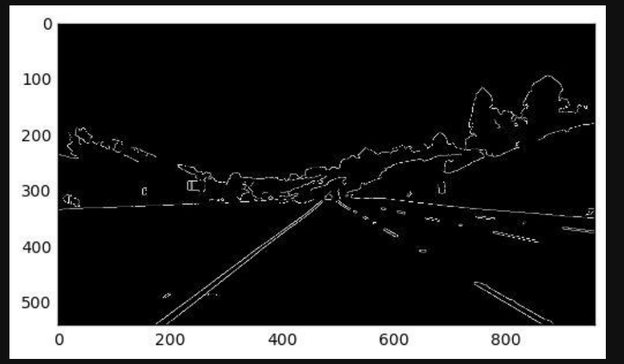
C

Gaussian Blur



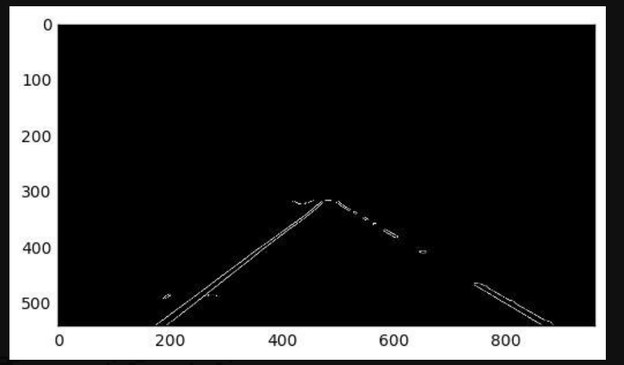
D

Canny Edges



E

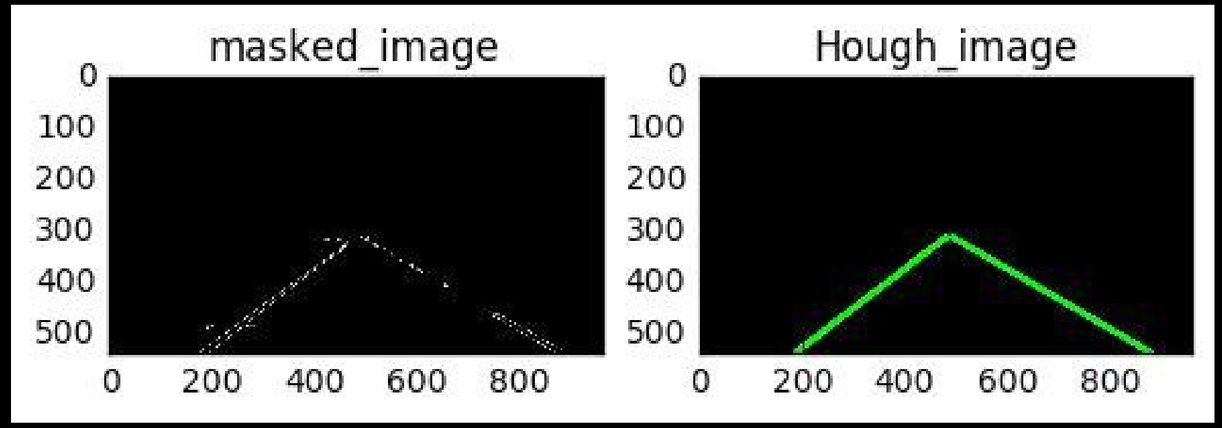
Masked Image

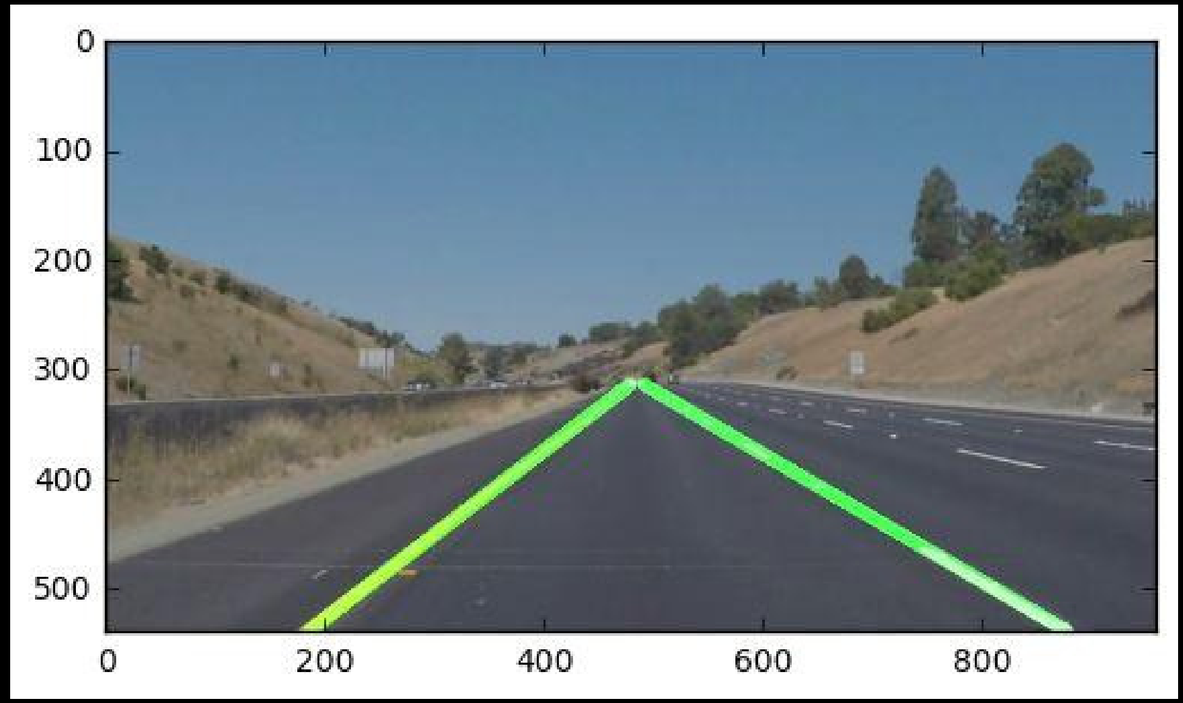


F) Hough Transform

- Theta = pi/180
- Rho = 1.89
- Threshold = 75
- min_line_len = 25
- max_line_gap = 125
- (values dependent on features)

Apply Hough Transform then draw_lines Onto original image





A

Original Color Image (960,540,3)

B

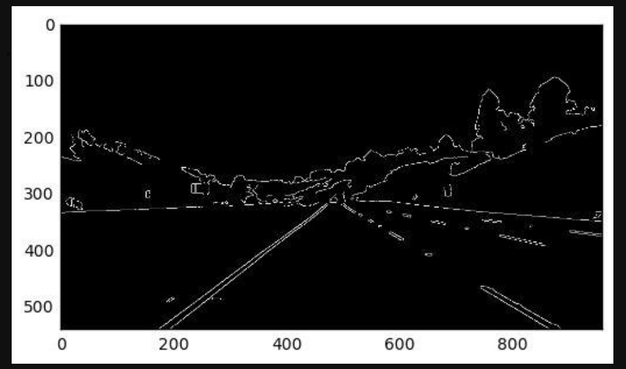
Grayscale (960,540)

C

Gaussian Blur

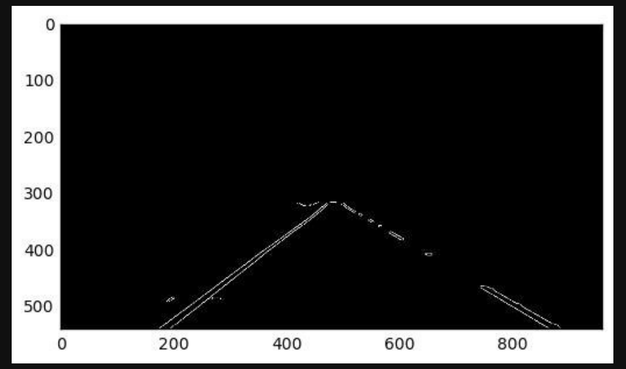
D

Canny Edges



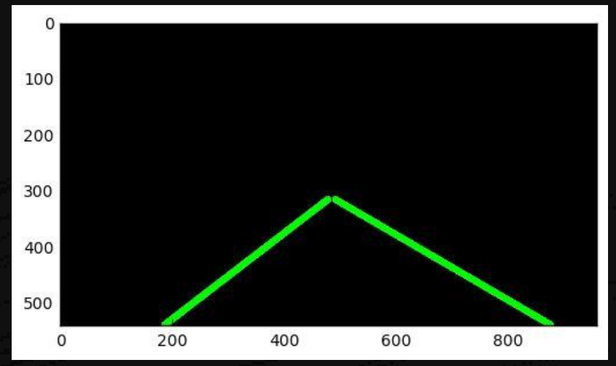
E

Masked Image



F

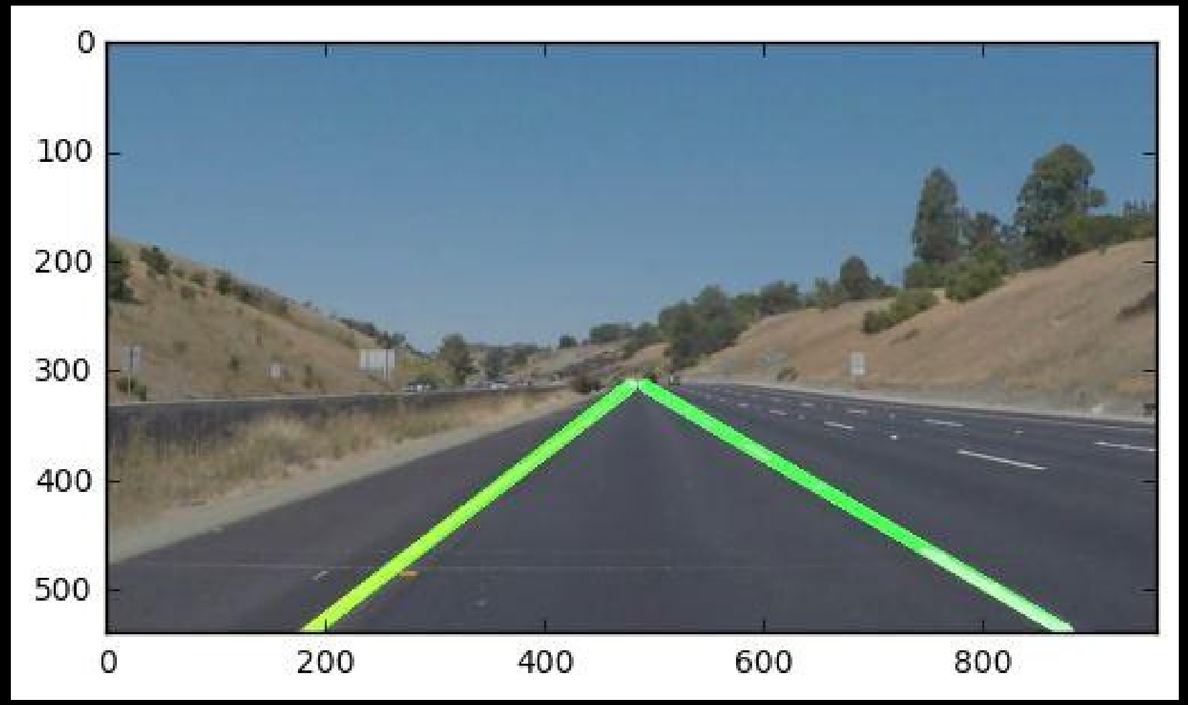
Hough Image

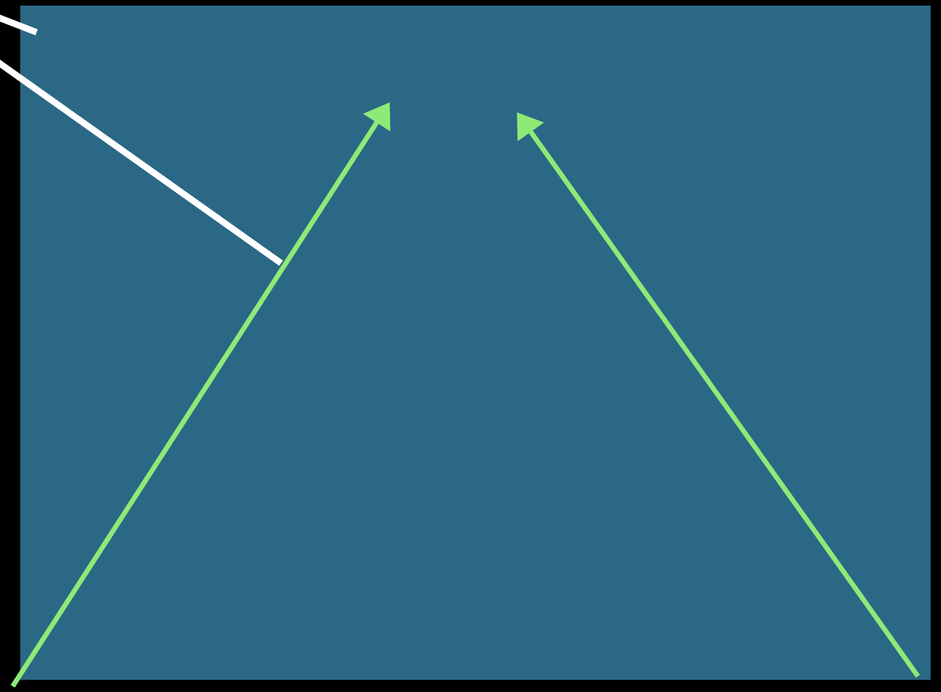


There was a Draw Lines process in between E and F

G) Overlay Hough Image on Original

- cv2.addWeighted(initial_img, α, img, β, λ)
- initial_img: original img to write over
- Img: img to overlay on top of initial_img





A

Original Color Image (960,540,3)

B

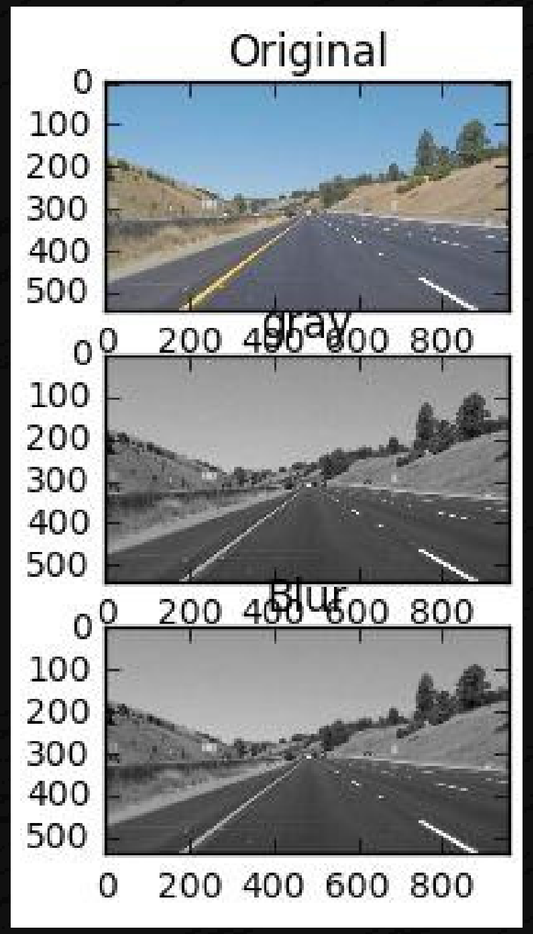
Grayscale (960,540)

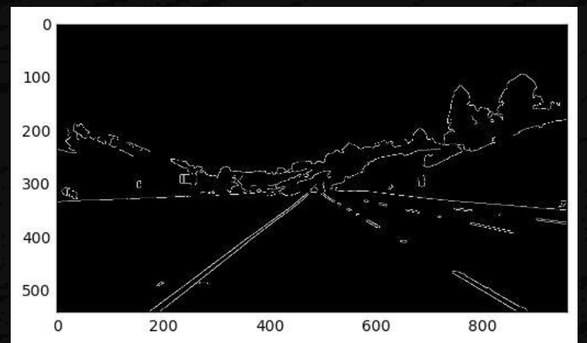
C

Gaussian Blur

D

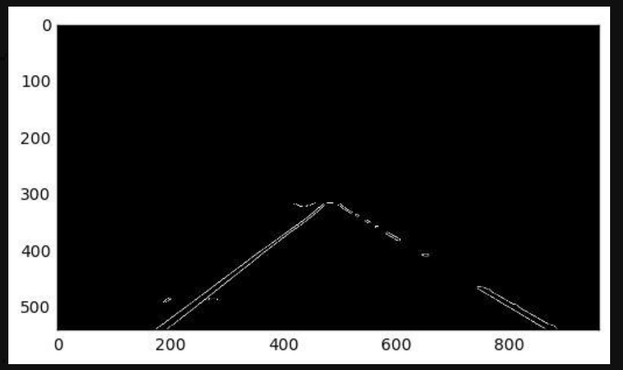
Canny Edges





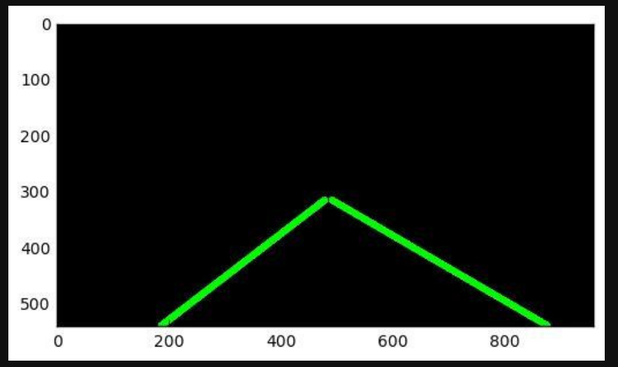
E

Masked Image



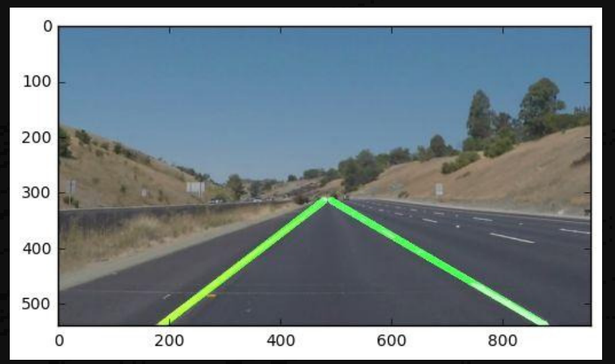
F

Hough Image



G

weighted Image



Future Referance:

Dynamic Masking: Currently we are using static masking system to mask the road but it is limited. Dynamic Masking is used to append the masking area of the frame to use various envioremnt
For Eg. Masking use in Cinematography like Green Screen and Blue Screen

Median Blurr and Bilateral Blurr:
This is highly effective against salt-and-pepper noise in an image. Interestingly, in the above filters, the central element is a newly calculated value which may be a pixel value in the image or a new value.

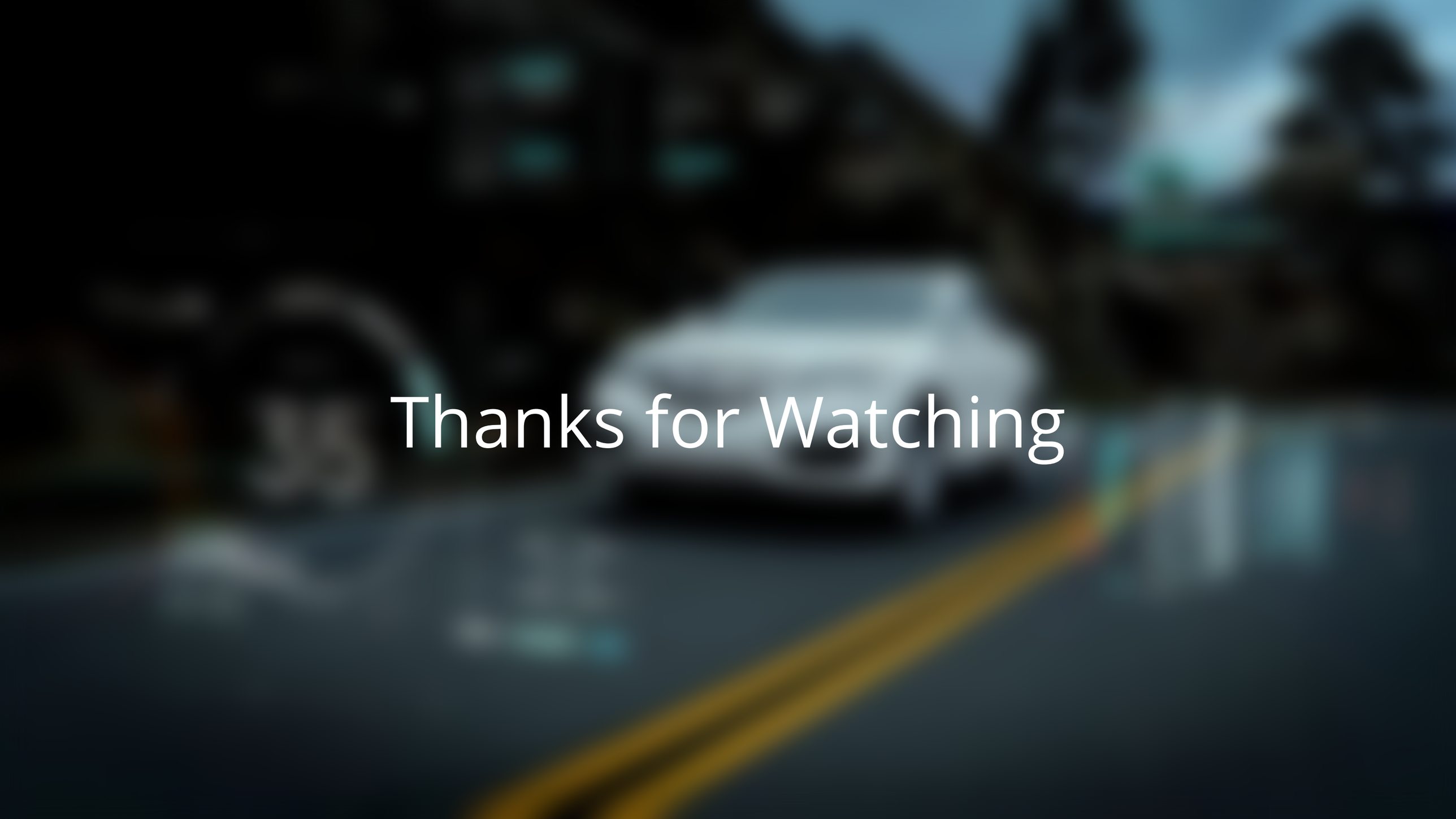
It is more effective than the current blurr technique

Instance Segmentation: Instance segmentation is the task of detecting and delineating each distinct object of interest appearing in an image.
Mostly it is used as object detection

Implementation of CNN: Adding Of convulation neural network can bring more precision and This pixel labeling task is also called as dense prediction.

References:

[github] https://github.com/MaybeShewill-CV/lanenet-lane-detection
[youtube] https://www.youtube.com/watch?v=rvnHikUJ9T0
[analytic vidya] https://www.analyticsvidhya.com/blog/2020/05/tutorial-real-time-lane-detection-opencv/
[medium] https://medium.com/analytics-vidhya/finding-road-lane-lines-using-opencv-python-bd1c8eb34652
[opencv] https://docs.opencv.org/master/d6/d00/tutorial_py_root.html


Thanks for Watching