Lane Line Detection

In [ ]:
```
import numpy as np
import pandas as pd

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
%matplotlib inline
```

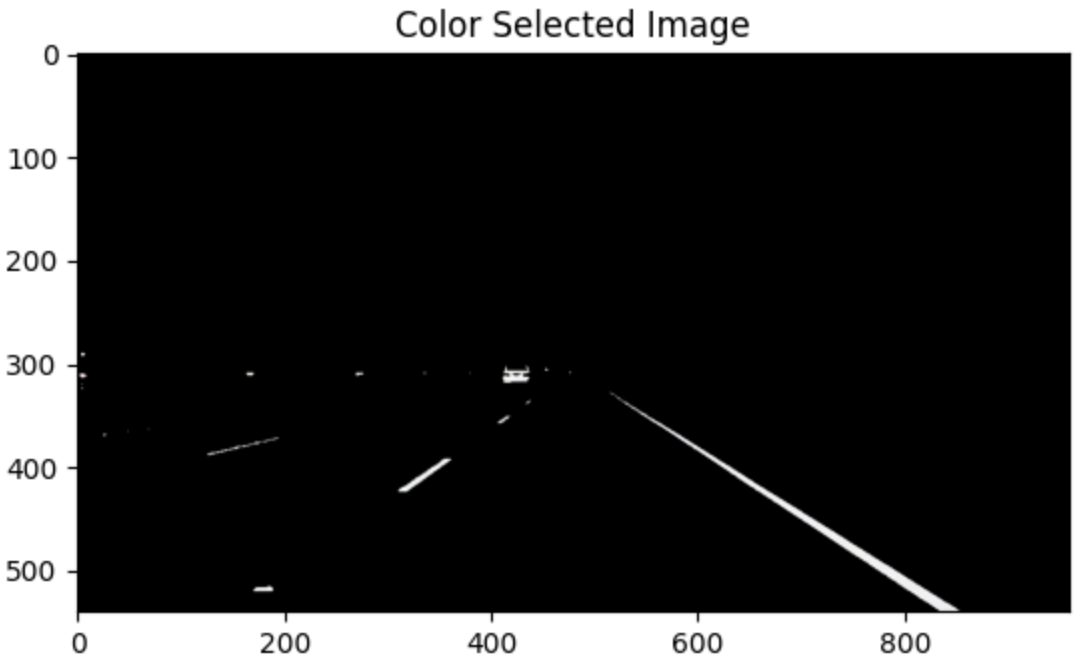
In [ ]:
```
!git clone https://github.com/udacity/CarND-LaneLines-P1.git
```

```
Cloning into 'CarND-LaneLines-P1'...
remote: Enumerating objects: 265, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 265 (delta 2), reused 7 (delta 1), pack-reused 254
Receiving objects: 100% (265/265), 43.45 MiB | 27.18 MiB/s, done.
Resolving deltas: 100% (123/123), done.
```

In [ ]:
```
from distutils.dir_util import copy_tree
import shutil
copy_tree("./CarND-LaneLines-P1/test_images", "./test_images")
copy_tree("./CarND-LaneLines-P1/test_videos", "./test_videos")
shutil.rmtree('./CarND-LaneLines-P1', ignore_errors=False, onerror=None)
```

In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

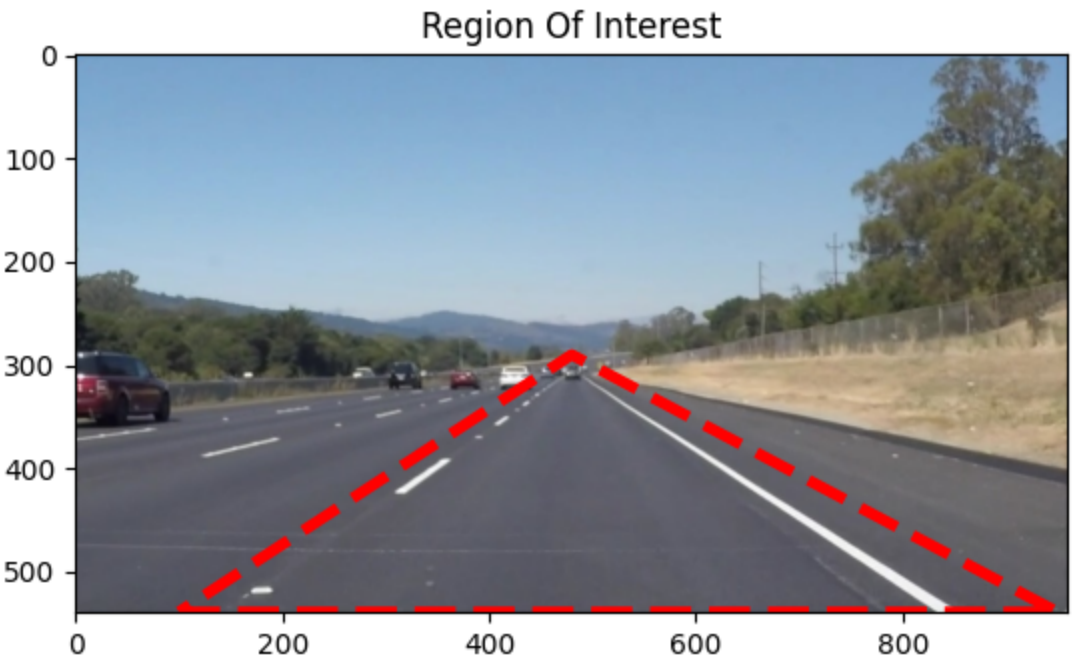
```
import numpy as np

image = mpimg.imread('test_images/solidWhiteRight.jpg')

ysize = image.shape[0]
xsize = image.shape[1]
color_select = np.copy(image)

red_threshold = 200
green_threshold = 200
blue_threshold = 200


rgb_threshold = [red_threshold, green_threshold, blue_threshold]


thresholds = (image[:,:,0] < rgb_threshold[0]) \
            | (image[:,:,1] < rgb_threshold[1]) \
            | (image[:,:,2] < rgb_threshold[2])
color_select[thresholds] = [0,0,0]

# Display the image
plt.imshow(image)
plt.title("Input Image")
plt.show()
plt.imshow(color_select)
plt.title("Color Selected Image")
plt.show()
```

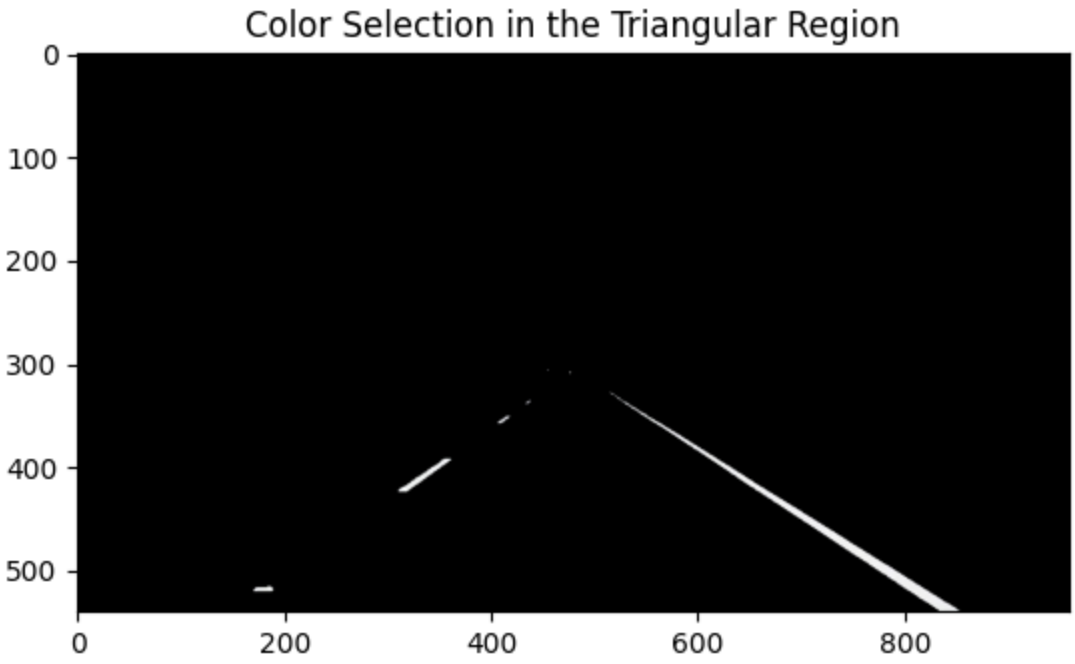


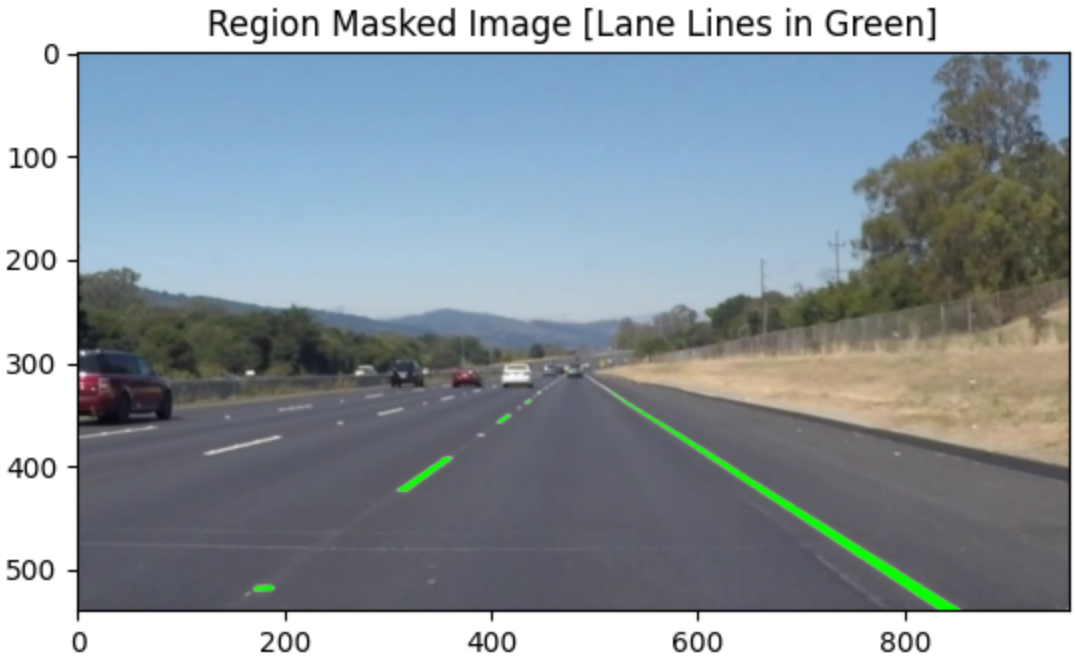
In the above output we can clearly see the lane lines

In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np


image = mpimg.imread('test_images/solidWhiteRight.jpg')

ysize = image.shape[0]
xsize = image.shape[1]
color_select = np.copy(image)
line_image = np.copy(image)


red_threshold = 200
green_threshold = 200
blue_threshold = 200

rgb_threshold = [red_threshold, green_threshold, blue_threshold]


left_bottom = [100, 539]
right_bottom = [950, 539]
apex = [480, 290]

fit_left = np.polyfit((left_bottom[0], apex[0]), (left_bottom[1], apex[1]), 1)
fit_right = np.polyfit((right_bottom[0], apex[0]), (right_bottom[1], apex[1]), 1)
fit_bottom = np.polyfit((left_bottom[0], right_bottom[0]), (left_bottom[1], right_b

color_thresholds = (image[:,:,0] < rgb_threshold[0]) | \
                   (image[:,:,1] < rgb_threshold[1]) | \
                   (image[:,:,2] < rgb_threshold[2])
```

```
XX, YY = np.meshgrid(np.arange(0, xsize), np.arange(0, ysize))
region_thresholds = (YY > (XX*fit_left[0] + fit_left[1])) & \
                    (YY > (XX*fit_right[0] + fit_right[1])) & \
                    (YY < (XX*fit_bottom[0] + fit_bottom[1]))

color_select[color_thresholds | ~region_thresholds] = [0, 0, 0]
# Color pixels red where both color and region selections met
line_image[~color_thresholds & region_thresholds] = [9, 255, 0]

plt.imshow(image)
x = [left_bottom[0], right_bottom[0], apex[0], left_bottom[0]]
y = [left_bottom[1], right_bottom[1], apex[1], left_bottom[1]]
plt.plot(x, y, 'r--', lw=4)
plt.title("Region Of Interest")
plt.show()
plt.imshow(color_select)
plt.title("Color Selection in the Triangular Region")
plt.show()
plt.imshow(line_image)
plt.title("Region Masked Image [Lane Lines in Green]")
plt.show()
```

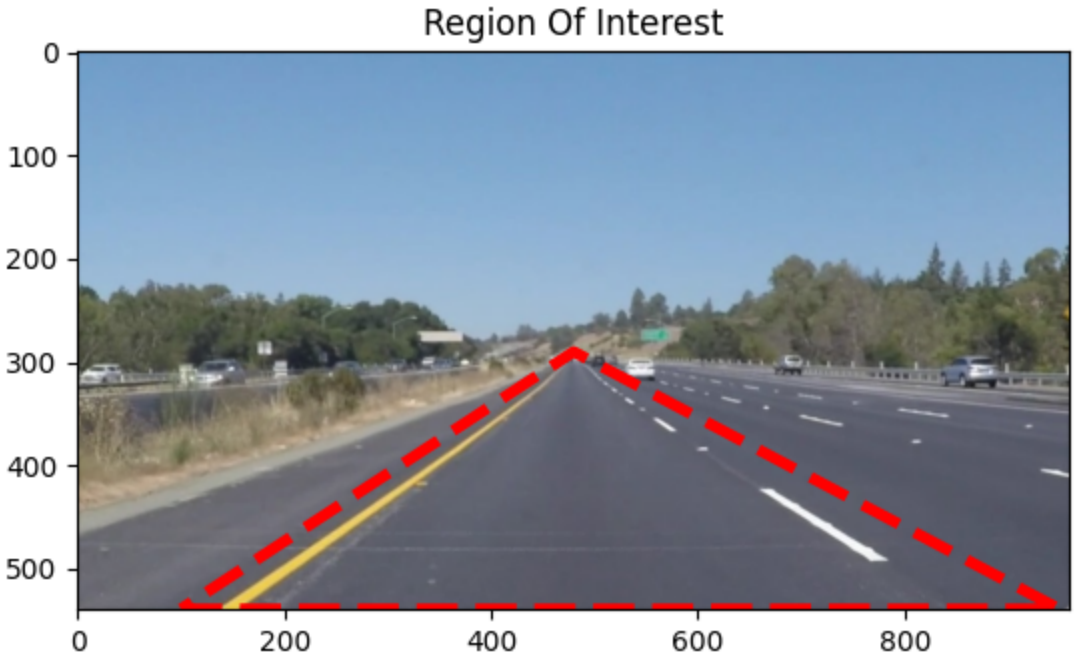


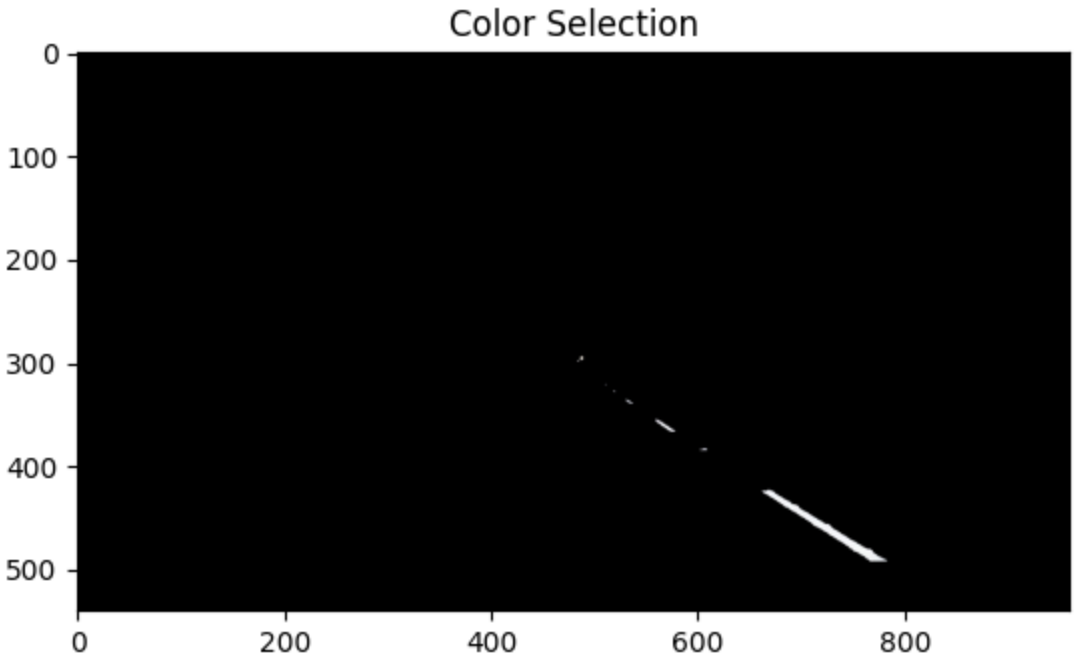


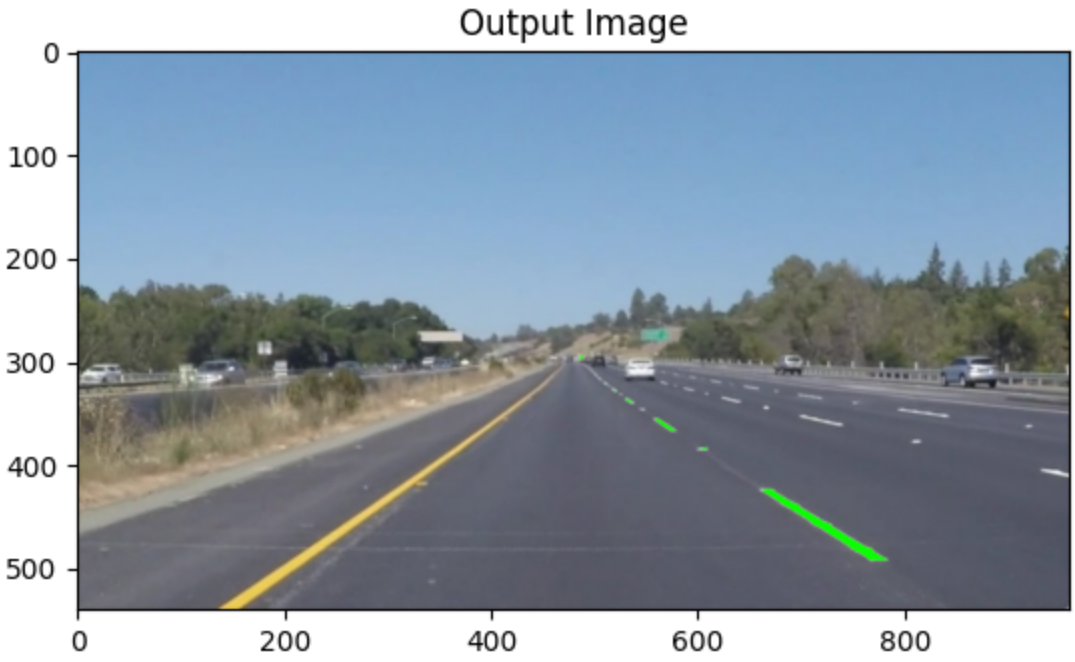


In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

image = mpimg.imread('test_images/solidYellowLeft.jpg')

ysize = image.shape[0]
xsize = image.shape[1]
color_select = np.copy(image)
line_image = np.copy(image)


red_threshold = 200
green_threshold = 200
```

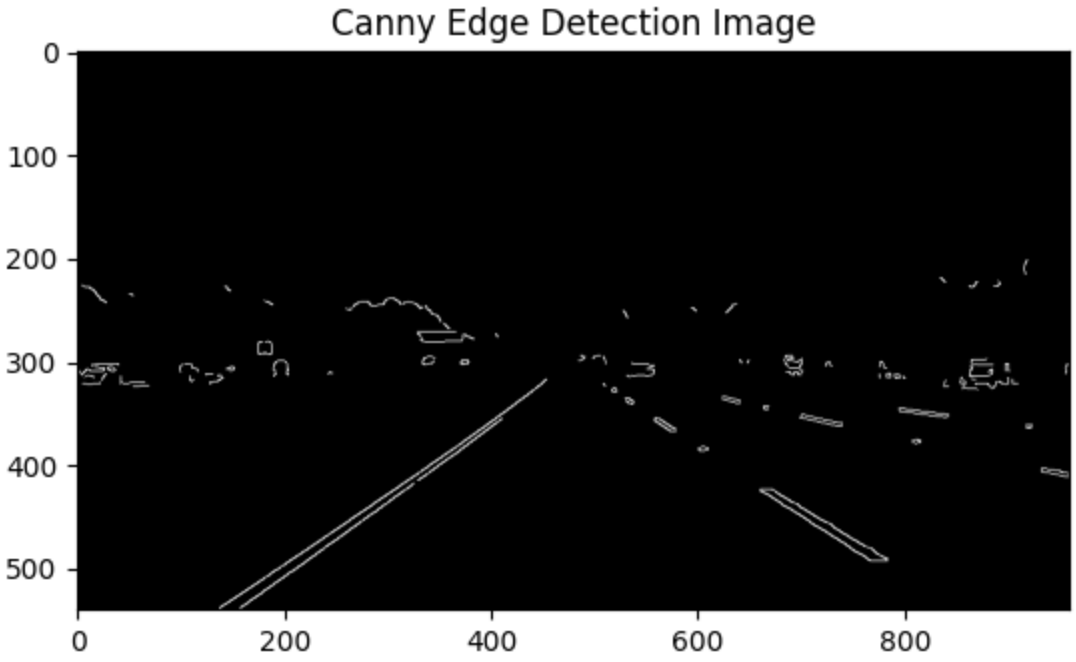
```
blue_threshold = 200

rgb_threshold = [red_threshold, green_threshold, blue_threshold]


left_bottom = [100, 539]
right_bottom = [950, 539]
apex = [480, 290]

fit_left = np.polyfit((left_bottom[0], apex[0]), (left_bottom[1], apex[1]), 1)
fit_right = np.polyfit((right_bottom[0], apex[0]), (right_bottom[1], apex[1]), 1)
fit_bottom = np.polyfit((left_bottom[0], right_bottom[0]), (left_bottom[1], right_b

color_thresholds = (image[:,:,0] < rgb_threshold[0]) | \
                   (image[:,:,1] < rgb_threshold[1]) | \
                   (image[:,:,2] < rgb_threshold[2])

XX, YY = np.meshgrid(np.arange(0, xsize), np.arange(0, ysize))
region_thresholds = (YY > (XX*fit_left[0] + fit_left[1])) & \
                    (YY > (XX*fit_right[0] + fit_right[1])) & \
                    (YY < (XX*fit_bottom[0] + fit_bottom[1]))

color_select[color_thresholds | ~region_thresholds] = [0, 0, 0]
line_image[~color_thresholds & region_thresholds] = [9, 255, 0]

plt.imshow(image)
x = [left_bottom[0], right_bottom[0], apex[0], left_bottom[0]]
y = [left_bottom[1], right_bottom[1], apex[1], left_bottom[1]]
plt.plot(x, y, 'r--', lw=4)
plt.title("Region Of Interest")
plt.show()
plt.imshow(color_select)
plt.title("Color Selection")
plt.show()
plt.imshow(line_image)
plt.title("Output Image")
plt.show()
```


Region Of Interest
0
100
200
300
400
500
0
200
400
600
800


Color Selection
0
100
200
300
400
500
0
200
400
600
800

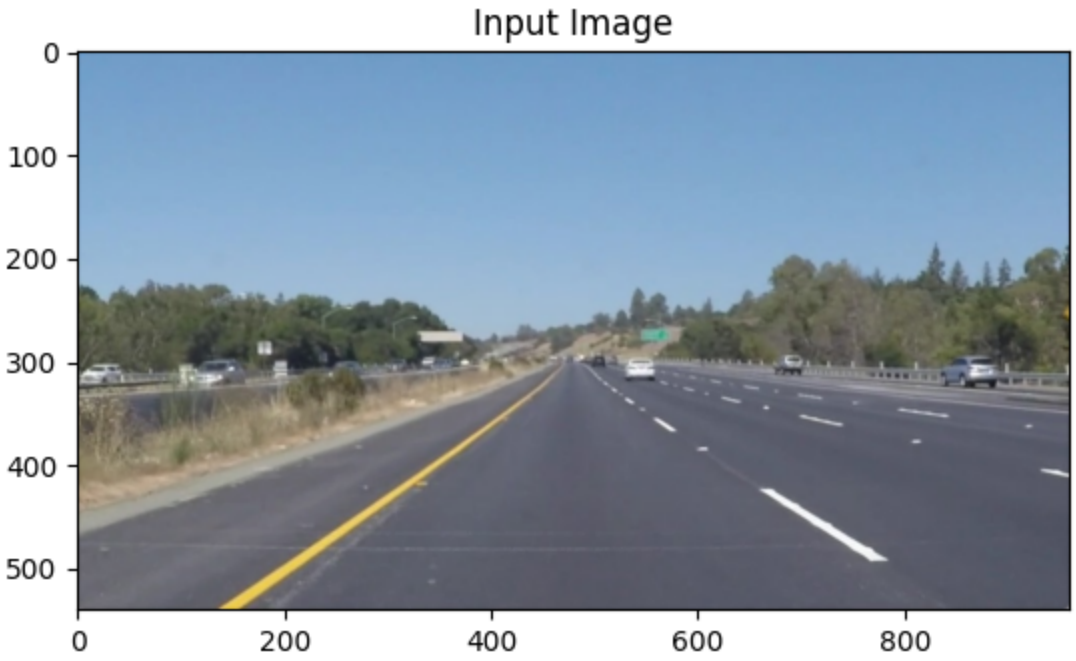


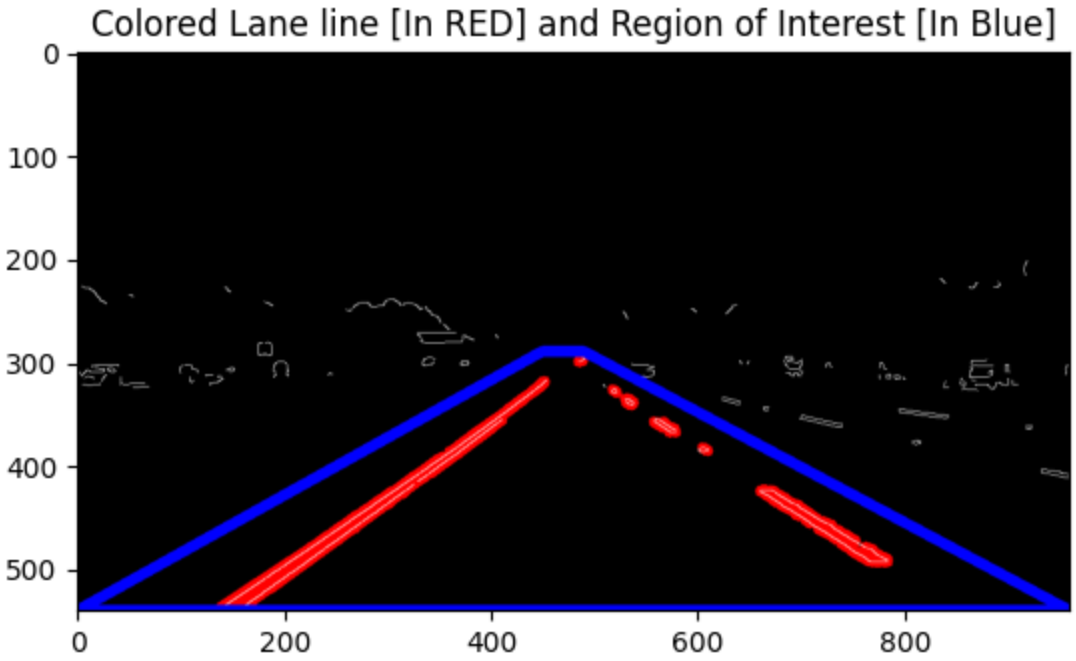
In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2


image = mpimg.imread('test_images/solidYellowLeft.jpg')
gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)

blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)

low_threshold = 180
high_threshold = 240
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)

plt.imshow(edges, cmap='Greys_r')
plt.title("Canny Edge Detection Image")
plt.show()
```



In [ ]:
```
image = mpimg.imread('test_images/solidYellowLeft.jpg')
gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)

kernel_size = 5
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)

low_threshold = 180
high_threshold = 240
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)

mask = np.zeros_like(edges)
ignore_mask_color = 255

imshape = image.shape
vertices = np.array([[(0,imshape[0]),(450, 290), (490, 290), (imshape[1],imshape[0]
cv2.fillPoly(mask, vertices, ignore_mask_color)
masked_edges = cv2.bitwise_and(edges, mask)


rho = 1
theta = np.pi/180
threshold = 2
min_line_length = 4
max_line_gap = 5
line_image = np.copy(image)*0


lines = cv2.HoughLinesP(masked_edges, rho, theta, threshold, np.array([]),
                        min_line_length, max_line_gap)

for line in lines:
    for x1,y1,x2,y2 in line:
        cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)

color_edges = np.dstack((edges, edges, edges))
```

```
lines_edges = cv2.addWeighted(color_edges, 0.8, line_image, 1, 0)
lines_edges = cv2.polylines(lines_edges,vertices, True, (0,0,255), 10)
plt.imshow(image)
plt.title("Input Image")
plt.show()
plt.imshow(lines_edges)
plt.title("Colored Lane line [In RED] and Region of Interest [In Blue]")
plt.show()
```





In [ ]:
```
import math

def grayscale(img):
    return cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
```

```
def canny(img, low_threshold, high_threshold):
    return cv2.Canny(img, low_threshold, high_threshold)

def gaussian_blur(img, kernel_size):
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)

def region_of_interest(img, vertices):
    mask = np.zeros_like(img)

    if len(img.shape) > 2:
        channel_count = img.shape[2]
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    cv2.fillPoly(mask, vertices, ignore_mask_color)

    masked_image = cv2.bitwise_and(img, mask)
    return masked_image


def draw_lines(img, lines, color=[255, 0, 0], thickness=10):
    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(img, (x1, y1), (x2, y2), color, thickness)

def slope_lines(image,lines):

    img = image.copy()
    poly_vertices = []
    order = [0,1,3,2]

    left_lines = [] # Like /
    right_lines = [] # Like \
    for line in lines:
        for x1,y1,x2,y2 in line:

            if x1 == x2:
                pass #Vertical Lines
            else:
                m = (y2 - y1) / (x2 - x1)
                c = y1 - m * x1

                if m < 0:
                    left_lines.append((m,c))
                elif m >= 0:
                    right_lines.append((m,c))

    left_line = np.mean(left_lines, axis=0)
    right_line = np.mean(right_lines, axis=0)

    for slope, intercept in [left_line, right_line]:

        rows, cols = image.shape[:2]
```

```
        y1= int(rows)

        y2= int(rows*0.6)

        x1=int((y1-intercept)/slope)
        x2=int((y2-intercept)/slope)
        poly_vertices.append((x1, y1))
        poly_vertices.append((x2, y2))
        draw_lines(img, np.array([[[x1,y1,x2,y2]]]))

    poly_vertices = [poly_vertices[i] for i in order]
    cv2.fillPoly(img, pts = np.array([poly_vertices],'int32'), color = (0,255,0))
    return cv2.addWeighted(image,0.7,img,0.4,0.)

def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), minLineLength
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    line_img = slope_lines(line_img,lines)
    return line_img

def weighted_img(img, initial_img, α=0.1, β=1., γ=0.):
    lines_edges = cv2.addWeighted(initial_img, α, img, β, γ)
    return lines_edges
def get_vertices(image):
    rows, cols = image.shape[:2]
    bottom_left  = [cols*0.15, rows]
    top_left     = [cols*0.45, rows*0.6]
    bottom_right = [cols*0.95, rows]
    top_right    = [cols*0.55, rows*0.6]

    ver = np.array([[bottom_left, top_left, top_right, bottom_right]], dtype=np.int
    return ver
```

In [ ]:

```
def lane_finding_pipeline(image):

    gray_img = grayscale(image)

    smoothed_img = gaussian_blur(img = gray_img, kernel_size = 5)

    canny_img = canny(img = smoothed_img, low_threshold = 180, high_threshold = 240

    masked_img = region_of_interest(img = canny_img, vertices = get_vertices(image)

    houghed_lines = hough_lines(img = masked_img, rho = 1, theta = np.pi/180, thres

    output = weighted_img(img = houghed_lines, initial_img = image, α=0.8, β=1., γ=

    return output
```

In [ ]:

```
for image_path in list(os.listdir('./test_images')):
    fig = plt.figure(figsize=(20, 10))
    image = mpimg.imread(f'./test_images/{image_path}')
    ax = fig.add_subplot(1, 2, 1,xticks=[], yticks=[])
    plt.imshow(image)
    ax.set_title("Input Image")
```

```
    ax = fig.add_subplot(1, 2, 2,xticks=[], yticks=[])
    plt.imshow(lane_finding_pipeline(image))
    ax.set_title("Output Image [Lane Line Detected]")
    plt.show()
```

In [ ]:
```
fig = plt.figure(figsize=(20, 10))
image = mpimg.imread('/content/sample_data/Highway.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGRA2RGB)
image = (image * (255.0 / image.max())).astype(np.uint8)
ax = fig.add_subplot(1, 2, 1,xticks=[], yticks=[])
plt.imshow(image)
ax.set_title("Input Image")
ax = fig.add_subplot(1, 2, 2,xticks=[], yticks=[])
plt.imshow(lane_finding_pipeline(image))
ax.set_title("Output Image [Lane Line Detected]")
plt.show()
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Install moviepy package

In [ ]:
```
!pip install moviepy
```

```
Requirement already satisfied: moviepy in /usr/local/lib/python3.10/dist-packages
(1.0.3)
Requirement already satisfied: decorator<5.0,>=4.0.2 in /usr/local/lib/python3.10/di
st-packages (from moviepy) (4.4.2)
Requirement already satisfied: tqdm<5.0,>=4.11.2 in /usr/local/lib/python3.10/dist-p
ackages (from moviepy) (4.66.4)
Requirement already satisfied: requests<3.0,>=2.8.1 in /usr/local/lib/python3.10/dis
t-packages (from moviepy) (2.31.0)
Requirement already satisfied: proglog<=1.0.0 in /usr/local/lib/python3.10/dist-pack
ages (from moviepy) (0.1.10)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packa
ges (from moviepy) (1.26.4)
Requirement already satisfied: imageio<3.0,>=2.5 in /usr/local/lib/python3.10/dist-p
ackages (from moviepy) (2.34.2)
Requirement already satisfied: imageio-ffmpeg>=0.2.0 in /usr/local/lib/python3.10/di
st-packages (from moviepy) (0.5.1)
Requirement already satisfied: pillow>=8.3.2 in /usr/local/lib/python3.10/dist-packa
ges (from imageio<3.0,>=2.5->moviepy) (9.4.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages
(from imageio-ffmpeg>=0.2.0->moviepy) (71.0.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.1
0/dist-packages (from requests<3.0,>=2.8.1->moviepy) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packag
es (from requests<3.0,>=2.8.1->moviepy) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-
packages (from requests<3.0,>=2.8.1->moviepy) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-
packages (from requests<3.0,>=2.8.1->moviepy) (2024.7.4)
```

In [ ]:
```
from moviepy.editor import VideoFileClip
from IPython.display import HTML
```

In [ ]:
```
# !wget -O "test_videos/jaipurHighway.mp4" "https://r6---sn-gwpa-ccpe.googlevideo.c
```

Test with Video Clip 1 [Solid White Lane Lines]

In [ ]:
```
white_output = './solidWhiteRight.mp4'
clip1 = VideoFileClip("test_videos/solidWhiteRight.mp4")
white_clip = clip1.fl_image(lane_finding_pipeline)
%time white_clip.write_videofile(white_output, audio=False)
```

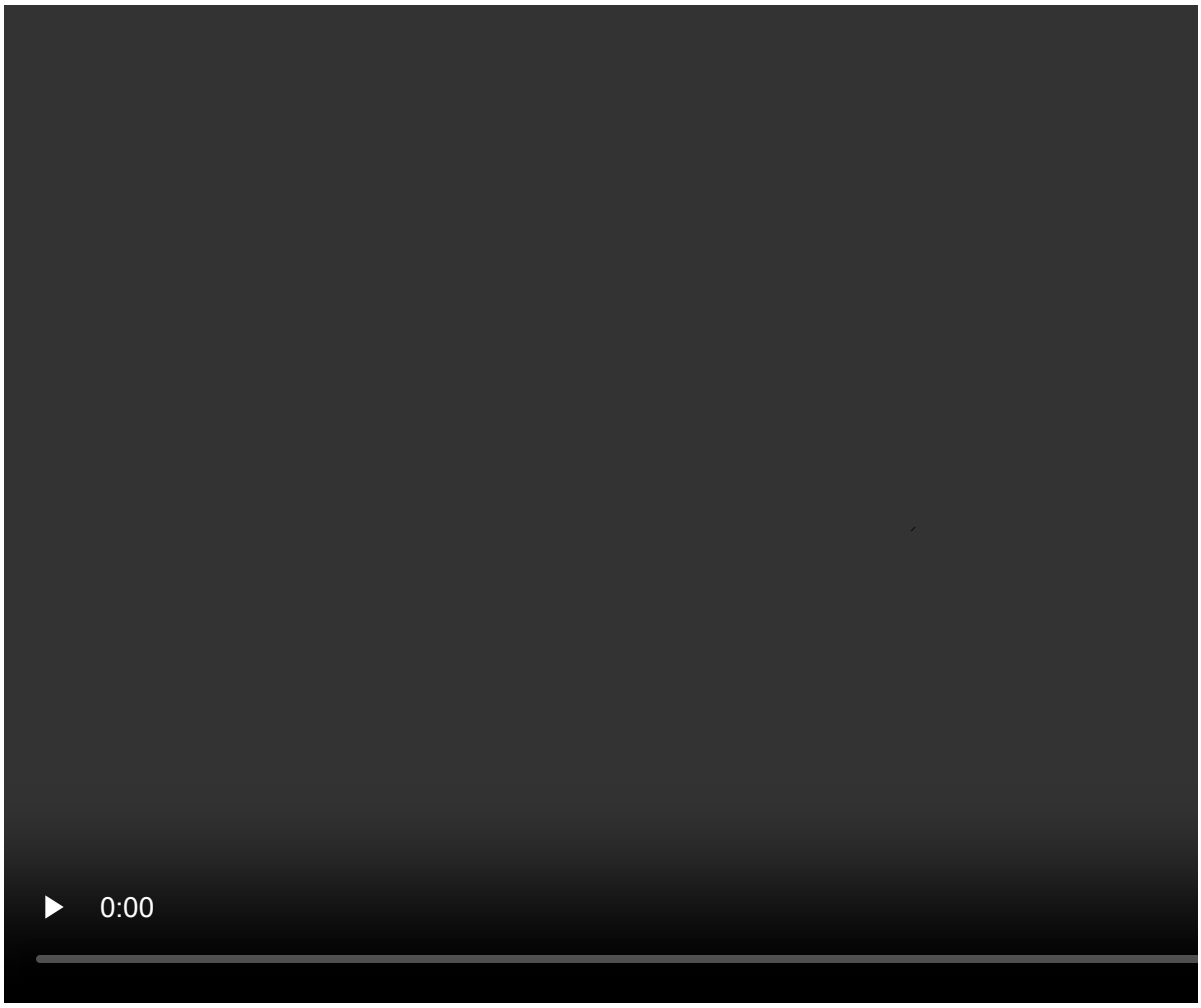
```
Moviepy - Building video ./solidWhiteRight.mp4.
Moviepy - Writing video ./solidWhiteRight.mp4

Moviepy - Done !
Moviepy - video ready ./solidWhiteRight.mp4
CPU times: user 2.56 s, sys: 427 ms, total: 2.99 s
Wall time: 25.4 s
```

Output Video

In [ ]:
```
HTML("""
<video width="960" height="500" controls>
```

```
  <source src="{0}">
</video>
""".format(white_output))
```

Out[ ]:



Test with Video Clip 2 [With Yellow Lane Lines]

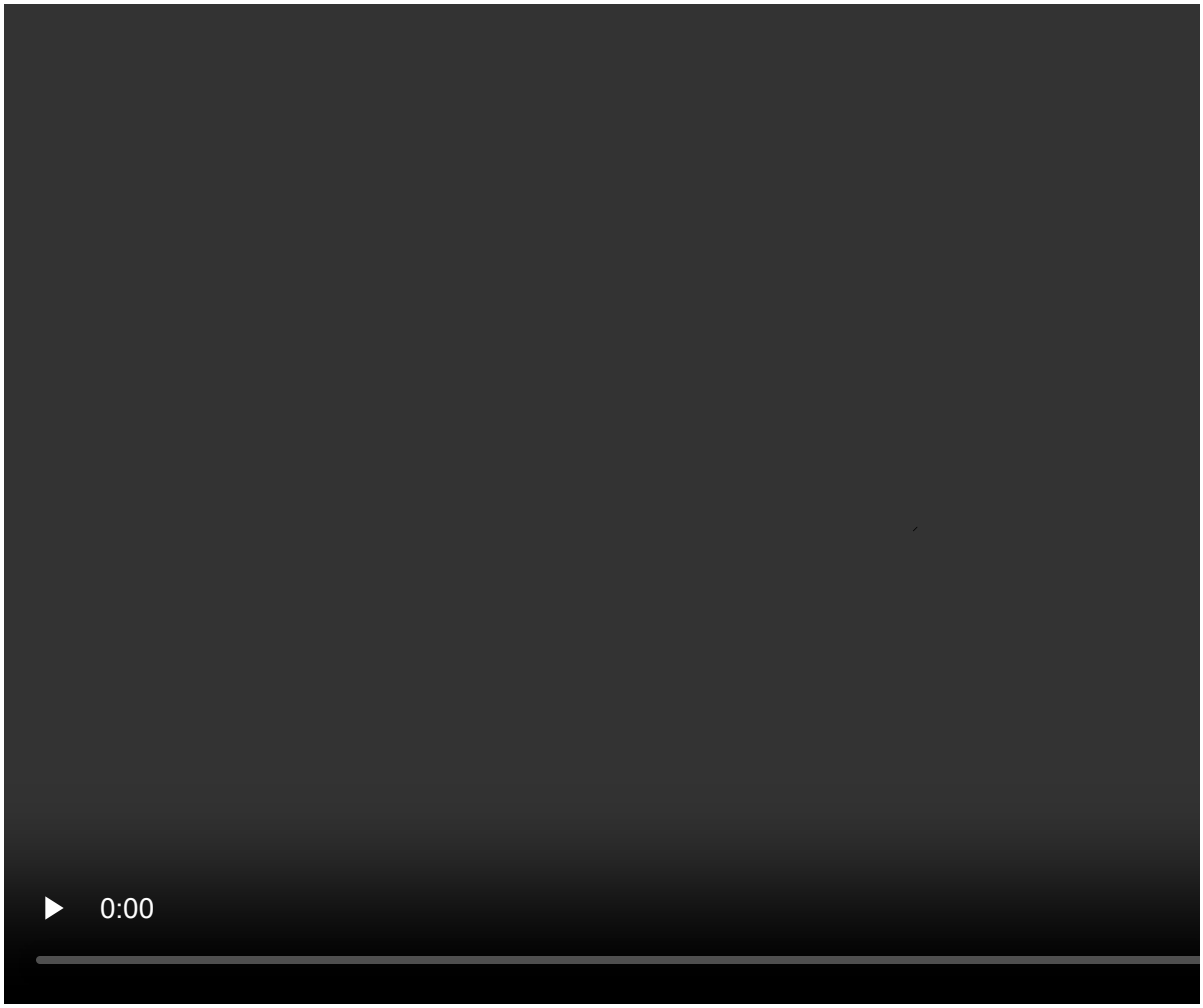
In [ ]:
```
yellow_output = './solidYellowLeft.mp4'
clip2 = VideoFileClip('test_videos/solidYellowLeft.mp4')
yellow_clip = clip2.fl_image(lane_finding_pipeline)
%time yellow_clip.write_videofile(yellow_output, audio=False)
```

```
Moviepy - Building video ./solidYellowLeft.mp4.
Moviepy - Writing video ./solidYellowLeft.mp4

Moviepy - Done !
Moviepy - video ready ./solidYellowLeft.mp4
CPU times: user 7.59 s, sys: 1.25 s, total: 8.85 s
Wall time: 40.6 s
```

Output

In [ ]:
```
HTML("""
<video width="960" height="500" controls>
  <source src="{0}">
</video>
""".format(yellow_output))
```

Out[ ]:



Lane Departure Warning Video Clip

In [ ]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from moviepy.editor import VideoFileClip
from IPython.display import HTML
import os


ym_per_pix = 30/720
xm_per_pix = 3.7/700

def fit_polynomial_and_calculate_curvature(left_fitx, right_fitx, ploty):
    if len(left_fitx) == 0 or len(right_fitx) == 0:
        return None, None
```

```
    left_fit_cr = np.polyfit(ploty * ym_per_pix, left_fitx * xm_per_pix, 2)
    right_fit_cr = np.polyfit(ploty * ym_per_pix, right_fitx * xm_per_pix, 2)

    y_eval = np.max(ploty)
    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)
    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])*

    return left_curverad, right_curverad

def calculate_vehicle_position_and_lane_departure(left_fitx, right_fitx, ploty, ima
    if len(left_fitx) == 0 or len(right_fitx) == 0:
        return "No lane lines detected", 0

    lane_center_position = (left_fitx[-1] + right_fitx[-1]) / 2
    car_center_position = image.shape[1] / 2

    center_offset = (car_center_position - lane_center_position) * xm_per_pix

    lane_width = np.mean(right_fitx - left_fitx)
    lane_width_meters = lane_width * xm_per_pix
    safe_margin = lane_width_meters / 4

    if center_offset > safe_margin:
        warning_text = "Right Lane Departure!"
    elif center_offset < -safe_margin:
        warning_text = "Left Lane Departure!"
    else:
        warning_text = "Centered in Lane"

    return warning_text, center_offset

def lane_finding_pipeline_with_departure_warning(image):
    image = np.copy(image)

    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    blur_gray = cv2.GaussianBlur(gray, (5, 5), 0)

    edges = cv2.Canny(blur_gray, 50, 150)

    mask = np.zeros_like(edges)
    ignore_mask_color = 255

    imshape = image.shape
    vertices = np.array([[(0,imshape[0]),(450, 320), (490, 320), (imshape[1],imshap
    cv2.fillPoly(mask, vertices, ignore_mask_color)
    masked_edges = cv2.bitwise_and(edges, mask)

    rho = 2
    theta = np.pi/180
    threshold = 15
    min_line_len = 40
    max_line_gap = 20
    line_image = np.copy(image)*0
```

```
    lines = cv2.HoughLinesP(masked_edges, rho, theta, threshold, np.array([]), minL

    left_lines_x = []
    left_lines_y = []
    right_lines_x = []
    right_lines_y = []

    if lines is not None:
        for line in lines:
            for x1,y1,x2,y2 in line:
                slope = (y2 - y1) / (x2 - x1)
                if slope < 0:
                    left_lines_x.extend([x1, x2])
                    left_lines_y.extend([y1, y2])
                else:
                    right_lines_x.extend([x1, x2])
                    right_lines_y.extend([y1, y2])

    if len(left_lines_x) > 0 and len(left_lines_y) > 0:
        left_fit = np.polyfit(left_lines_y, left_lines_x, 2)
    else:
        left_fit = None

    if len(right_lines_x) > 0 and len(right_lines_y) > 0:
        right_fit = np.polyfit(right_lines_y, right_lines_x, 2)
    else:
        right_fit = None

    ploty = np.linspace(0, imshape[0]-1, imshape[0])

    if left_fit is not None:
        left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
    else:
        left_fitx = []

    if right_fit is not None:
        right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]
    else:
        right_fitx = []

    left_curverad, right_curverad = fit_polynomial_and_calculate_curvature(left_fit
    warning_text, center_offset = calculate_vehicle_position_and_lane_departure(lef

    for i in range(len(ploty)):
        if left_fit is not None and len(left_fitx) > i:
            cv2.circle(line_image, (int(left_fitx[i]), int(ploty[i])), 5, (255, 0,
        if right_fit is not None and len(right_fitx) > i:
            cv2.circle(line_image, (int(right_fitx[i]), int(ploty[i])), 5, (255, 0,

    if left_curverad is not None and right_curverad is not None:
        cv2.putText(image, 'Left curve radius: {:.2f}m'.format(left_curverad), (50,
        cv2.putText(image, 'Right curve radius: {:.2f}m'.format(right_curverad), (5

    cv2.putText(image, warning_text, (50, 150), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,

    result = cv2.addWeighted(image, 0.8, line_image, 1, 0)
```

```
    return result

input_video_path = "/content/solidWhiteRight.mp4"

output_video_path = './solidYellowLeft_with_departure_warning.mp4'

clip = VideoFileClip(input_video_path)
processed_clip = clip.fl_image(lane_finding_pipeline_with_departure_warning)
processed_clip.write_videofile(output_video_path, audio=False)

if os.path.exists(output_video_path):
    print(f"Video saved successfully at {output_video_path}")

    HTML(f"""
    <video width="960" height="500" controls>
      <source src="{output_video_path}">
    </video>
    """)
else:
    print("Failed to save the video.")
```

```
Moviepy - Building video ./solidYellowLeft_with_departure_warning.mp4.
Moviepy - Writing video ./solidYellowLeft_with_departure_warning.mp4
```

```
Moviepy - Done !
Moviepy - video ready ./solidYellowLeft_with_departure_warning.mp4
Video saved successfully at ./solidYellowLeft_with_departure_warning.mp4
```

In [ ]:

```
import cv2
import numpy as np
from moviepy.editor import VideoFileClip

def calculate_iou(ground_truth, prediction):
    intersection = np.logical_and(ground_truth, prediction)
    union = np.logical_or(ground_truth, prediction)
    iou_score = np.sum(intersection) / np.sum(union)
    return iou_score

def evaluate_lane_detection_video(ground_truth_video_path, prediction_video_path):
    gt_video = VideoFileClip(ground_truth_video_path)
    pred_video = VideoFileClip(prediction_video_path)

    if gt_video.duration != pred_video.duration or gt_video.fps != pred_video.fps:
        raise ValueError("Ground truth and prediction videos must have the same dur

    iou_scores = []

    for frame_gt, frame_pred in zip(gt_video.iter_frames(), pred_video.iter_frames(
        ground_truth = cv2.cvtColor(frame_gt, cv2.COLOR_RGB2GRAY)
        prediction = cv2.cvtColor(frame_pred, cv2.COLOR_RGB2GRAY)

        _, ground_truth_bin = cv2.threshold(ground_truth, 127, 255, cv2.THRESH_BINA
        _, prediction_bin = cv2.threshold(prediction, 127, 255, cv2.THRESH_BINARY)

        ground_truth_bin = ground_truth_bin / 255
        prediction_bin = prediction_bin / 255
```

```
        iou = calculate_iou(ground_truth_bin, prediction_bin)
        iou_scores.append(iou)

    mean_iou = np.mean(iou_scores)
    return mean_iou, iou_scores

ground_truth_video_path = '/content/test_videos/solidWhiteRight.mp4'
prediction_video_path = '/content/solidWhiteRight.mp4'

mean_iou, iou_scores = evaluate_lane_detection_video(ground_truth_video_path, predi

print(f'Mean IoU: {mean_iou}')
for i, iou in enumerate(iou_scores):
    print(f'Frame {i+1}: IoU = {iou}')
```

```
Mean IoU: 0.5218120409871114
Frame 1: IoU = 0.5038732977701202
Frame 2: IoU = 0.5003833602013132
Frame 3: IoU = 0.49779584098891755
Frame 4: IoU = 0.48991624103612463
Frame 5: IoU = 0.4896665712696274
Frame 6: IoU = 0.48780566239178813
Frame 7: IoU = 0.4894085845872006
Frame 8: IoU = 0.4880011491317671
Frame 9: IoU = 0.4906041747160497
Frame 10: IoU = 0.49334200136362916
Frame 11: IoU = 0.4934887059905164
Frame 12: IoU = 0.4947390592237008
Frame 13: IoU = 0.4936057122931791
Frame 14: IoU = 0.49496899644056497
Frame 15: IoU = 0.49806785516163193
Frame 16: IoU = 0.5041415187252203
Frame 17: IoU = 0.5047760553694345
Frame 18: IoU = 0.5063022163372112
Frame 19: IoU = 0.503794361943116
Frame 20: IoU = 0.5109479649665121
Frame 21: IoU = 0.5150690869342615
Frame 22: IoU = 0.5174553502561083
Frame 23: IoU = 0.5196947200181579
Frame 24: IoU = 0.5193239447240918
Frame 25: IoU = 0.5227698144495673
Frame 26: IoU = 0.5289982203883649
Frame 27: IoU = 0.5302742106939963
Frame 28: IoU = 0.5334826938047048
Frame 29: IoU = 0.5350448519488767
Frame 30: IoU = 0.5370083892670747
Frame 31: IoU = 0.5369891655578787
Frame 32: IoU = 0.5394895450705696
Frame 33: IoU = 0.5436135395367386
Frame 34: IoU = 0.5460592373453105
Frame 35: IoU = 0.5454355843343225
Frame 36: IoU = 0.5448818599723804
Frame 37: IoU = 0.5450799676415133
Frame 38: IoU = 0.5468233323045721
Frame 39: IoU = 0.5430690120089048
Frame 40: IoU = 0.5432275703922563
Frame 41: IoU = 0.5399322887961577
Frame 42: IoU = 0.542671147105649
Frame 43: IoU = 0.5439240219525704
Frame 44: IoU = 0.5439061293421144
Frame 45: IoU = 0.5435348776133588
Frame 46: IoU = 0.5459609679547096
Frame 47: IoU = 0.5459566111896801
Frame 48: IoU = 0.5461282370897199
Frame 49: IoU = 0.5485249761591914
Frame 50: IoU = 0.5495105532593091
Frame 51: IoU = 0.5499644254160984
Frame 52: IoU = 0.5500094941929781
Frame 53: IoU = 0.5507593200641072
Frame 54: IoU = 0.5552424257776877
Frame 55: IoU = 0.5546818501921625
```

```
Frame 56: IoU = 0.55600050530571
Frame 57: IoU = 0.5560065856704893
Frame 58: IoU = 0.5562798127203036
Frame 59: IoU = 0.5565358164750766
Frame 60: IoU = 0.5555871161871705
Frame 61: IoU = 0.5547134410526705
Frame 62: IoU = 0.5554236542737553
Frame 63: IoU = 0.5540653376474272
Frame 64: IoU = 0.5536333241311616
Frame 65: IoU = 0.5518879329605519
Frame 66: IoU = 0.553961742336944
Frame 67: IoU = 0.5549313300623097
Frame 68: IoU = 0.5590581861362625
Frame 69: IoU = 0.5598362324832314
Frame 70: IoU = 0.5592956921841329
Frame 71: IoU = 0.5602005458280949
Frame 72: IoU = 0.5628557583787231
Frame 73: IoU = 0.5639060871697419
Frame 74: IoU = 0.5651651816815165
Frame 75: IoU = 0.5661361091222853
Frame 76: IoU = 0.5674050312652378
Frame 77: IoU = 0.5637686873966481
Frame 78: IoU = 0.5625878400419498
Frame 79: IoU = 0.5609343951459992
Frame 80: IoU = 0.56291322376769
Frame 81: IoU = 0.5607440406787463
Frame 82: IoU = 0.5584850700646105
Frame 83: IoU = 0.5562781137358873
Frame 84: IoU = 0.5554571262949806
Frame 85: IoU = 0.5506951229561267
Frame 86: IoU = 0.5484944069628201
Frame 87: IoU = 0.5493079888903212
Frame 88: IoU = 0.5518090553038882
Frame 89: IoU = 0.5489663757205732
Frame 90: IoU = 0.5476339770922164
Frame 91: IoU = 0.5454800397385371
Frame 92: IoU = 0.540903617894601
Frame 93: IoU = 0.5382464675585987
Frame 94: IoU = 0.5330537368373413
Frame 95: IoU = 0.5316731841552611
Frame 96: IoU = 0.5300854664704976
Frame 97: IoU = 0.5243036067482839
Frame 98: IoU = 0.5227712537711696
Frame 99: IoU = 0.521119647945749
Frame 100: IoU = 0.5216130669050668
Frame 101: IoU = 0.5214466372402388
Frame 102: IoU = 0.5182364066048927
Frame 103: IoU = 0.5181097024769163
Frame 104: IoU = 0.5213349079030106
Frame 105: IoU = 0.521949394494175
Frame 106: IoU = 0.5186936257251115
Frame 107: IoU = 0.5191713152254868
Frame 108: IoU = 0.5196248950946885
Frame 109: IoU = 0.5194500122441206
Frame 110: IoU = 0.5185531958311201
Frame 111: IoU = 0.5173569518916254
```

```
Frame 112: IoU = 0.5147636202338922
Frame 113: IoU = 0.5107077196296771
Frame 114: IoU = 0.5070112812200166
Frame 115: IoU = 0.5046701684240007
Frame 116: IoU = 503117869814003
Frame 117: IoU = 0.5047309143859023
Frame 118: IoU = 0.5028945963580284
Frame 119: IoU = 0.5028938201980949
Frame 120: IoU = 0.5089165679893873
Frame 121: IoU = 513647088928496
Frame 122: IoU = 0.5151622548902813
Frame 123: IoU = 0.5156223858122804
Frame 124: IoU = 0.5164620070916922
Frame 125: IoU = 517658999656701
Frame 126: IoU = 0.5183745689008847
Frame 127: IoU = 519604963433641
Frame 128: IoU = 0.5170023189065787
Frame 129: IoU = 0.5168413654918731
Frame 130: IoU = 0.5165711929286682
Frame 131: IoU = 0.5161091289535102
Frame 132: IoU = 517092565106728
Frame 133: IoU = 517839733682083
Frame 134: IoU = 0.5198767306886499
Frame 135: IoU = 0.5184813787685826
Frame 136: IoU = 0.5166649277868034
Frame 137: IoU = 0.5184927352636249
Frame 138: IoU = 0.5157448726617083
Frame 139: IoU = 0.5170211998412717
Frame 140: IoU = 0.5191461572260041
Frame 141: IoU = 0.5205462542864282
Frame 142: IoU = 0.5185890408901837
Frame 143: IoU = 515972421889975
Frame 144: IoU = 0.5103236607142857
Frame 145: IoU = 0.5093368039837031
Frame 146: IoU = 0.5100417850079555
Frame 147: IoU = 0.5119644033295881
Frame 148: IoU = 512040063037915
Frame 149: IoU = 0.5088182502805468
Frame 150: IoU = 0.5097767743671785
Frame 151: IoU = 0.5111083746061685
Frame 152: IoU = 0.5064346897002947
Frame 153: IoU = 0.5047731660954035
Frame 154: IoU = 0.5049266844583046
Frame 155: IoU = 0.5044465025941678
Frame 156: IoU = 50472490550189
Frame 157: IoU = 0.5037195459372149
Frame 158: IoU = 0.5069260966444644
Frame 159: IoU = 0.5092991013813155
Frame 160: IoU = 0.5082307680747787
Frame 161: IoU = 0.5084986516273566
Frame 162: IoU = 0.5042312968872424
Frame 163: IoU = 0.5048416786084746
Frame 164: IoU = 0.5037317576023119
Frame 165: IoU = 50270493101007
Frame 166: IoU = 0.5036510685401082
Frame 167: IoU = 0.5059792395344767
```

```
Frame 168: IoU = 0.5077899596076169
Frame 169: IoU = 0.5096686396158692
Frame 170: IoU = 0.5121575530861481
Frame 171: IoU = 0.5144215815288911
Frame 172: IoU = 0.5099468007077272
Frame 173: IoU = 0.5084923953220627
Frame 174: IoU = 0.5038566284659393
Frame 175: IoU = 0.49989927969722864
Frame 176: IoU = 0.4957761239044274
Frame 177: IoU = 0.49541074726198964
Frame 178: IoU = 0.4927272395725696
Frame 179: IoU = 0.49098186684590905
Frame 180: IoU = 0.49641457378302506
Frame 181: IoU = 0.4992486068233153
Frame 182: IoU = 0.501919406118504
Frame 183: IoU = 0.501598062368277
Frame 184: IoU = 0.5006725659446385
Frame 185: IoU = 0.4985142843324255
Frame 186: IoU = 0.49615286515898654
Frame 187: IoU = 0.49670685029947415
Frame 188: IoU = 0.49524041684441467
Frame 189: IoU = 0.49719938718510237
Frame 190: IoU = 0.49884771061118904
Frame 191: IoU = 0.503567600109347
Frame 192: IoU = 0.4985495412252616
Frame 193: IoU = 0.5033481526904798
Frame 194: IoU = 0.5035777222494453
Frame 195: IoU = 0.5046186529664383
Frame 196: IoU = 0.5076867321642841
Frame 197: IoU = 0.5074558678410876
Frame 198: IoU = 0.5063093168954068
Frame 199: IoU = 0.5041992329724744
Frame 200: IoU = 0.5047747031736305
Frame 201: IoU = 0.5080096706900917
Frame 202: IoU = 0.5063647564383809
Frame 203: IoU = 0.5080773866235472
Frame 204: IoU = 0.5082499120430373
Frame 205: IoU = 0.508762438876316
Frame 206: IoU = 0.5105071616574036
Frame 207: IoU = 0.5115315547587712
Frame 208: IoU = 0.5118688714754376
Frame 209: IoU = 0.513874465231332
Frame 210: IoU = 0.5105551235540978
Frame 211: IoU = 0.5146562897443914
Frame 212: IoU = 0.5153209078383575
Frame 213: IoU = 0.5138466906920638
Frame 214: IoU = 0.5137121718964229
Frame 215: IoU = 0.5138783111597687
Frame 216: IoU = 0.5164697905142571
Frame 217: IoU = 0.5159015101695016
Frame 218: IoU = 0.5149422511735292
Frame 219: IoU = 0.5157376643574155
Frame 220: IoU = 0.5133960641024504
Frame 221: IoU = 0.5140412143721003
```