Highway Lane Line Detection Using Hough Transforms and Canny Edge Detection

Proposed tools and techniques:

The color selection, region of interest selection, gray scaling, Gaussian smoothing, Canny Edge Detection and Hough Transform line detection are the main tools used here. The project goal is to develop and explore techniques on detection the line segments in an image/video streams, then average/extrapolate them and draw them onto the image for display the display purpose.

Initialization

In [1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
%matplotlib inline
```

In [2]:
```
#reading in an image
image= mpimg.imread('SouthernExpressWay_Sri Lanka/image2.jfif')
```

In [3]:
```
# Size of the image
image.shape
```

Out[3]: (576, 1024, 3)

In [4]:
```
type(image)
```

Out[4]: numpy.ndarray

In [5]:
```
plt.imshow(image)
```

Out[5]: <matplotlib.image.AxesImage at 0x1a66c5e1448>

Steps in Line Detection

1. Grayscale Transformation of the image. This will return an image with only one color channel.

In [6]:
```
import math
def grayscale(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

2. Applying a Gaussian Noise kerne

In [7]:
```
def gaussian_noise(img, kernel_size):
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
```

3. Canny Transformation of the image

In [8]:
```
def canny(img, low_threshold, high_threshold):
    return cv2.Canny(img, low_threshold, high_threshold)
```

4. Applying an image mask.

This mask only keeps the region of the image defined by the polygon formed from `vertices`. The rest of the image is set to black.

In [9]:
```
def region_of_interest(img, vertices):

    #defining a blank mask to start with
    mask = np.zeros_like(img)

    #defining a 3 channel or 1 channel color to fill the mask with dependi
    if len(img.shape) > 2:
        channel_count = img.shape[2]  # i.e. 3 or 4 depending on the image
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    #filling pixels inside the polygon defined by "vertices" with the fill
    cv2.fillPoly(mask, vertices, ignore_mask_color)

    #returning the image only where mask pixels are nonzero
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image
```

5. Drawing Lines on the image inplace

In [10]:

```
def draw_lines(img, lines, color=[255, 0, 0], thickness=12):

    # initialize lists to hold line formula values
    bLeftValues     = []  # b of left lines
    bRightValues    = []  # b of Right lines
    mPositiveValues = []  # m of Left lines
    mNegitiveValues = []  # m of Right lines

    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(img, (x1, y1), (x2, y2), color, thickness)

            # calculate slope and intercept
            m = (y2-y1)/(x2-x1)
            b = y1 - x1*m

            # threshold to check for outliers
            if m >= 0 and (m < 0.2 or m > 0.8):
                continue
            elif m < 0 and (m < -0.8 or m > -0.2):
                continue

            # seperate positive line and negative line slopes
            if m > 0:
                mPositiveValues.append(m)
                bLeftValues.append(b)
            else:
                mNegitiveValues.append(m)
                bRightValues.append(b)

    # Get image shape and define y region of interest value
    imshape = img.shape
    y_max   = imshape[0] # lines initial point at bottom of image
    y_min   = 330        # lines end point at top of ROI

    # Get the mean of all the lines values
    AvgPositiveM = mean(mPositiveValues)
    AvgNegitiveM = mean(mNegitiveValues)
    AvgLeftB     = mean(bLeftValues)
    AvgRightB    = mean(bRightValues)

    # use average slopes to generate line using ROI endpoints
    if AvgPositiveM != 0:
        x1_Left = (y_max - AvgLeftB)/AvgPositiveM
        y1_Left = y_max
        x2_Left = (y_min - AvgLeftB)/AvgPositiveM
        y2_Left = y_min
    if AvgNegitiveM != 0:
        x1_Right = (y_max - AvgRightB)/AvgNegitiveM
        y1_Right = y_max
        x2_Right = (y_min - AvgRightB)/AvgNegitiveM
        y2_Right = y_min

        # define average left and right lines
        cv2.line(img, (int(x1_Left), int(y1_Left)), (int(x2_Left), int(y2_
        cv2.line(img, (int(x1_Right), int(y1_Right)), (int(x2_Right), int(
```

```
def mean(list):
    return float(sum(list)) / max(len(list), 1)
```

6.Drawing Hough lines

This function takes Canny transformed image and returns an image with hough lines drawn.

In [11]:
```
def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), minL
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    draw_lines(line_img, lines)
    return line_img

def weighted_img(img, initial_img, α=0.8, β=1., λ=0.):
    return cv2.addWeighted(initial_img, α, img, β, λ)
```

Build a Lane Finding Pipeline

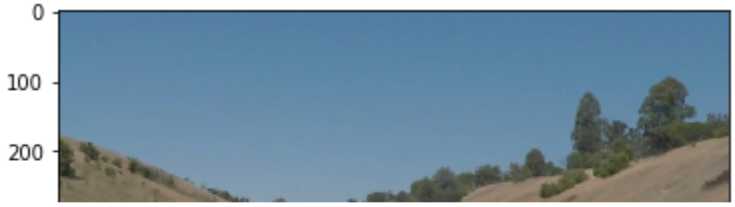
It is required to build a pipeline to run defined functions on test_images or videos. With this pipleline we can test the procedure by tuning various parameters particulaly the low and high Canny thresholds as well as the Hough lines parameters.

In [12]:
```
def process_image(imgName):
    # Get imag
    image = mpimg.imread('test_images/'+imgName)

    # Create gray image
    grayImg = grayscale(image)

    # Gaussian smoothing / blurring
    gblur = gaussian_noise(grayImg, 3)

    # Get Canny Img
    edges = canny(gblur, 50, 150)

    # Mask surroundings
    imgShape = image.shape
    polygon = np.array([ [(50, imgShape[0]), (460, 310), (490, 310), (imgS
    roi = region_of_interest(edges, polygon)

    # Hough Lines (img, rho, theta, threshold, min_line_len, max_line_gap)
    hLines = hough_lines(roi, 1, np.pi/180, 10, 18, 1)

    # Draw lines on edge image
    imgWithLines = weighted_img(hLines,image)

    plt.imshow(imgWithLines)
    return imgWithLines
```

Testing on Images

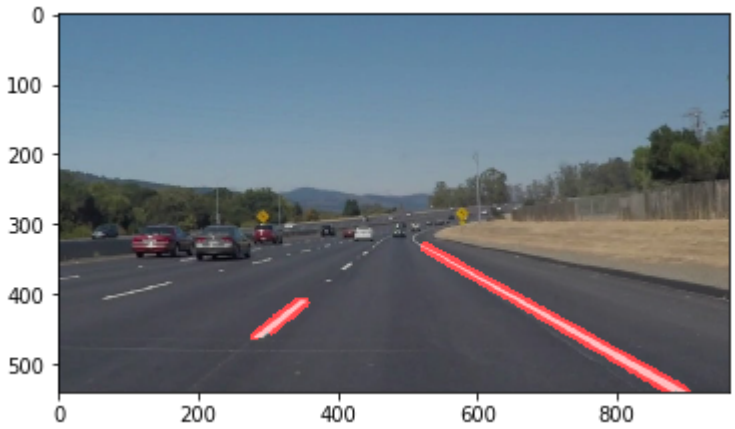
In [13]:
```
import os
files = os.listdir("test_images/")
print(files)

for file in files:
    print("\nProcessing ",file,"\n")
    pImage = process_image(file)
    mpimg.imsave("output_images/"+file[:-4]+"processed.png", pImage)
```

```
Processing  whiteCarLaneSwitch.jpg
```

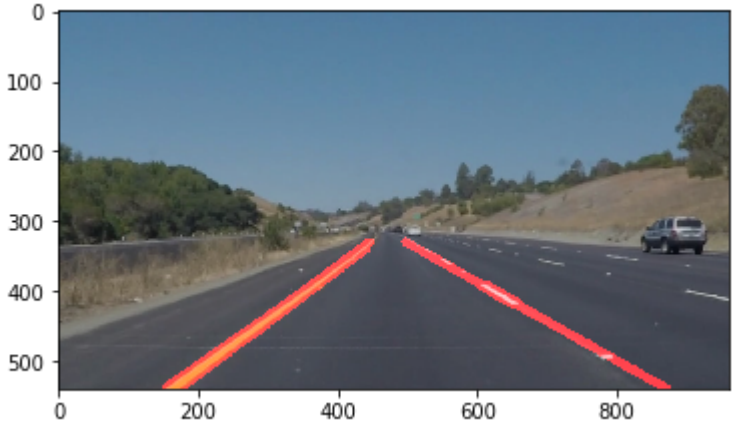


In [14]:
```
pImage = process_image('solidWhiteRight.jpg')
mpimg.imsave("output_images/"+"solidWhiteRightp.jpg", pImage)
```

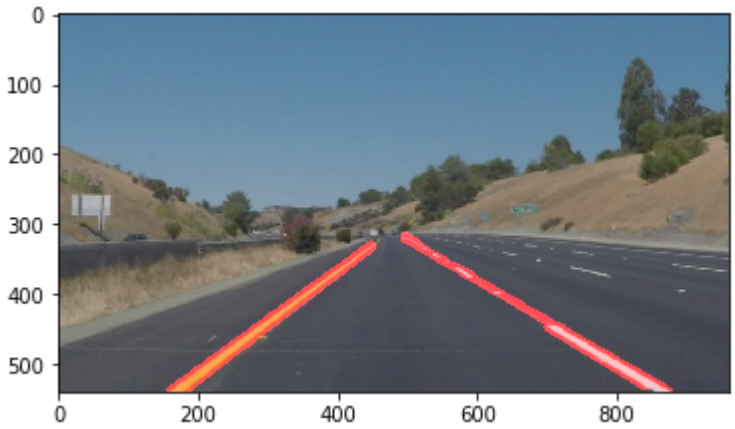
In [15]:
```
pImage = process_image('solidWhiteCurve.jpg')
mpimg.imsave("output_images/"+"solidWhiteCurveP.png", pImage)
```



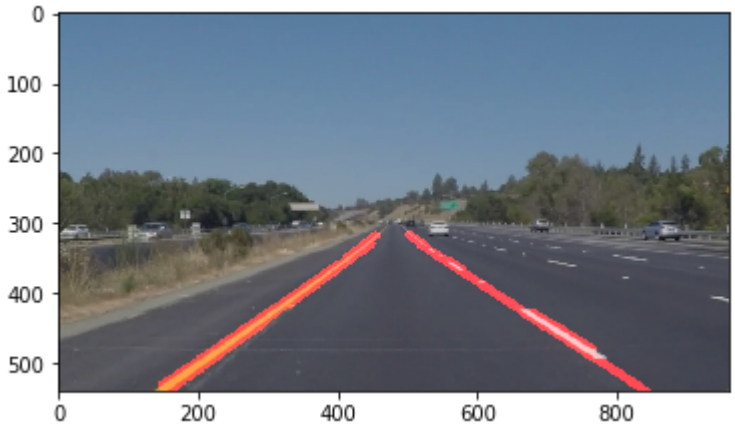
In [16]:
```
pImage = process_image('solidYellowCurve.jpg')
mpimg.imsave("output_images/"+"solidYellowCurveP.png", pImage)
```



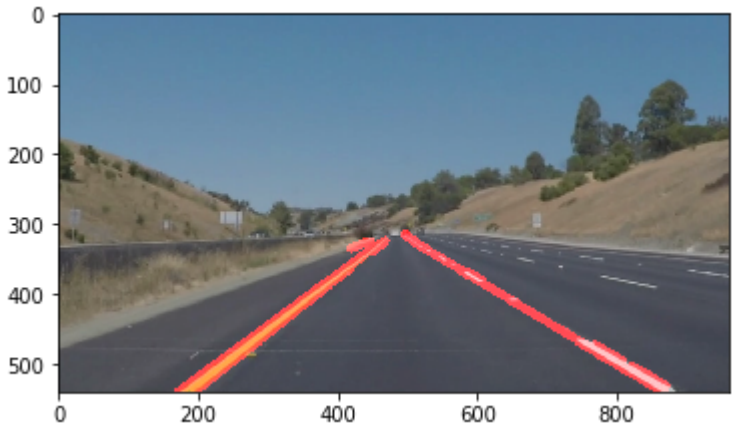
In [17]:
```
pImage = process_image('solidYellowCurve2.jpg')
mpimg.imsave("output_images/"+"solidYellowCurve2P.png", pImage)
```



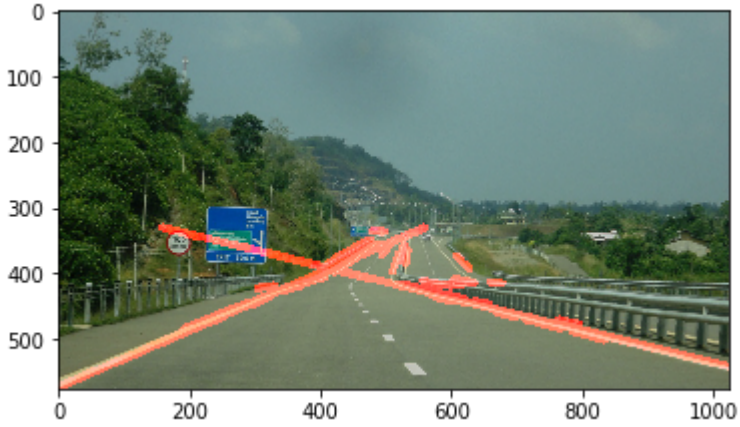
In [18]:
```
pImage = process_image('solidYellowLeft.jpg')
mpimg.imsave("output_images/"+"solidYellowLeftP.png", pImage)
```



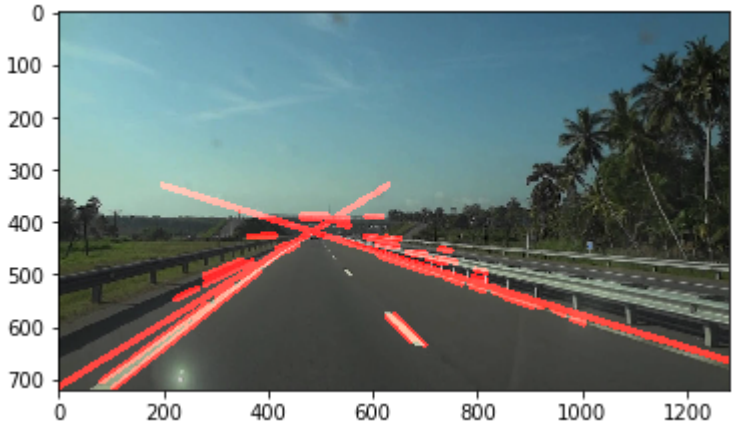
In [19]:
```
pImage = process_image('whiteCarLaneSwitch.jpg')
mpimg.imsave("output_images/"+"whiteCarLaneSwitchP.jpg", pImage)
```



In [20]:
```
pImage = process_image('image1.jfif')
mpimg.imsave("output_images/"+"image1P.jpg", pImage)
```

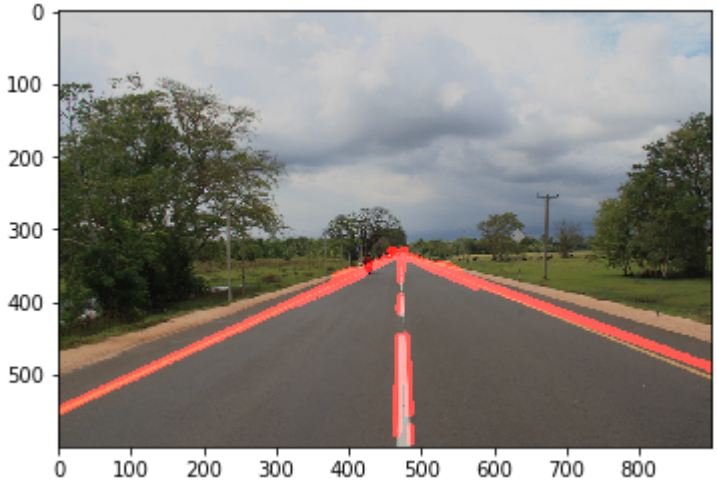


In [21]:
```
pImage = process_image('image2.jpg')
mpimg.imsave("output_images/"+"image8P.jpg", pImage)
```



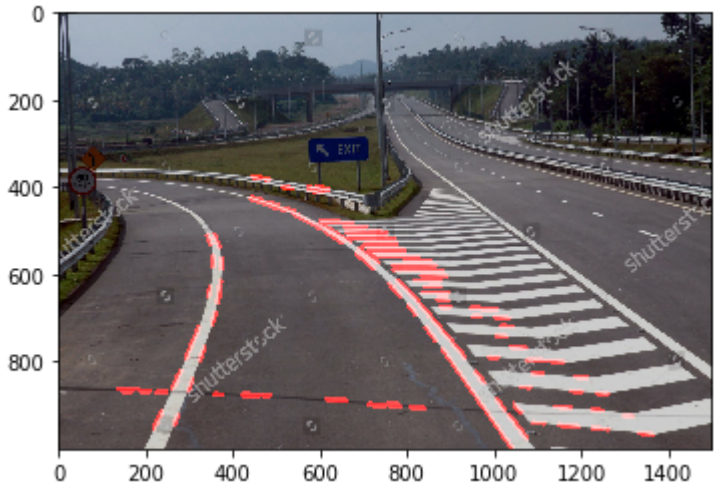
In [22]:
```
pImage = process_image('image3.jpg')
mpimg.imsave("output_images/"+"image3P.JPG", pImage)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: Run
timeWarning: divide by zero encountered in int_scalars
```



In [23]:
```
pImage = process_image('image4.JPG')
mpimg.imsave("output_images/"+"image7P.JPG", pImage)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: Run
timeWarning: divide by zero encountered in int_scalars
```



Testing on Videos

In [25]:
```
import imageio
from moviepy.editor import VideoFileClip
from IPython.display import HTML
```

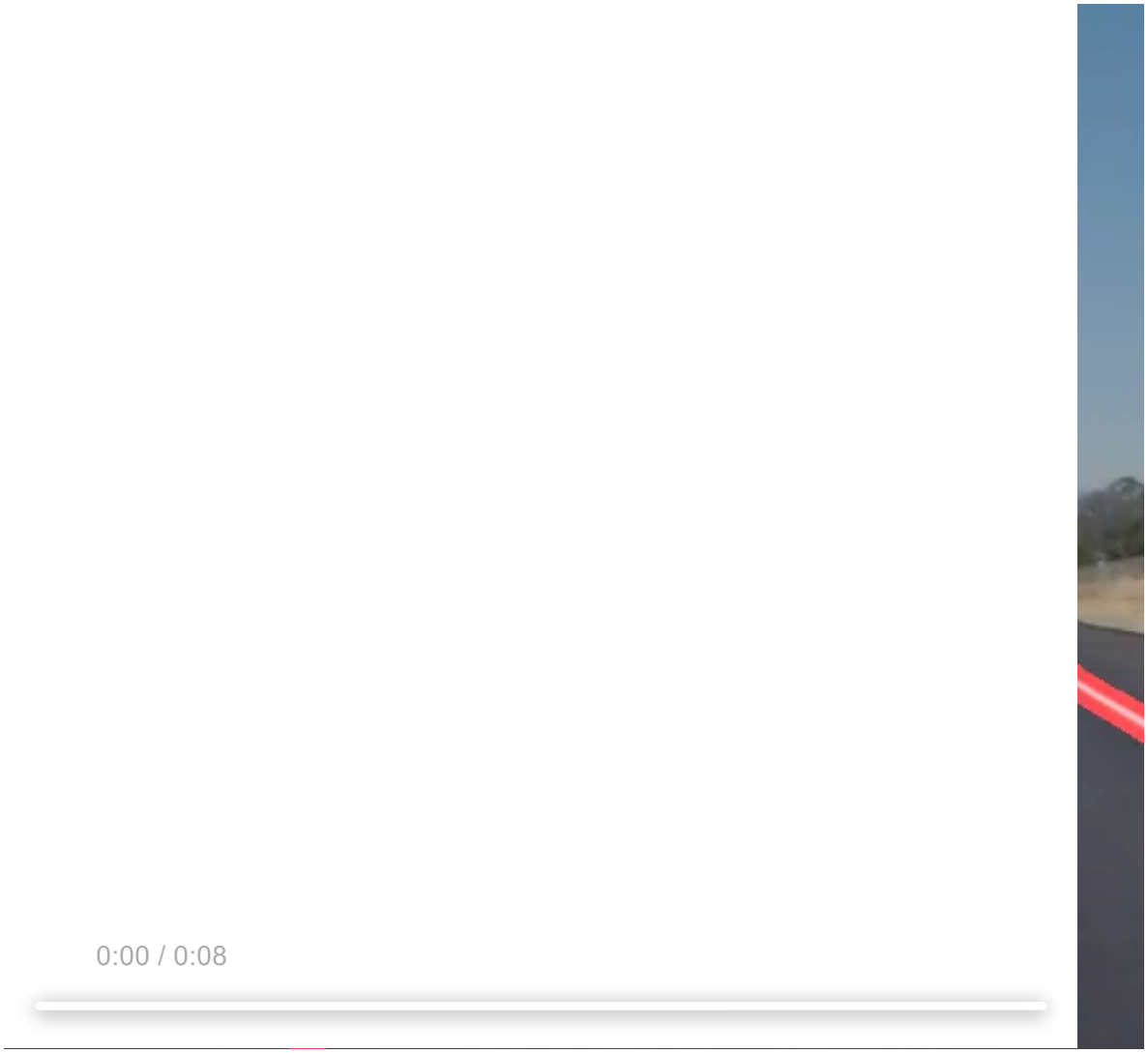
In [26]:
```
def process_image(image):
    # grayscale the image
    gray = grayscale(image)

    # gaussian blur
    blur_gray = gaussian_noise(gray,5)

    # canny edge detection
    edges = canny(blur_gray, 50, 150)

    # create region of interest
    imshape = image.shape
    vertices = np.array([[(50,imshape[0]),(460, 310), (490, 310), (imshape
    masked_edges = region_of_interest(edges,vertices)

    # hough tranform
    line_image = hough_lines(masked_edges, 1, np.pi/180, 22, 18, 1)

    # draw lines
    lines_edges = weighted_img(line_image,image)

    return lines_edges
```

In [27]:
```
white_output = 'output_videos/solidWhiteRightP.mp4'
clip1 = VideoFileClip("test_videos/solidWhiteRight.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

```
Moviepy - Building video output_videos/solidWhiteRightP.mp4.
Moviepy - Writing video output_videos/solidWhiteRightP.mp4



Moviepy - Done !
Moviepy - video ready output_videos/solidWhiteRightP.mp4
Wall time: 7.52 s
```

In [28]:
```
HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(white_output))
```

Out[28]:

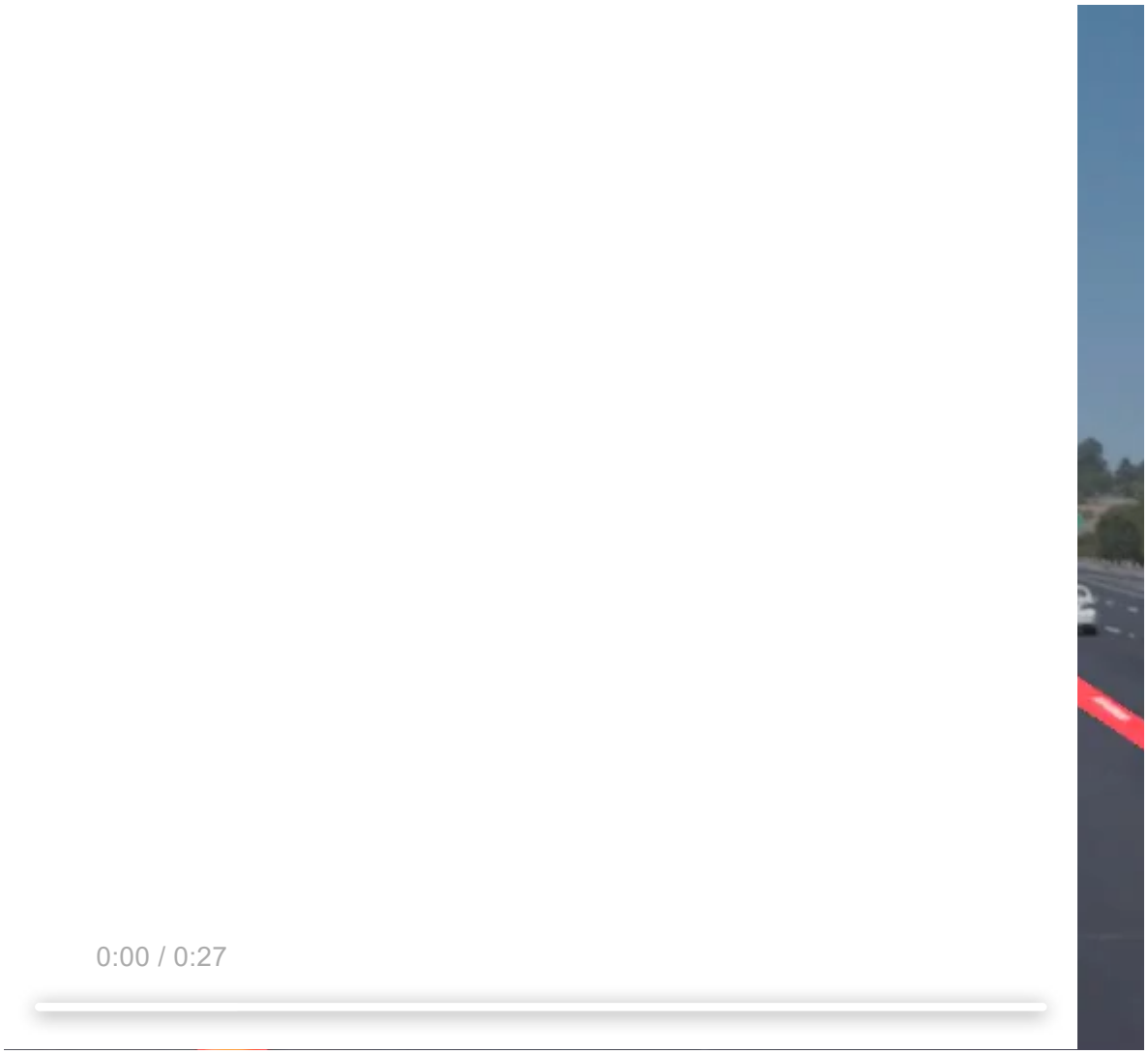


In [29]:
```
white_output = 'output_videos/solidYellowLeftP.mp4'
clip1 = VideoFileClip("test_videos/solidYellowLeft.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

```
Moviepy - Building video output_videos/solidYellowLeftP.mp4.
Moviepy - Writing video output_videos/solidYellowLeftP.mp4



Moviepy - Done !
Moviepy - video ready output_videos/solidYellowLeftP.mp4
Wall time: 19.9 s
```

In [30]:
```
HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(white_output))
```

Out[30]:



In [31]:
```
white_output = 'output_videos/SouthernE1cut1P.mp4'
clip1 = VideoFileClip("test_videos/SouthernE1cut1.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

```
Moviepy - Building video output_videos/SouthernE1cut1P.mp4.
Moviepy - Writing video output_videos/SouthernE1cut1P.mp4



Moviepy - Done !
Moviepy - video ready output_videos/SouthernE1cut1P.mp4
Wall time: 10.9 s
```

In [32]:
```
white_output = 'output_videos/SouthernE1cut2P.mp4'
clip1 = VideoFileClip("test_videos/SouthernE1cut2.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

```
Moviepy - Building video output_videos/SouthernE1cut2P.mp4.
Moviepy - Writing video output_videos/SouthernE1cut2P.mp4



Moviepy - Done !
Moviepy - video ready output_videos/SouthernE1cut2P.mp4
Wall time: 15.5 s
```

In [ ]:
```
white_output = 'output_videos/SouthernE1cut4P.mp4'
clip1 = VideoFileClip("test_videos/SouthernE1cut4.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

In [ ]:
```
white_output = 'output_videos/SouthernE1cut5P.mp4'
clip1 = VideoFileClip("test_videos/SouthernE1cut5.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

In [ ]:
```
white_output = 'output_videos/SouthernE2cut1P.mp4'
clip1 = VideoFileClip("test_videos/SouthernE2cut1.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

In [ ]:
```
white_output = 'output_videos/SouthernE2cut2P.mp4'
clip1 = VideoFileClip("test_videos/SouthernE2cut2.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

In [ ]:
```
white_output = 'output_videos/SouthernE2cut3P.mp4'
clip1 = VideoFileClip("test_videos/SouthernE2cut3.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

In [ ]:
```
white_output = 'output_videos/SouthernE4cut1P.mp4'
clip1 = VideoFileClip("test_videos/SouthernE4cut1.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

In [ ]:
```
white_output = 'output_videos/SouthernE4cut2P.mp4'
clip1 = VideoFileClip("test_videos/SouthernE4cut2.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

In [ ]:
```
white_output = 'output_videos/SouthernE4cut3P.mp4'
clip1 = VideoFileClip("test_videos/SouthernE4cut3.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects co
%time white_clip.write_videofile(white_output, audio=False)
```

In [ ]:
```

```

In [ ]:
```

```