Import Packages

```
#importing some useful packages

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import numpy as np

import cv2

%matplotlib inline
```

Read in an Image

```
#reading in an image

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

#printing out some stats and plotting

print('This image is:', type(image), 'with dimensions:', image.shape)

plt.imshow(image)    # if you wanted to show a single color channel image called 'gray', for example, call as plt.imshow(gray, cmap='gray')
```

Helper Functions

```
def grayscale(img):

    """Applies the Grayscale transform

    This will return an image with only one color channel

    but NOTE: to see the returned image as grayscale

    (assuming your grayscaled image is called 'gray')

    you should call plt.imshow(gray, cmap='gray')"""

    image_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

    return image_gray
```

改了這段，將 GRB 格式圖像轉換成灰度圖像，然後將其輸出圖像定義成 image_gray

```
    # Or use BGR2GRAY if you read an image with cv2.imread()
    # return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

def canny(img, low_threshold, high_threshold):
    """Applies the Canny transform"""
    image_canny = cv2.Canny(img, low_threshold, high_threshold)
    return image_canny

def gaussian_blur(img, kernel_size):
    """Applies a Gaussian Noise kernel"""
    image_blur = cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
    return image_blur
```

在這段程式碼中，我使用了 OpenCV 庫中的`cv2.GaussianBlur()`函數，並傳遞了圖像和高斯核的大小作為參數，這樣做的目的是降低圖像的噪點和細節，從而改善後續處理的效果，然後將其輸出圖像定義成 image_blur

```
def region_of_interest(img, vertices):
    """
    Applies an image mask.

    Only keeps the region of the image defined by the polygon
    formed from `vertices`. The rest of the image is set to black.
    `vertices` should be a numpy array of integer points.
    """
    # Define a blank mask to start with
    mask = np.zeros_like(img)
    # Define color according to image channels
    if len(img.shape) > 2:
        ignore_mask_color = (255,) * img.shape[2]
    else:
        ignore_mask_color = 255

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

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

def draw_lines(img, lines, color=[255, 0, 0], thickness=2):
    """
    NOTE: this is the function you might want to use as a starting point once you want to
    average/extrapolate the line segments you detect to map out the full
    extent of the lane (going from the result shown in raw-lines-example.mp4
    to that shown in P1_example.mp4).

    Think about things like separating line segments by their
    slope ((y2-y1)/(x2-x1)) to decide which segments are part of the left
    line vs. the right line.  Then, you can average the position of each of
    the lines and extrapolate to the top and bottom of the lane.

    This function draws `lines` with `color` and `thickness`.
    Lines are drawn on the image inplace (mutates the image).
    If you want to make the lines semi-transparent, think about combining
    this function with the weighted_img() function below
    """
    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(img, (x1, y1), (x2, y2), color, thickness)

def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):
    """
    `img` should be the output of a Canny transform.

    Returns an image with hough lines drawn.
    """
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), minLineLength=min_line_len, maxLineGap=max_line_gap)
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    draw_lines(line_img, lines)
    return line_img

# Python 3 has support for cool math symbols.
```

```
def weighted_img(img, initial_img, α=0.8, β=1., γ=0.):
    """
    `img` is the output of the hough_lines(), An image with lines drawn on it.
    Should be a blank image (all black) with lines drawn on it.

    `initial_img` should be the image before any processing.

    The result image is computed as follows:

    initial_img * α + img * β + γ
    NOTE: initial_img and img must be the same shape!
    """
    result = cv2.addWeighted(initial_img, α, img, β, γ)
    return result
```

這一行使用 OpenCV 中的`cv2.addWeighted()`函數將兩張圖像合成。它將兩張圖像按照指定的比例加權相加，然後返回合成後的圖像。

Test Images

```
import os
os.listdir("test_images/")
```

Build a Lane Finding Pipeline

```
# TODO: Build your pipeline that will draw lane lines on the test_images
# then save them to the test_images_output directory.
```

Test on Videos

```
pip install moviepy
```

```
# Import everything needed to edit/save/watch video clips
from moviepy.editor import VideoFileClip
from IPython.display import HTML

def process_image(image):
    rho = 1
    theta = np.pi/180

    # 將圖像轉換為灰度圖
    image_gray = grayscale(image)

    # 使用 Canny 邊緣檢測
    image_canny = canny(image_gray, 200, 250)    # 調整 Canny 邊緣檢測的閾值

    # 高斯模糊處理
    image_blur = gaussian_blur(image_canny, 3) # 增加高斯模糊的內核大小

    # 定義 ROI 的頂點位置
    imshape = image.shape
    vertices = np.array([[(50, imshape[0]), (imshape[1]*0.45, imshape[0]*0.6),
(imshape[1]*0.55, imshape[0]*0.6), (imshape[1]-50, imshape[0])]], dtype=np.int32)

    # 應用感興趣區域
    masked_image = region_of_interest(image_blur, vertices)

    # 霍夫變換
    lines = hough_lines(masked_image, rho, theta, 50, 10, 100)

    # 將車道線圖像與原始圖像進行融合
    result = weighted_img(lines, image, α=0.8, β=1, γ=0.)

    return result

white_output = 'test_videos_output/solidWhiteRight.mp4'
## To speed up the testing process you may want to try your pipeline on a shorter subclip of the video
## To do so add .subclip(start_second,end_second) to the end of the line below
## Where start_second and end_second are integer values representing the start and end of the subclip
## You may also uncomment the following line for a subclip of the first 5 seconds
##clip1 = VideoFileClip("test_videos/solidWhiteRight.mp4").subclip(0,5)
clip1 = VideoFileClip("test_videos/solidWhiteRight.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects color images!!
```

```
%time white_clip.write_videofile(white_output, audio=False)


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

Improve the draw_lines() function

```
yellow_output = 'test_videos_output/solidYellowLeft.mp4'
## To speed up the testing process you may want to try your pipeline on a shorter subclip of the video
## To do so add .subclip(start_second,end_second) to the end of the line below
## Where start_second and end_second are integer values representing the start and end of the subclip
## You may also uncomment the following line for a subclip of the first 5 seconds
##clip2 = VideoFileClip('test_videos/solidYellowLeft.mp4').subclip(0,5)
clip2 = VideoFileClip('test_videos/solidYellowLeft.mp4')
yellow_clip = clip2.fl_image(process_image)
```

```
%time yellow_clip.write_videofile(yellow_output, audio=False)


HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(yellow_output))
```

Optional Challenge

```
challenge_output = 'test_videos_output/challenge.mp4'
## To speed up the testing process you may want to try your pipeline on a shorter subclip of the video
## To do so add .subclip(start_second,end_second) to the end of the line below
## Where start_second and end_second are integer values representing the start and end of the subclip
## You may also uncomment the following line for a subclip of the first 5 seconds
##clip3 = VideoFileClip('test_videos/challenge.mp4').subclip(0,5)
clip3 = VideoFileClip('test_videos/challenge.mp4')
challenge_clip = clip3.fl_image(process_image)
%time challenge_clip.write_videofile(challenge_output, audio=False)


HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>

""".format(challenge_output))
```

影片 ：

challenge : https://youtu.be/Bho707qcvP0

solidWhiteRight : https://youtu.be/q41iG0DRpEM

solidYellowLeft : https://youtu.be/MVJa2bFkwT0